Anomalien auf Mehrkern-Systemen - Warum läuft meine Anwendung nicht schneller?

Eine Ausarbeitung im Rahmen des KvBK-Seminars

Maximilian Wagner
Friedrich-Alexander-Universität Erlangen-Nürnberg

Zusammenfassung

Diese Ausarbeitung behandelt Ausführungszeit-Anomalien auf Mehrkern-Systemen, wie diese entstehen, und warum sie für Echtzeit-Systeme problematisch sind. Im besonderen wird auf die Systemelemente Pipelines, Caches, und Busse im Kontext von Anomalien eingegangen. Sie steigern zwar die Leistung, aber auch die Wahrscheinlichkeit für unerwünschtes Auftreten von Anomalien und Interferenzen. Dennoch überwiegt der Leistungsgewinn oft das Risiko. Daher wird betrachtet, was bei der Verwendung dieser Elemente in Echtzeitsystemen zu beachten ist. Außerdem werden Zeitpunkt-Anomalien und Domino-Effekte, sowie inhärente und virtuelle Interferenzen betrachtet. Diese versucht man möglichst zu vermeiden, da sie eine korrekte Ermittlung von bester/schlechtester anzunehmender Ausführungszeit (BCET/ WCET) anhand einer Abstraktion erschweren. Wie sie entstehen und wie man sie vermeiden kann ist Inhalt dieser Ausarbeitung.

1. EINLEITUNG

Mehrkern-Prozessoren sind im privaten Anwendungsbereich heutzutage nahezu allgegenwärtig, und auch in der Forschung sind sie in vielen Feldern vertreten. Der Hauptgrund hierfür liegt auf der Hand: Parallelisierung innerhalb von Anwendungen sowie parallele Ausführung mehrerer unabhängiger Programme gleichzeitig. Und, daraus resultierend, eine schnellere Ausführungszeit einzelner Aufgaben im Vergleich zu Einkern-Systemen. Gerade im Hinblick auf Echtzeitsysteme ist dies wünschenswert, da Geschwindigkeit zwar keinesfalls Rechtzeitigkeit garantiert, aber zumindest unterstützt. Zudem bestehen aktuelle Echtzeitsysteme in der Regel aus vielen einzelnen Aufgaben, die idealerweise unabhängig voneinander gleichzeitig bearbeitet werden können sollten. Jedoch kann man im Umgang mit Mehrkern-Systemen, und, im Speziellen, mit Mehrkern-Echtzeitsystemen, beobachten, dass die Ausführungsgeschwindigkeit bereits angepasster Anwendungen im Vergleich zu klassischen Einkern-Systemen nicht unbedingt konsistent reduziert wird. Im Gegenteil, teilweise liegt die Ausführungszeit auf einem Mehrkern-Systemen sogar über der auf einem Einkern-System. Dies ist gerade im Hinblick auf Vorhersagbarkeit, die für Zeitanalysen von Echtzeitsystemen essentiell ist, problematisch. Somit ist es durchaus möglich, dass das erwartete vom realen Zeitverhalten abweicht: eine Anomalie. Diese Anomalien, ihre Ursachen und Auswirkungen sind das Thema dieser Ausarbeitung.

Im Folgenden wird zuerst eine kurze Einführung zu System-Abstraktionen gegeben. Danach wird erläutert, wodurch besagte Anomalien grundsätzlich entstehen können. Darauf folgt eine Beschreibung konkreter Anomalien. Abschließend werden explizit einzelne Betriebsmittel hinsichtlich ihrer Eignung und Problematik bei der Verwendung in Mehrkern-Echtzeitsystemen diskutiert.

2. SYSTEM-ABSTRAKTIONEN

Abstraktionen sind vereinfachte Modelle von Systemen, die zu Analysezwecken erstellt werden (z.B. für Zeitanalyse). Eine Möglichkeit ist, sich diese Abstraktionen wie einen Zustandsautomaten vorzustellen. Somit bestimmt man Attribute wie mögliche Ausgangszustände, Zustandsübergänge und Endzustände. Idealerweise ist die Anzahl an Möglichkeiten für diese Attribute, und damit die Komplexität der Abstraktion, möglichst gering zu halten. Dies reduziert den Aufwand von Analysen wie z.B. der Zeitanalyse. Das Stichwort ist hierbei die sog. *Vorhersagbarkeit*. Diese beschreibt die Komplexität des Ermittelns von Aussagen über konkrete Zustände der Abstraktion. Hierfür sind vor allem folgende Eigenschaften hilfreich: Determinismus des Modells und eine möglichst geringe Menge an möglichen Anfangszuständen. So verfügen simple Systeme mit Determinismus und irrelevantem/garantiertem Initialzustand über gute Vorhersagbarkeit.

Hauptanwendungszweck dieser Abstraktionen im Kontext von Echtzeitsystemen ist die Zeitanalyse, mit der idealerweise feste Grenzen für die beste und schlechteste Ausführungszeit (*Best-/Worst-Case-Execution-Time*, kurz *BCET/ WCET*) ermittelt werden. Leider sind diese Abstraktionen in der Realität nicht immer vollständig. Probleme wie Informationsverlust aufgrund von extrem großen Zustandsmengen liegen vor, d.h. es existieren mehr Möglichkeiten als berücksichtigt werden können. Ursache hierfür ist beispielsweise Nicht-Determinismus. Diese Probleme beeinträchtigen die Analyse und sorgen für ungenauere/unnötig große Grenzen des ermittelten Ausführungszeit-Intervalls.

Gerade Mehrkern-Systeme gestalten sich in der Praxis als schwierig zu abstrahieren, da sie in der Regel komplexe (z.B. nicht-deterministische) Ausführungsreihenfolgen aufweisen.

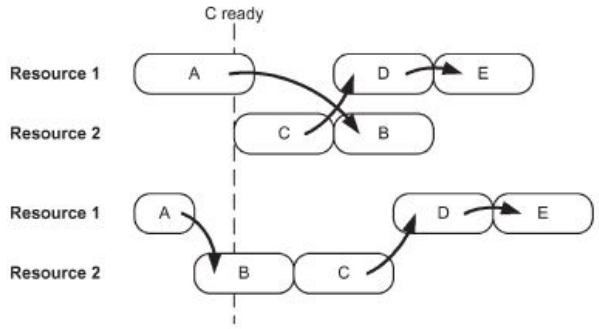

Figure 1: zeitliche Anomalie bei Ablaufplanung [9]

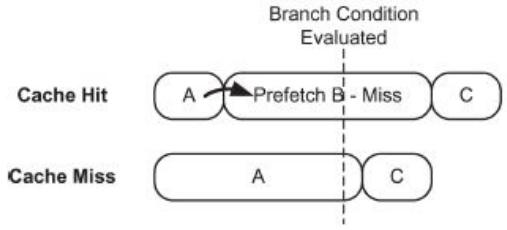

Figure 2: zeitliche Anomalie durch falsche Verzweigungsvorhersage [9]

3. PROBLEM-URSACHEN

Hauptursache der Probleme, die stark schwankende Ausführungszeiten zur Folge haben, ist die gemeinsame Verwendung von Betriebsmitteln (BM) wie Caches oder Busse. Das geteilte Verwenden der Betriebsmittel ist in Mehrkern-Systemen oftmals üblich und dient der Kosten- und Leistungsoptimierung. Leider entstehen hierdurch *Interferenzen*, d.h. Interaktionen und Abhängigkeiten zwischen normalerweise isolierten Anwendungen. Diese erschweren die Analyse des Systems, da sie Nicht-Determinismus mit sich bringen können. Man unterscheidet konkret zwischen *inhärenten* und *virtuellen* Interferenzen, wobei inhärente realen und virtuelle künstlichen Nicht-Determinismus zur Folge haben. [9] Wie bereits im Abschnitt zu Abstraktionen erwähnt, ist gerade Nicht-Determinismus hinderlich für das Erstellen korrekter Zeitanalysen. Egal welche von beiden Interferenzarten auftritt, sie behindern also maßgeblich die *Vorhersagbarkeit* des Systems.

3.1 Inhärente Interferenzen

Inhärente Interferenzen bezeichnen Zugriffe auf BM durch eine andere Anwendung, über die vom Standpunkt der eigenen Anwendung wenig bis nichts bekannt ist. Somit können sie zwar beobachtet werden, treten aber zu unvorhersagbaren Zeitpunkten auf. [9] Ein Beispiel für ein BM, bei dem häufig inhärente Interferenzen auftreten, ist Speicher (egal ob Hauptspeicher oder Cache), der von mehreren Prozessen oder Prozessorkernen gleichzeitig genutzt wird. Einzelne Prozesse verfügen über keinerlei Information, wann andere auf diese BM zugreifen. Somit gestaltet es sich schwierig, diese Zugriffe in der Zeitanalyse zu allen möglichen Zeitpunkten mit zu berücksichtigen, da die Anzahl an Möglichkeiten oftmals zu groß ist. Verzichtet man jedoch darauf diese zu berücksichtigen, ist eine korrekte Zeitanalyse nicht mehr garantiert.

3.2 Virtuelle Interferenzen

Virtuelle Interferenzen entstehen erst durch die Abstraktion der vorliegenden Architektur, die im Kontext einer Zeitanalyse notwendig ist. [9] Durch die Abstraktion kann künstlicher Nicht-Determinismus eingeführt werden. Genau dieser ist die Ursache für virtuelle Interferenzen. Als Beispiel: Man betrachtet einen *"out-of-order"*-Prozessor [6], der einen konkreten Eingabestrom stets in der gleichen deterministische Reihenfolge ausführt. Diese hängt jedoch sowohl vom Ausgangszustand, als auch von eventuell auftretenden Eingaben ab. Somit wäre es theoretisch zwar möglich, die Reihenfolge konkret zu bestimmen, praktisch ist dies jedoch durch die Vielzahl an möglichen Kombinationen nicht umsetzbar. Somit wird die Ausführungsreihenfolge als unbekannt betrachtet, was zur Folge hat, dass auch Aufrufzeitpunkte auf geteilte BM variieren können. Genau diese Varianz ist die Ursache für virtuelle Interferenzen.

4. ANOMALIEN

4.1 Zeitliche Anomalien

Als zeitliche Anomalie bezeichnet man den Spezialfall, bei dem sich das Auftreten von lokalen zeitlichen Unterschieden, wie Verzögerungen, unerwartet auf das globale Zeitverhalten auswirkt. [9] Somit behindern sie die Vorhersagbarkeit eines Systems massiv.

Ein Beispiel: Wir haben fünf Anweisungen A-E, die von zwei Prozessoren bearbeitet werden. Prozessor 1(P1) ist für A, D und E zuständig, Prozessor 2(P2) für B und C. Zudem gelten die folgenden Sequenzen: (A->B) und (C->D->E). Jedoch ist C zu Beginn noch nicht bereit, ausgeführt zu werden. Im lokalen Idealfall ist A daher bearbeitet, bevor C bereit ist. Somit befindet sich P1 im Ruhezustand, bis P2 C bearbeitet hat. Da C noch nicht bereit ist, bearbeitet P2 jedoch zunächst B. Dadurch befindet sich P1 unnötig lange im Ruhezustand => die globale Ausführungszeit steigt.
Gegenteilig dazu der Fall, dass A durch beispielsweise einen Cache-Miss(siehe Kapitel 5.2) länger als im Idealfall benötigt. Nun ist C bereit, bevor A bearbeitet wurde, und kann somit vor B behandelt werden. Somit überlappen sich die aktiven Zeiten der Prozessoren besser => die globale Ausführungszeit sinkt. Dies ist in Abbildung 1 veranschaulicht. Obwohl A in letzterem Fall also länger braucht, sinkt die gesamte Ausführungszeit: Eine *zeitliche Anomalie*.

Ein weiteres Beispiel: Dieses Mal werden Verzweigungsvorhersagen betrachtet. Es existieren drei Anweisungen A, B und C. A könnte zu B führen, dies hängt jedoch von einer Verzweigungsauswertung ab. Man nimmt an, dass A einen Cache-Hit hat. Nun ist A abgearbeitet, bevor die Verzweigungsauswertung abgeschlossen ist. Somit wird nun B präventiv in den Cache geladen. Falls die Verzweigungsauswertung jedoch ergeben würde, dass B nicht erreicht wird, wurde der Cache unnötig beladen. Dies könnte wiederum zu zukünftigen Cache-Misses führen, die die gesamte Ausführungszeit unnötig steigern. Nun nimmt man an, dass A aber durch einen Cache-Miss lange genug verzögert wird, bis die Verzweigungsauswertung abgeschlossen ist. Jetzt ist be-

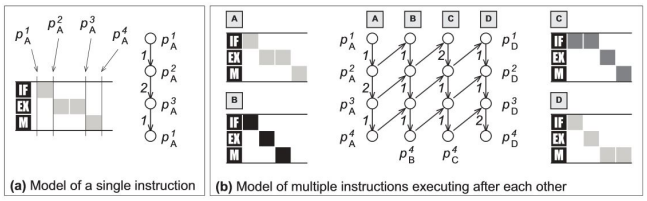

Figure 3: Pipeline-Funktionsweise [3]

kannt, dass B nicht ausgeführt werden muss, und somit auch nicht unnötig in den Cache geladen wird. Dadurch kann die Ausführungszeit niedriger als im ersten Fall sein. Abbildung 2 zeigt dies nochmal graphisch.

4.2 Domino-Effekte

Unter einem *Domino-Effekt* versteht man, wenn zwei unterschiedliche Initialzustände s,t eines Systems sich dermaßen stark auf die benötige Ausführungszeit einer oder mehrerer Anweisungen auswirken, dass die Differenz zwischen den einzelnen Ausführungszeiten nicht durch eine Konstante erfassbar ist. Sie sind Spezialfälle von zeitliche Anomalien. Als anschauliches Beispiel eignet sich z.B. ein in Schleife endlos ausgeführtes Programm. Bei Fall s bleibt die Ausführungszeit über jede Iteration konstant, bei Fall t hingegen steigt die Ausführungszeit mit jeder Iteration. Somit wird die zeitliche Differenz zwischen wiederholten Ausführungen des selben Programms bei jeder Iteration größer. Würden diese Domino-Effekte nicht existieren, könnte man konkrete Initialzustände bei der Analyse vernachlässigen und stattdessen eine vorher ermittelte zeitliche Differenzkonstante verwenden. Leider treten sie in der Realität jedoch auf, wie bspw. in der Pipeline eines PowerPC 755 [8]. Mehr zu Domino-Effekten findet sich in [1].

5. PROBLEMATISCHE BETRIEBSMITTEL

In realen Mehrkern-Systemen treten sämtliche im folgenden behandelten BM natürlich oft in Kombination auf, im Rahmen dieser Ausarbeitung werden sie jedoch, soweit möglich, isoliert betrachtet. Für konkrete Abstraktionen und darauf basierende Analysen müssen sie aber als Ganzes, d.h. inklusive Interaktionen zwischen den einzelnen BM, betrachtet werden.

5.1 Pipelines

Pipelines reduzieren die Ausführungszeit eines Anweisungsblocks, in dem sie verschiedene Anweisungen teil-parallel statt streng sequentiell ausführen(s.Figur 3). Dies beeinflusst jedoch andere geteilte BM, z.B. Caches. Und diese BM im Rückschluss wieder die einzelnen Anweisungen. *Inhärente Interferenzen* sind also zu berücksichtigen. Im Vergleich zu streng sequentieller Ausführung steigt jedoch auch die Anzahl an möglichen Zuständen, die in unserer Abstraktion zu berücksichtigen sind. Für jeden Zwischenschritt sind nun eine Vielzahl neuer, bislang irrelevanter Variablen zu beachten. Diese enthalten Information über beteiligte BM, die ohne die Verwendung von Pipeline nicht zu berücksichtigen wären. Somit steigt die Komplexität einer Zeitanalyse. Durch die Abstraktion kann jedoch auch Nicht-Determinismus in unser ansonsten deterministisches System eingeführt werden, falls konkrete Informationen über andere beteiligte BM nicht vorliegen. *Virtuelle Interferenzen* sind also ebenfalls möglich (vgl. Kapitel 3.2). Zudem gilt: je komplexer (bzgl. implementierten Funktionen) die Pipeline, desto höher die Anzahl an möglichen Zuständen, desto schwieriger (und damit potentiell ungenauer) die Zeitanalyse. Und somit steigt auch die Wahrscheinlichkeit einer Anomalie. Pipelines ermöglichen also bessere Leistung, bergen jedoch die Gefahr Anomalien in ein sonst stabiles System einzuführen und somit Echtzeit-Anforderungen zu verletzen.

5.2 Caches

Das Gleiche gilt für Caches: Sie reduzieren die Anzahl an Zugriffen auf den Hauptspeicher und ermöglichen somit eine schnellere Ausführung. Hierfür werden einzelne Inhalte des Hauptspeichers im schneller abrufbaren Cache zwischengespeichert. Soll auf diese Inhalte erneut zugegriffen werden, befinden sie sich möglicherweise noch im Cache. Ist dies der Fall, spricht man von einem *Cache-Hit*, ansonsten von einem *Cache-Miss*.
Betrachtet man Caches hinsichtlich ihrer Vorhersagbarkeit, ist vor allem die implementierte Ersetzungsstrategie ausschlaggebend. Von allen Strategien ist aus Vorhersagbarkeitsgründen nur eine wirklich für die Verwendung in einem Echtzeit-System geeignet: Least-Recently-Used (LRU). Weitere Strategien, beispielsweise First-In-First-Out (FIFO) sind aufgrund schlechterer Vorhersagbarkeit reduziert geeignet und werden daher hier nicht weiter behandelt. [9] LRU weist die beste Vorhersagbarkeit auf [9], da am sichersten zu ermitteln ist ob ein Cache-Hit oder Cache-Miss auftreten wird. Ist eine Unterscheidung zwischen Hit/Miss nicht im voraus möglich, müssen aufgrund von zeitlichen Anomalien beide Fälle berücksichtigt werden. Da jedoch ein einzelner Cache-Miss eine Verzögerung von bis zu 700 Rechenzyklen verursachen kann [9], wirkt sich jeder einzelne dieser nicht bestimmbaren Speicherzugriffe stark auf die Größe des Intervalls zwischen bester und schlechtester ermittelter Ausführungszeit aus.
Auch der Initialzustand des zu analysierenden Caches spielt eine elementare Rolle für die Analyse (s. Domino-Effekt), ist in der Praxis aber schwierig zu bestimmen. Grund hierfür ist die Vielzahl an möglichen vorangehenden Zuständen, beispielsweise verursacht durch früher ausgeführte und bereits abgeschlossene Anwendungen. Somit müssen für eine korrekte Analyse *alle* möglichen Initial-/Vorgängerzustände berücksichtigt werden. [9]. Eine Untersuchung, inwiefern die verschiedenen Ersetzungsstrategien hinsichtlich ihrer Hits/ Misses von dem Initialzustand des verwendeten Caches abhängen, hat jedoch ergeben, dass LRU sich verglichen zu FIFO und anderen Strategien sehr unabhängig/robust zeigt. [9] Für eine detaillierte Analyse der einzelnen Ersetzungsstrategien hinsichtlich *WCET*-Ermittlung etc. siehe [9].
Dennoch gilt für geteilte Caches die gleiche Regel wie für Pipelines: Zur Verwendung in Echtzeitsystem sind sie bedingt geeignet, da sie Anomalien in ein sonst stabiles System einführen können. Eine mögliche Alternative wäre beispielsweise die Verwendung von sogenanntem *scratchpad-memory* [5].

5.3 Busse

Die Aufgabe eines Busses ist, entfernte Objekte (bspw. Rechner, oder Elemente innerhalb eines Systems) zu verbinden und Datentransfer zwischen diesen zu ermöglichen. Anders als bei Punkt-zu-Punkt-Verbindungen werden dabei mehr als zwei Objekte gleichzeitig verbunden. Der Datenaustausch

wird über spezielle Protokolle reguliert. Wie man sieht, sind Busse somit stets geteilt verwendete BM.
Üblicherweise sind Busse langsamer getaktet als Prozessoren. Somit kann es passieren, dass Prozessoren auf den Beginn des nächsten Bus-Taktzyklus warten müssen um mit den Bussen (und damit mit der über den Bus verbunden Peripherie) zu interagieren. Für eine korrekte Analyse ist es daher wichtig, dass bekannt ist wie lange man zu jedem möglichen Zeitpunkt warten muss bis der nächste Bus-Taktzyklus beginnt. Je schneller der Bus-Takt, desto weniger Möglichkeiten sind zu berücksichtigen. Es existieren bestimmte Schwellenwerte für den Bustakt, die sich anhand folgender Formel [9] bestimmen lassen:

$$states := \frac{f_{CPU}}{\gcd(f_{CPU}, f_{BUS})}$$

Ideal wäre also prozessor-identisch getaktete Busse zu verwenden. Allerdings ergibt bereits halber Prozessortakt lediglich zwei mögliche Zustände und bietet sich damit als guter Kompromiss an.
Zudem unterscheidet man zwischen parallelen und bit-seriellen Bussen. Erstere verfügen über (wie der Name schon vermuten lässt) mehrere Leitungen, die den parallelen Transport verschiedener Datenstränge zum selben Zeitpunkt ermöglichen. Üblicherweise trennt man hierbei Adressen von Daten. Ähnlich wie bei Prozessoren ermöglicht dies das sog. *Bus-Pipelining* [2], da sich Adress- und Datenübertragung verschiedener Aufrufe überlappen können [9]. Allerdings führen diese Bus-Pipelines die selbe Problematik wie Prozessor-Pipelines mit sich: eine übergreifende Logik muss die eingehenden Anfragen organisieren und die Anzahl an zu analysierenden Zuständen in einer Abstraktion steigt. Objekte, die solche Anfragen stellen dürfen, werden als *Bus-Meister* bezeichnet [9]. In Mehrkern-Systemen existiert in der Regel mehr als einer dieser Bus-Meister, da meist mehrere/alle Rechenkerne über die nötigen Rechte verfügen. Zur Verwaltung mehrerer Meister existieren daher viele verschiedene Ansätze. Der für Echtzeit-Systeme interessanteste ist der sog. *zentrale Bus-Arbiter* [4]. Dieser hat den großen Vorteil, dass er, einer konkreten Logik folgend, *deterministisch* Zugriff gewährt. Ein Beispiel für die angewandte Logik ist das sog. *Zeitmultiplex-Verfahren* (eng: Time-Division Multiple Access, *TDMA*). Hierbei werden jedem Meister feste Zeitfenster zugeteilt. Diese Zuteilung erfordert zwar präzise Vorarbeit, um möglichst hohe Leistung zu erhalten, ist aber dafür in der Abstraktion leicht zu analysieren.
Zusammenfassend sollten Busse für die Verwendung in Mehrkern-Echtzeitsystemen also folgende Eigenschaften haben: Ein Takt zumindest gleich der Hälfte des Prozessor-Takts; und Bit-seriell oder, falls Leistung wichtig ist, parallel mit deterministischer Zugriffslogik.

6. SCHLUSS

Es gibt bei der Auswahl der Architektur für ein Mehrkern-Echtzeitsystem also viel zu beachten, um das Auftreten von Anomalien möglichst gering zu halten und somit die Zeitanalyse einfacher und präziser zu gestalten. Dies ermöglicht Echtzeit-Systemen auch auf Mehrkern-Basis notwendige Garantien wie Rechtzeitigkeit gewährleisten zu können. Als Grundsatz gilt: "The principle to be applied in the design of multicore architecture with predictable timing behavior is the *systematic elimination of interference on shared resources wherever they are not absolutely needed for performance*" [9]. Ein weiterer Ansatz zur Reduzierung der Problematik: Konflikte auf gemeinsam genutzten Betriebsmitteln können über fest vergebene Prioritäten verhindert werden. Dies ermöglicht eine einfachere Analyse, da der mögliche Zustandsraum, der zu beachten ist, konsequent schrumpft. Die Autoren von [9] schlagen zu dem folgende Mehrkern-Architektur vor: Jeder Kern soll über private L1- und L2-Caches verfügen, bei denen LRU als Ersetzungsstrategie angewendet wird. Weiterhin wird dazu angehalten, für expliziten Anweisungscode isolierte Speicher-Hierarchien zu verwenden, da in der Praxis Code selten zwischen Kernen geteilt wird (abseits des Betriebssystems) und somit unnötige Interferenzen vermieden werden. Allerdings trifft dies nicht auf eingebettete Systeme zu, welche heutzutage häufig die Ziel-Umgebung für Echtzeit-Systeme sind. Hier wird vorgeschlagen, jedem einzelnen Nutzer feste Zugriffszeiten auf gemeinsam genutzte Betriebsmittel zuzuweisen. Dies erfordert jedoch Vorsicht bei der Ablaufplanung, um Interferenz-Freiheit zu garantieren. Weitere Informationen zu der beschriebenen Architektur finden sich in [7].

7. LITERATUR

[1] C. Berg. Plru cache domino effects. *WCET*, 6:6th, 2006.

[2] J. D. Carr. Data transfer method/engine for pipelining shared memory bus accesses, Dec. 26 2000. US Patent 6,167,475.

[3] J. Engblom. Processor pipelines and static worst-case execution time analysis. 2002.

[4] K. A. Felix. Bus arbiter, Dec. 2 1980. US Patent 4,237,534.

[5] P. Marwedel, L. Wehmeyer, M. Verma, S. Steinke, and U. Helmig. Fast, predictable and low energy memory references through architecture-aware compilation. In *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, pages 4–11. IEEE Press, 2004.

[6] V. Popescu, M. A. Schultz, G. A. Gibson, J. E. Spracklen, and B. D. Lightner. Processor architecture providing out-of-order execution, May 6 1997. US Patent 5,627,983.

[7] J. Rosen, A. Andrei, P. Eles, and Z. Peng. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 49–60. IEEE, 2007.

[8] J. Schneider. *Combined schedulability and WCET analysis for real-time operating systems*. Shaker, 2003.

[9] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand. Memoriy hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *IEEE transactions on computer-aided design of integrated circuits and systems*, 28(7):966–978, July 2009.