Seminararbeit: Hardware für Echtzeitsysteme

Konstantin Müller
Friedrich-Alexander-Universität Erlangen-Nürnberg
Nürnberg, Germany

ZUSAMMENFASSUNG

Dieses Seminarthema behandelt das Thema der Worst Case Execution Time (WCET) Analyse von Echtzeitsystemen mit harten Zeitanforderungen. Der Schwerpunkt liegt hierbei auf den Aspekten von moderner Hardware bei diesen Berechnungen. Nach einer Übersicht über das Phänomen von sogenannten Zeitanomalien bei der WCET-Analyse, wird ein Prozessordesign beschrieben, welches explizit für die WCET-Analyse entwickelt wurde und diese Zeitanomalien verhindert [6] [3].

ACM Reference Format:
Konstantin Müller. 2024. Seminararbeit: Hardware für Echtzeitsysteme. *Proc. ACM Meas. Anal. Comput. Syst.* 37, 4, Article 111 (August 2024), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 EINLEITUNG

Thema dieser Seminararbeit sind Hardwarebetrachtungen in Echtzeitsystemen. Die beiden Papiere, die in dieser Arbeit behandelt werden, stehen im Kontext der sogenannten Worst Case Execution Time-Analyse. In mehreren Industrieanwendungen wie der Luftfahrt oder der Automobilbranche ist es wichtig, harte Laufzeitgrenzen für Systeme anzugeben, deren Nichteinhaltung zu katastrophalen Verhalten führen können. Die Worst Case Execution Time-Analyse (WCET) eines Programms dient der Angabe von oberen Schranken für solche Systeme. Im Kontext von WCET-Analysen behandeln Reineke et al. [6] das Problem sogenannter Zeitanomalien (timing anomalies). Randfälle, welche die Analyse der WCET erschweren. Ziel dieser Seminararbeit ist es, einen Überblick über die WCET-Analyse zu geben und das Problem der Zeitanomalien in diesen Kontext einzuordnen. Zeitanomalien sollen anhand von Papier 1: [6] beschrieben und eingeteilt werden. Anschließend soll ein Prozessordesign von Reineke und Hahn [3] beschrieben werden, welches Zeitanomalien verhindert und somit die WCET-Berechnung erleichtert.

2 WCET

Um die behandelten Papiere einordnen zu können und wichtige Begrifflichkeiten zu klären, soll zuerst eine Übersicht über die Methoden der WCET-Analyse gegeben werden. Im Fokus stehen hier die erwähnten Probleme der Zeitanomalien sowie Hardwarebetrachtungen bei der Laufzeitberechnung. Letztere sind für da Prozessordesign aus der zweiten Arbeit wichtig.

2.1 WCET Generell

Als wichtigste Übersicht über das Thema der WCET-Analyse gilt die Arbeit: 'The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools' von Wilhelm et al. [9]. Dieses teilt die WCET-Analyse in statische- und messungsbasierte Methoden ein. 'Measurment-based methods' nutzen Programmausführungszeiten und Messungen, um obere Laufzeitschranken anzugeben. Die Arbeiten aus diesem Seminar beziehen sich auf die statische Analyse, von der im folgenden bei der WCET-Berechnung ausgegangen wird. Das Ziel der statischen Laufzeitberechnung ist es, obere Schranken für die Programmlaufzeit anzugeben, ohne das Programm wirklich auszuführen. Hierfür wird eine Menge möglicher Kontrollpfade mit einem Hardwaremodell kombiniert, um die höchste worst-case Laufzeit zu berechnen. Vor der eigentlichen WCET-Berechnung gibt es nach [9] zwei Schritte. In einer **Kontrollfluss-Analyse** werden die möglichen Programmpfade berechnet, welche in der WCET Berechnung betrachtet werden müssen. Die **Hardware-Analyse** fügt mögliche Hardwarezustände der darunter liegenden Architektur ein, welche Einfluss auf die Laufzeit haben. Beispielsweise der Zustand der Speicherungsstrukturen oder Pipelinezustände.

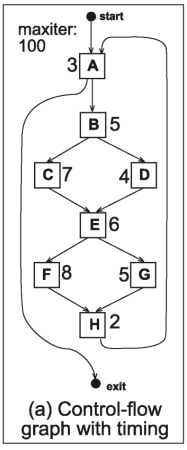

Abbildung 1: Kontrollflussgraph [9]

2.2 Kontrollfluss-Analyse

Der erste Schritt der WCET-Berechnung besteht darin, alle möglichen Kontrollpfade des Programmablaufes zu bestimmen. Dies geschieht in der Kontrollfluss-Analyse, die einen Control-Flow-Graph(CFG) mit der Menge aller Kontrollflüsse ausgibt. Der in abb 1 dargestellte cfg besteh aus Gründblöcken, welche aus Codestücken

Author's address: Konstantin Müller, Friedrich-Alexander-Universität Erlangen-Nürnberg, Nürnberg, Germany.

ACM 2476-1249/2024/8-ART111
https://doi.org/10.1145/nnnnnnn.nnnnnnn

bestehen und kannten, welche Übergänge zwischen den Blöcken darstellen. Jeder Grundblock des CFG verbraucht eine gewisse Zeit, abhängig von der darunterliegenden Hardware.[7].

Bei modernen Prozessorarchitekturen ist zu beachten, dass aufgrund von Optimierungen wie zB. der Out-of-Order Execution, der echte Programmablauf von CFG abweichen kann [9].

2.3 Hardware-Analyse

Um eine obere Schranke für Laufzeiten der Grundblöcke, also einzelner Codeabschnitte des CFG zu errechnen, ist es wichtig, Hardwareeinflüsse zu beachten. Ein großteil der Performance moderner Prozessoren hängt von Optimierungsmethoden wie Caching, Pipelining, branch-prediction und Out-of Order Execution ab. Die Ausführungsszeit einer Instruktion hängt dadurch stark vom aktuellen Zustand der darunterliegenden Hardware ab. Die Laufzeit einer Instruktion B hängt maßgeblich davon ab, welchen Hardwarezustand eine Instruktion A, die davor läuft, erzeugt [9]. Da es das Ziel der WCET-Analyse ist, obere Schranken anzugeben und die tatsächliche Laufzeit nicht höher als diese Grenze sein darf, muss von den schlechtmöglichsten Hardwarezuständen ausgegangen werden, dem local worst-case.

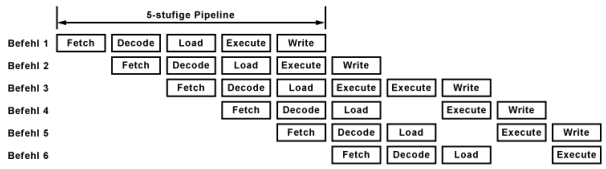

Abbildung 2: Pipelinestufen

2.3.1 Hardware Crashkurs. Speicherungsstrukturen und Pipelining sind Themen, die in diesem Kapitel als vorausgesetzt angenommen werden. Zur kurzen Wiederholung sind in Abbildung 2 die Pipelinestufen abgebildet, in welche eine Assemblerinstruktion aufgeteilt wird. Eine normale Pipelinearchitektur schafft es dadurch im Optimalfall in jedem Takt ein neues Ergebnis zu produzieren. Gibt es jedoch Datenabhängigkeiten, muss die Pipeline blockiert werden und der Durchsatz fällt ab. Moderne Prozessoroptimierungen, die in der Hardware-Analyse der WCET-Berechnung betrachtet werden müssen und in beiden behandelten Arbeiten vorkommen sind Branch-Prediction und auch In-order und Out-of.order Arichitekturen.

Branch-Prediction:
Um die Prozessorpipeline bei einer Verzweigung nicht blockieren zu müssen, versuchen moderne Architekturen den Ausgang der Verzweigung vorherzusagen. Es werden bereits weitere Instruktionen in die Pipeline geladen, bevor der Ausgang der Verzweigung bekannt ist. Ist der vorhergesagte Ausgang falsch, müssen die Instruktionen aus der Pipeline gelöscht und die der anderen Verzweigung geladen werden. Für die Berechnung des local worst-case für diesen Block des CFG wird von einer falsch gewählten Branchprediction und dadurch längeren Ausführungszeit ausgegangen.
Out-of-Order Execution Eine weitere Optimierung, welche die Komplexität der Hardware und somit der Hardware-Analyse erhöht, sind Out-of-Order Prozessoren. In diesen können auf HW-Ebene Instruktionen in veränderter Reihenfolge als im Assemblercode ausgeführt werden, um Pipeline Stalls zu verhindern. Anstatt auf eine datenabhängige Instruktion zu warten, kann eine unabhängige Instruktion in die Pipeline geladen und ausgeführt werden. Die Instruktionsausführung geschieht daher Out-of-Order zur ursprünglichen Anordnung des Assemblercodes.

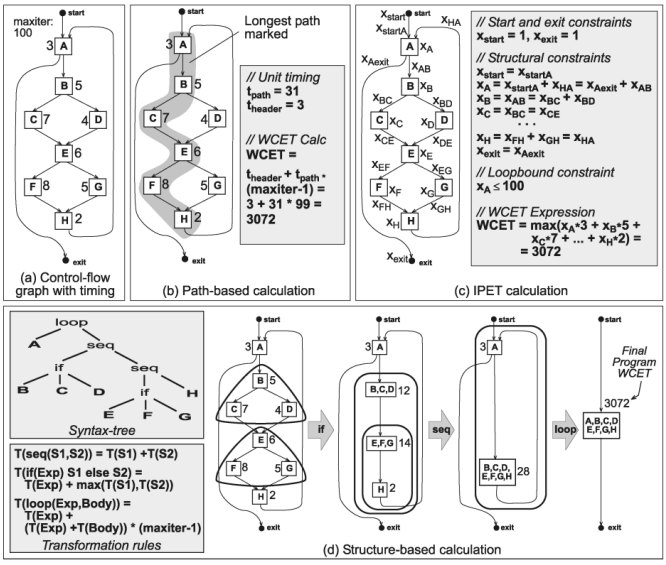

Abbildung 3: Möglichkeiten der WCET-Berechnung [9]

2.4 Laufzeitberechnung

Das WCET-Übersichtspapier [9] teilt die anschließende Berechnung der WCET in drei Kategorien ein.

- Pfadbasiert (Path-based Calculation)
- Strukturbasiert (Structure-Based Calculation)
- IPET

In Abbildung 3 sind die Berechnungsarten mit einem beispielhaften CFG gezeigt. In der 'Path-based Calculation' Abb.3(b) wird intuitiv der längste Ausführungspfad des CFG gesucht und für diesen die längste Laufzeit berechnet. In der Abbildung ist dieser längste Pfad in grau dargestellt. Nachteil dieser Methode ist, dass die Zahl der zu betrachtenden Pfade bei großen Programmen stark steigt [7]. Im Ansatz der 'structure-Based Calculation' Abb.3(d) wird eine obere Laufzeitgrenze berechnet, indem Instruktionslaufzeiten kombiniert und der CFG durchlaufen wird [9]. Die Implicit Path Enumeration Technique (IPET) Abb.3(c) transformiert den CFG in ein ganzzahliges, lineares Optimierungsproblem. Zusätzlich werden Flussrestriktionen als Nebenbedingungen annotiert für Fälle der Programmausführung, welche nicht eintreten können. Diese Nebenbedingungen sind im grauen Kasten dargestellt. Obwohl eine exakte Berechnung dieses Problems schwer ist, erreichen moderne WCET-Berechner durch Optimierungen eine akzeptable Laufzeit [8].

2.5 Compositional timing analysis

Aufgrund moderner, komplexer Systeme kann die Berechnung der oberen Schranke für die Programmlaufzeit sehr komplex werden. Als Vereinfachung der WCET-Berechnung geht eine Vielzahl von Arbeiten, die auch in [3] genannt werden, von der sogenannten **Compositional timing analysis** aus. Diese teilt die Programmausführung in einzelne Schritte ein. Für diese Schritte werden dann lokale Worst Case Ausführungszeiten berechnet und die Laufzeiten aller Schritte danach aufaddiert. Die Autoren des ASIC-Papiers [9] haben diese Art der Berechnung schon in einer Arbeit behandelt und unter anderem eine formale Definition für compositional Tasks gegeben [4]. Vor allem in modernen Systemen mit Prozessoroptimierungen und Multicore-Systemen ist diese compositionality eine wichtige Hardware-Eigenschaft, welche für eine mögliche WCET-Berechnung nötig ist.

2.6 Zeitanomalien

Schon das Übersichtspapier [9] beschreibt das Problem sogenannter Zeitanomalien bei der Berechnung der WCET. Generell sind Zeitanomalien Fälle, in denen Umstände, die intuitiv zu einer schlechtmöglichsten Programmlaufzeit führen sollten, nicht die globale Worst-Case execution time darstellen. Ein deutliches Beispiel ist ein Cache Miss, der aufgrund von Schedulingeffekten eine kürzere Laufzeit als ein Cache Hit hat. Dies erzeugt offensichtlich Probleme für eine WCET-Berechnung, die für die längstmögliche Laufzeit nur von dem lokalen worst case scencario, also hier dem Cache Miss, ausgeht, den Fall des Cache Hit aber nicht betrachtet. Eine so berechnete obere Grenze wäre also falsch. Für die vorher beschriebene Berechnungsart der compositional timing analysis stellen die Zeitanomalien ein Problem dar, da es nicht mehr ausreicht, den lokalen Worst Case einzelner Programmabschnitte zu betrachten. So wie in der Arbeit 'Design and Analysis of SIC: A Provably Timing-Predictable Pipelined Processor Core' [3], ist es hier wichtig zu erwähnen, dass dieses Problem nur die statischen Berechnungsmethoden betrifft, welche nicht alle möglichen Fälle in ihrer Berechnung betrachten. Sogenannte 'fully-integrated timing anlysen' betrachten in ihrer Berechnung alle mögliche Kombinationen aus Prozessorzuständen [3]. Dieser Ansatz würde alle Systemzustände in die Berechnung einbeziehen, ist aber aufgrund der hohen Zahl an zu betrachtenden Zustände nicht praktikabel. 1999 waren Lundqvist et al. [5] die ersten Autoren, die Zeitanomalien beschrieben haben. In dieser Arbeit gingen Lundqvist et al. davon aus, dass Zeitanomalien nur in Out-of-Order Architekturen aufgrund der komplexeren Schedulingeffekte aufreten. Lundqvist und auch Reineke et al [2] beschreiben jedoch später, dass auch in simplen In-order Prozessoren Zeitanomalien auftreten können.

2.6.1 Einteilung: Das Problem von Zeitanomalien im Kontext der WCET Berechnung ist lange bekannt und wird auch schon in der frühen Übersichtsarbeit [9] genannt. Papier 1 dieses Seminarthemas [6] beschäftigt sich ausschließlich mit diesen Anomalien, gibt als erstes Arbeit eine formale Definition und teilt Zeitanomalien in drei Kategorien ein.

Scheduling Timing Anomalies:
Zeitanomalien können dann entstehen, wenn durch eine lokal längere Ausführung besseres Scheduling und so eine global kürzere Ausführungszeit entsteht. Diese ist in Abbildung 4 dargestellt.

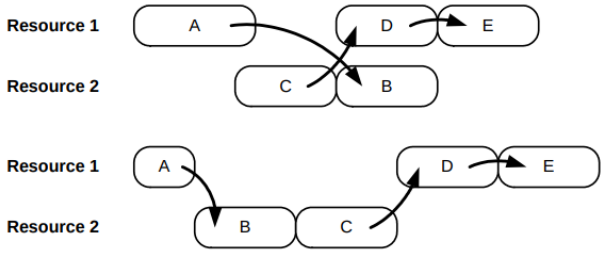

Abbildung 4: Scheduling anomaly [6]

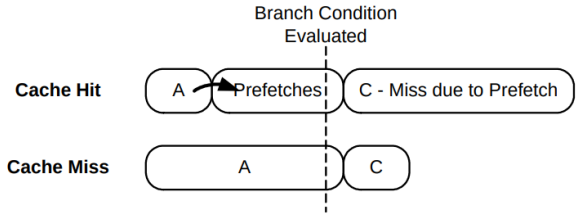

Abbildung 5: Speculative anomaly [6]

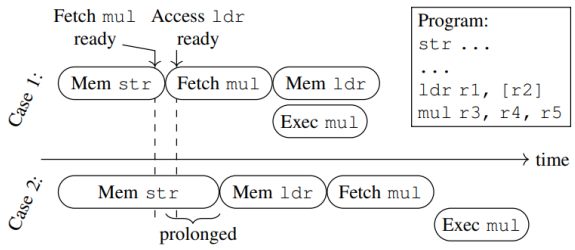

Abbildung 6: Effekt von unterschiedlichen Speicherzugriffen [3]

Durch eine längere lokale Ausführungszeit in Task A wird die wichtigere Task C nicht blockiert und die gesamte Ausführungszeit ist kürzer. Würde die WCET-Analyse also nur diesen lokalen worstcase betrachten, wäre die obere Grenze falsch.

Speculation Timing Anomalies:
Diese Art der Zeitanomalien bezieht sich auf die vorher beschriebene Branch-Prediction. Wird eine Verzweigung im Code falsch vorhergesagt, können die Instruktionen, die ausgeführt werden, bevor die Verzweigung evaluiert wurde, den Zustand des Cache für folgende Instruktionen verschlechtern. Die Instruktionen aus der korrekten Verzweigunsrichtung brauchen nun länger, da benötigte Daten aus dem Cache verdrängt wurden. Eine Speculation Timing Anomalie tritt auf, wenn der lokal schlechte Fall eines Cache Miss dafür sorgt, dass weniger spekulativ ausgeführte Instruktionen aus der falschen Verzweigung ausgeführt werden können. Dadurch, dass der Zustand des Cache nun besser ist, ist die globale Ausführungszeit kürzer. So ein Fall ist in Abbildung 5 gezeigt, in dem eine längere Ausführung von Task A durch einen Cache Miss zu einer schnelleren Ausführung von C führt, da der Cache zustand durch spekulative Branch-prediction nicht verschlechtert wird.

Cache Timing Anomalies:

Diese Anomalien entstehen durch Sonderfälle im Cacheverhalten. Ein Beispiel dafür ist der Motorola ColdFire 5307 Prozessor, dessen Cache Ersetzungsstrategie dafür sorgt, dass nicht der lokale worst-case die global längste Ausführungszeit zur Folge hat.

Dominoeffekte:

In seiner Arbeit 'Timing Anomalies Reloaded' [1] definiert Gebhard die Effekte, welche Zeitanomalien auf die Programmlaufzeit haben. Als k-bound timing anomalies definiert er Zeitanomalienit-anomalien deren Effekte auf die Laufzeit begrenzt sind. In vielen Fällen nähert sich der Pfad, der eine Zeitanomalie hat der Laufzeit eines Pfades ohne Anomalie plus zusätzlicher Laufzeitgrenze p an. Bleibt dieses p über die Pfadlaufzeit konstant, spricht man von einer k-bounded Timing Anomaly. Ist die Konstante k bekannt, könnte eine WCET-Berechnung weiterhin nur von einem lokalen Worst case ausgehen und auf dessen Laufzeit die Konstante k aufaddieren. Unbegrenzte Zeitanomalien nennt Gebhard Anomalien mit einem Dominoeffekt. Zum einen nennt er einen konkreten Prozessor, in dem Dominoeffekte nach Zeitanomalien nachweislich auftreten. Zum anderen nennt er Cache Ersetzungsstrategien als mögliche Auslöser solcher Effekte.

2.7 Prozessor mit Zeitanomalie

Neben der Einordnung der Laufzeiteffekte von Zeitanomalien gibt Gebhard noch ein konkretes Beispiel eines Prozessors, welcher Zeitanomalien aufweist. Er analysiert hierfür das Cache-Verhalten des LEON2 Prozessors und zeigt, dass dieser trotz seiner einfachen Architektur speculation timing anomalies vorweist. Intuitiv sollten in LEON2 keine Anomalien auftreten. Der Prozessor hat keine spekulative Ausführung und ist In-order. Eine Instruktion kann eine andere in der Pipelinearchitektur somit nicht 'überholen'. Des weiteren ist die LRU Cacheersetzungsstrategie des LEON2 dafür bekannt 'fully timing compositional' zu sein. Gebhard zeigt jedoch einen Fall, in dem der Instruktionscache des LEON2 dafür sorgt, dass bei der Ausführung mehrerer Basisblöcke ein Cache-Hit eine längere Ausführungszeit hat als ein Cache-Miss. Somit treten in LEON2 trotz simpler Hardware spekulative Zeitanomalien auf.

2.8 Problem für Compositional Timing Analysis

Die eben beschriebenen Probleme lassen sich umgehen, indem man nicht nur den lokalen worst-case betrachtet und so auch die Kontrollflüsse mit global längeren Laufzeiten findet. Eine 'fully integrated timing analysis' , welche alle möglichen Zustände in der WCET Analyse betrachtet, umgeht das Problem der Zeitanomalien. Durch die hohe Zahl der zu betrachtenden Zustände ist diese Methode jedoch unpraktikabel für moderne Systeme. Das Ziel soll im folgenden sein, die Compositional Timing Anlyse trotz Zeitanomalien zu ermöglichen. Ein konkretes Beispiel für einen Prozessor der die 'fully compositional timing analysis' aufgrund seines Designs nicht ermöglicht ist der besprochenen LEON2 Prozessor.

Speicherzugriffe

Ein weiteres Problem neben den Zeitanomalien für die 'compositionality' sind verzögerte Speicherzugriffe

Diese treten auf wenn sich Speicherzugriffe gegenseitig auf dem Bus blockieren und dadurch andere Ausführungsüberschneidungen entstehen. In Abbildung 6 ist ein solcher Fall gezeigt, in dem ein Speicherzugriff aufgrund eines blockierten Bus mehr Laufzeit braucht. In diesem Fall entspricht die gesamte Verzögerung nicht der Verzögerung der ursprünglichen Bus-Blockierung. Die Berechnung der Länge dieses indirekten Effekts ist laut den Autoren ein bisher ungelöstes Problem. Die Autoren nennen so einen indirekten Effekt als weiteres Problem für die 'compositional' WCET Berechnung. Aufgrund der greedy scheduling Strategie der Speicherzugriffe könenn diese Fälle nicht nur in Out-of-Order, sondern auch in In-Order Prozessoren auftreten.

3 SIC

In 'Design and Analysis of SIC: A Provably Timing-Predictable Pipelined Processor Core'[3] präsentieren Hahn und Reineke ein Prozessordesign welches Zeitanomalien verhindert und die 'compositional timing analysis' und somit eine effiziente Hardwareanalyse der WCET-Berechnung ermöglicht. Zusammengefasst sind die Eigenschaften des Prozessors:

- Freiheit von Zeitanomalien und dadurch 'compositional timing analysis'
- Keine Branchprediciton
- In-Order Execution
- Speicherzugriffe sind strictly in-order

Um die indirekten Laufzeitverlängerungen bei Speicherzugrriffen, die durch den greedy-Speicherzugriff entstehen zu verhindern, führt der SIC Prozessor alle Bus-Zugriffe strictly in-order aus. Speicherzugriffe werden blockiert, solange vorherige Instruktionen noch auf den Speicher zugreifen könnten.

3.1 Monotonität

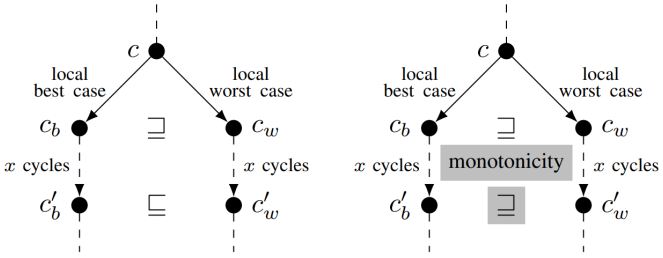

Abbildung 7: Prozessorfortschritt von zwei Kontrollflüssen. Links ohne, rechts mit monotonem Fortschritt [3]

Kernidee des SIC Prozessors ist die Eigenschaft der **Monotonität** im Bezug auf den Prozessfortschritt. Eine Zeitanomalie tritt auf, wenn ein lokaler worst-case den best-case im Bezug auf den Programmfortschritt im späteren Verlauf 'überholt'. Der lokale worst-case ist damit nicht mehr die sichere obere Grenze für die Programmlaufzeit. In Abbildung 7 links ist der Fortschritt der zwei Programmabläufe beschrieben. An einem Verzweigungspunkt kann man in der Hardware-Analyse vom best oder worst-case Szenario ausgehen. Das best-case Szenario wird dann schneller voranschreiten als der worst-case. Wie im Kapitel der Zeitanomalien beschrieben, kann der worst-case dann jedoch nach einer bestimmten Anzahl Takten weiter vorhergeschritten sein. Die Autoren nennen diesen fehlenden monoton steigenden Prozessfortschritt als Grund für Zeitanomalien. Ist es möglich, ein Prozessordesign zu entwerfen,

welches diese Eigenschaft der Monotonität hat, kann der local-worst case den best-case nicht überholen, da beide Instruktionen in jedem Takt voranschreiten, Abbildung 7 rechts. Es reicht dadurch aus in jedem Schritt der WCET-Berechnung den lokeln worst-case zu betrachten. Der Fortschritt einer Instruktion wird in der Arbeit als $\sqsubset_S$ definiert. Eine Instruktion kann auf zwei Arten fortschreiten:

- Sie wird in eine neue Pipelinestufe überführt:
  pre $\sqsubset_S$ IF $\sqsubset_S$ ID $\sqsubset_S$ EX $\sqsubset_S$ MEM $\begin{smallmatrix}\sqsubset_S WB \sqsubset_S \\ \sqsubset_S ST \sqsubset_S\end{smallmatrix}$ post
- Sie hat hat weniger Taktzyklen in der aktuellen Pipelinestufe übrig

Auf Basis dieses Prozessfortschritts und einer genauen Definition der Prozessorzustände auf die hier nicht näher eingegangen werden soll werden danach mehrere Theoreme und Lemma für den SIC definiert, welche anschließend verwendet werden um die Eigenschaften der Anomaliefreiheit und der 'fully compositional timing analysis' formal zu beweisen.

- **Lemma 1**: In SIC hängt der Fortschritt einer Instruktion nicht von den Nachfolgenden, sonder nur von vorherigen Instruktionen ab. Ein gewöhnlicher In-Order Prozessor kann diese Bedingung nicht erfüllen, wodurch es zu Abhängigkeiten von anderen Instruktionen kommen kann. Dies ist zum Beispiel der Fall, wenn eine nachfolgende Instruktion den Speicherzugriff der aktuellen Instruktion durch sein Instruction-Fetch verzögert.
  -> SIC erreicht diese Eigenschaft, indem alle Speicherzugriffe in Instruktionsreihenfolge stattfinden müssen.
- **Lemma 2** Wird das Verhalten des SIC auf einen Zustand angewand, entseht danach ein neuer Zustand der mehr Fortschritt hat. Entweder in eine neue Pipelinestufe oder weniger verbleibende Takte in der aktuellen Stufe.
- **Theorem:** Durch diese beiden Eigenschaften lässt sich nun beweisen, dass das Verhalten des SIC Prozessors monoton ist. Eine Instruktion schreitet in jedem Takt voran und wird durch keine andere Instruktion verzögert. Der genaue Beweis für diese Eigenschaft wurde in den Anhang der Arbeit verschoben

Mit diesem Lemma und Theoremen lässt sich nun durch einen einfachen Induktionsbeweis zeigen, dass eine Instruktion, die in der monotonen Ordnung vor einer anderen steht, seine Programmausführung auch früher beendet. Der Fall, in dem das worst-case Szenario den best-case 'überholt' ist dadurch nicht mehr möglich. Durch diese Anomaliefreiheit ist eine effiziente WCET-Analyse möglich, da es für die längste Laufzeit ausreicht, den worst-case Fall der Programmausführung zu betrachten.

3.2 Anomaliefreiheit und Compositionality

In einem zweiten Schritt wird bewiesen, dass durch die Eigenschaft der Monotonität sowohl Zeitanomalien verhindert als auch die 'compositional' WCET-Berechnung ermöglicht wird. Der Grundsatz dieser Beweise ist, dass Unsicherheiten, die zu Zeitanomalien oder indirekten Effekten führen können, durch das monotone Design und die strikt geordneten Speicherzugriffe verhindert werden.

Anomaliefreiheit bei Cache Zugriffen

Zeitanomalien können im SIC nicht durch Cache-misses hervorgerufen werden. Der lokale worst-case der vom Cache Miss ausgeht, braucht mindestens so viele Taktzyklen wie der Cache Hit. Diese Eigenschaft wird mittels der Eigenschaft der Monontonität beider Fälle und der Eigenschaft, dass eine Instruktion nicht von nachfolgenden Instruktionen abhängig ist, bewiesen.

Indirekte Effekte

Die compositional-Berechnung der WCET Berechnung geht kaputt, wenn ein Speicherzugriff auf den blockierten Bus zu indirekten Effekten und nicht abschätzbaren Verzögerungszeiten führt. Durch das SIC-Design ist diese Verzögerungszeit nun bestimmbar und Compositionality kann erreicht werden. Da Instruktionen im SIC stets monoton voranschreiten, kann die vorher unbestimmte Verzögerung nun auf höchstens p Taktzyklen begrenzt werden.

Compositionality bei Cachezugriffen

Auch bei Cachezugriffen kann eine Grenze für die zusätzliche Laufzeit bei cache-misses angegeben werden. Der Beweis dafür stützt sich analog wie die anderen Theoreme auf die Fortschrittseigenschaften des SIC.

3.3 PTARM

In der Arbeit von Hahn und Reineke [3] wird zusätzlich auf andere Ansätze zum Umgang mit Zeitanomalien in Hardware eingegangen. In einem Dagstuhl Workshop von 2003 wurden zwei Ansätze gegeben um die Vorhersage von Zeitschranken zu ermöglichen:

- "Reducing the sensitivity to interference from nonavailable information"
- "to match implementation concepts with analysis techniques to improve analyzability"

Kernthema des SIC war das ermöglichen der 'compositionality' um die effiziente WCET-Berechnung mittels 'compositional timing analysis' zu ermöglichen. Das SIC-Desing konzentriert sich also auf Ansatz 2. Neben mehreren anderen Arbeiten gehen die Autoren dann auf das Prozessordesign des Precision-Timed ARM (PTARM) ein. Der PTARM-Prozessor verschachtelt vier Hardwarethreads in einer Pipeline. Durch das Design hat eine Instruktion in PTARM eine fest Laufzeit und ist nicht vom Hardwarezustand abhängig. Der PTARM ist dadurch frei von Zeitanomalien und ermöglicht die 'compositional timing analysis'. Nachteil des Designs ist jedoch die längere Laufzeit von Instruktionen im Vergleich zu einem gewöhnlichen Prozessor mit Pipelining.

3.4 Performanceevaluierung

Abschließend soll auf die Performanceevaluierung des SIC eingegangen werden. Durch Messungen die die Autoren durchführen wird der Performanceverlust durch die Einschränkungen des SIC deutlich.

Performancevergleich des SIC mit In-Order:

Der SIC-Prozessor verhindert Zeitanomalien, indem er besprochene Prozessoroptimierungen, welche Zeitanomalien hervorrufen verhindert. Zusätzlich werden Speicherzugriffe strikt in Programmreihenfolge ausgeführt. In einer ersten Messung wird gezeigt, wie sich diese Einschränkungen auf die Laufzeit des Prozessors im Vergleich zu einem normalen In-Order Prozessors auswirken. Die WCET Messung wird zum einem messbasiert, mittels eines FPGAs, und statisch mit eine WCET-Analysetool durchgeführt. In beiden Fällen wurden Messungen für unterschiedliche Speicherlatenzen und Cachegrößen druchgeführt. Die Ergebnisse sind in Abbildung

8 dargestellt. Der Vergleich des SIC mit einem normalen In-Order Prozessors zeigt, dass die gemessene Laufzeit im FPGA und die statische Laufzeitanalyse des SIC 6 bis 7 Prozent schlechter sind. Wird die Geschwindigkeit des Speichermediums erhöht sinkt der Performancenachteil des SIC, da die Einschränkung des strictly in order Speicherzugriffs weniger stark ins Gewicht fällt.

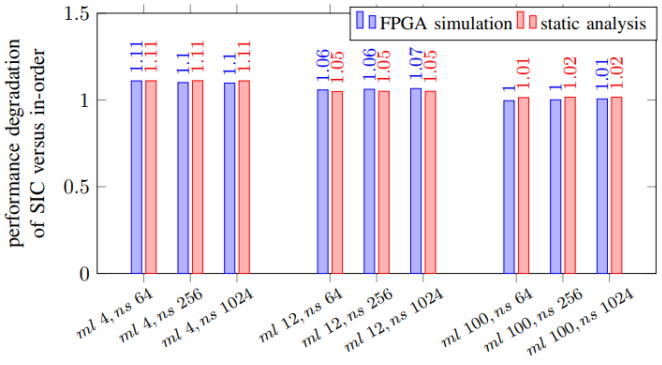

Abbildung 8: Performanceverlust des SIC verglichen mit einem normalen In-Order Prozessor. ml = Speicherlatenz nz = Anzahl an Cache-sets [3]

Vergleich mit PTARM:
Eine zweite Messung vergleicht den SIC mit dem vorher beschriebenen PTARM-Prozessor, welcher auch Zeitanomalien verhindert und die 'fully composition timing analysis' ermöglicht. Bei dieser Messung beachten die Autoren der Arbeit die speziellen Eigenschaften des PTARM. Auf den schnelleren Taktzyklus und die Besonderheiten beim Speicherzugriff gehen sie im Text und bei den Messungen ein. Gemessen wurde der Performanceverlust des PTARM gegenüber dem SIC für verschiedene Cachekonfigurationen. Neben den unterschiedlichen Speicherlatenzen und Cachegrößen wurden noch Messungen mit einem PTARM mit nur einem vierteil seiner Cachegröße eingefügt, da in PTARM normalerweise vier Threads gleichzeitig laufen. Außerdem werden Messungen von nicht-optimiertem Programmcode, welcher mehr Speicheroperationen enthält, durchgeführt und Messungen dargestellt, die die höheren Kosten des PTARM einbeziehen. Die Messungen ergeben eine rund doppelt so gute Performance des SIC gegenüber PTARM. Mit steigender Cachegröße sinkt der Vorteil des SIC, da dessen effektiv größerer Cache dadurch weniger stark gewichtet ist.
Laufzeit der statischen Analyse:
Ein Vergleich der Laufzeiten der statischen WCET-Berechnung zeigt den Vorteil vom SIC. Verglichen wurden die Laufzeiten von SIC, einem In-Order Prozessor und PTARM. Auch hier für verschiedene Cachegrößen und Speicherlatenzen. Ziel des SIC ist es, anomaliefrei zu sein und die kompositionelle Zeitanalyse zu ermöglichen und somit die Laufzeit der statischen WCET-Analyse zu verkürzen. Deshalb wird die WCET-Laufzeit des SIC mit der Laufzeit auf einem normalen In-Order Prozessor mit Zeitanomalien verglichen. Neben dem normalen In-Order Prozessor werden auch die Laufzeiten für einen mehrkernigen Prozessor gemessen. In dem mehrkernigen Prozessor treten die sogenannten indirekten Effekte auf, da es zu blockierenden Buszugriffen kommt. Als Obergrenze für diese indirekten Effekte wurde eine sogenannte compositional base bound angenommen. Die Messergebnisse in Abbildung 9 bestätigen das Ziel der Entwicklung des SIC. Zu sehen ist, dass die WCET-Laufzeit des SIC deutlich schneller ist als die der einkernigen In-Order Variante in einem realistischen Szenario mit einer Speicherlatenz von 5ms um Faktor 5. Der Laufzeitvorteil zu der mehrkernigen Variante fällt im gleichen Szenario noch deutlicher aus. Diese benötigt rund 30-mal länger als der SIC. Die Berechnungszeiten für SIC und PTARM sind annähernd gleich.

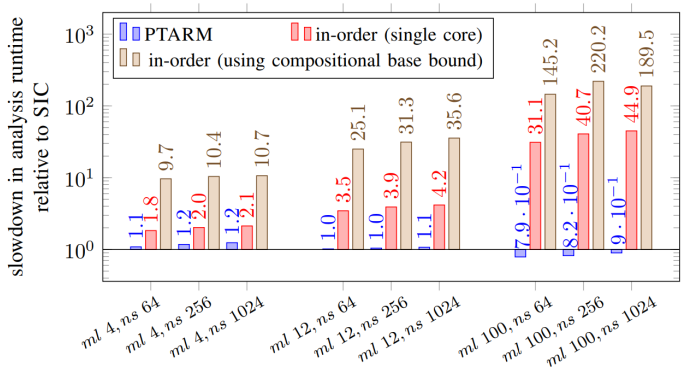

Abbildung 9: Laufzeit der statischen WCET-Berechnung [3]

4 SCHLUSS

Die WCET-Analyse ist ein wichtiger Teil der Forschung an Echtzeitsystemen. Zeitanomalien sind dabei ein bekanntes Problem, das schon in frühen Arbeiten zur Berechnung von oberen Laufzeitschranken beschrieben wird. Reineke et al. sind die Ersten, die dieses Problem formell beschreiben. Zusätzlich teilen sie diese in mehrere Kategorien ein. Des Weiteren sind Reineke und Hahn die Ersten, die einen formellen Beweis für einen anomaliefreihen Prozessor mit Pipelining geben, um eine effiziente WCET-Berechnung mit diesem zu ermöglichen.

Hierbei konzentrieren sich die Autoren auf die Eigenschaft der Monotonität und zeigen, dass diese Anomaliefreiheit garantiert, abschließend wurde die Performance des Prozessors und die Effizienz der WCET-Berechnung mit anderen Prozessoren verglichen und gezeigt, dass der SIC Prozessor ohne bedeutende Performanceeinbußen die WCET-Berechnung deutlich verbessert. Ein realistischer Einsatz des SIC Designs kann umgesetzt werden, indem aktuelle Prozessoren einen *montonic* Modus einbinden, welcher die Einschränkungen des SIC umsetzt. Wenn nötig können Zeitanomalien in diesem Modus dann verhindert werden. Zukünftige Arbeit der Autoren soll sich darauf konzentrieren, den SIC Prozessor zu erweitern, um komplexere Prozessoroptimierung, welche hier besprochen wurden, zu ermöglichen, ohne die Eigenschaft der Monotonität aufzugeben.

LITERATUR

[1] Gernot Gebhard. 2010. Timing anomalies reloaded. In *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[2] Sebastian Hahn, Michael Jacobs, and Jan Reineke. 2016. Enabling Compositionality for Multicore Timing Analysis. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems* (Brest, France) *(RTNS '16)*. Association for Computing Machinery, New York, NY, USA, 299–308. https://doi.org/10.1145/2997465.2997471

[3] Sebastian Hahn and Jan Reineke. 2018. Design and Analysis of SIC: A Provably Timing-Predictable Pipelined Processor Core. In *2018 IEEE Real-Time Systems Symposium (RTSS)*. 469–481. https://doi.org/10.1109/RTSS.2018.00060

[4] Sebastian Hahn, Jan Reineke, and Reinhard Wilhelm. 2015. Towards compositionality in execution time analysis: definition and challenges. *ACM SIGBED Review* 12, 1 (2015), 28–36.

[5] Thomas Lundqvist and Per Stenstrom. 1999. Timing anomalies in dynamically scheduled microprocessors. In *Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No. 99CB37054)*. IEEE, 12–21.

[6] Jan Reineke, Björn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. 2006. A Definition and Classification of Timing Anomalies, Vol. 4.

[7] Sebastian Rietsch. 2016. WCET-Analyse in Mehrkern-Systemen. https://doi.org/Lehre/WS15/PS_KVBK/papers/Paper-Rietsch.pdf

[8] Dr. Peter Ulbrich. [n. d.]. Vorlesung Echtzeitsysteme. https://www4.cs.fau.de/Lehre/WS19/V_EZS/.

[9] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. 2008. The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools. *ACM Trans. Embed. Comput. Syst.* 7, 3, Article 36 (may 2008), 53 pages. https://doi.org/10.1145/1347375.1347389

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009