Graduiertenkolleg 1103

Embedded Microsystems

Albert-Ludwigs-Universität Freiburg

Energieeffizientes Scheduling für Echtzeitsysteme

Statusbericht

Thorsten Zitterell

Betreuer: Prof. Dr. Christoph Scholl
Lehrstuhl: Betriebssysteme

Freiburg, im September 2008

Institut für Informatik

Institut für Mikrosystemtechnik

1 Aktueller Stand der Promotion

Meine Promotion befindet sich im dritten Jahr und befasst sich mit energieeffizienten Schedulingverfahren für Echtzeitsysteme. Im Moment beschäftige ich mich mit Erweiterungen und Optimierungen des Verfahrens unter Berücksichtigung von Besonderheiten realer Architekturen.

2 Zusammenfassung der Dissertation

Ein Echtzeitsystem ist ein System bei dem die Korrektheit des Systemverhaltens nicht nur davon abhängt, dass Berechnungen ein Ergebnis liefern, sondern auch das Ergebnis bis zu einem gegebenen Zeitpunkt (*Deadline*) vorliegt. Da Echtzeitsysteme meist in eingebetteten Mikrosystemen eingesetzt werden, unterliegen sie besonderen Einschränkungen und Anforderungen: Vor allem im Bereich mobiler Geräte sind dies begrenzte Energieressourcen und, um Produktionskosten gering zu halten, auch Speicherplatzbeschränkungen. Eine wesentliche Komponente eines Betriebssystems ist der Scheduler, der die Rechenzeit des Prozessors unter Einhaltung von Echtzeitkriterien an die Tasks verteilt. Um zeitkritische Abläufe für ein reales System überhaupt garantieren zu können, müssen der Scheduler entsprechend modelliert und die Tasks ausreichend spezifiziert sein, so dass bereits im Vorfeld entscheidbar ist, ob eine Taskmenge schedulbar ist oder nicht.

Mein besonderes Interesse zielt in diesem Zusammenhang auf energieeffiziente Schedulingverfahren. Hier werden Techniken der Spannungs- und Frequenzänderung zur Laufzeit, so genanntes *Dynamic Frequency* und *Voltage Scaling* (DFS/DVS), untersucht. Wird beispielsweise ein Prozessor mit geringerer Frequenz getaktet, so lässt sich auch die Betriebsspannung verringern und Energie einsparen. Dennoch lässt sich diese Methode bei Echtzeitsystemen nur ausnutzen, wenn das spezifizierte Zeitverhalten weiterhin erfüllt ist. Die Forschungsarbeit betrachtet hierbei energieeffizientes Echtzeitscheduling für sowohl Prozessoren mit diskreten Frequenzen als auch Taskmengen mit dynamischem Zeitverhalten. Damit wird zum Einen die oft verwendete Idealisierung kontinuierlicher Frequenzspektren fallengelassen und zum Anderen berücksichtigt, dass die Laufzeit bei Software stark variieren kann.

Typische Einsatzgebiete einer solchen Technik, die Echtzeitkritierien berücksichtigt und gleichzeitig elektrische Energie einspart, sind beispielsweise verzögerungsfreie Video-/Audiodekodierung bei Mobiltelefonen oder Datenverarbeitung und Kommunikation in Sensornetzwerken.

2.1 Energieeffizientes Echtzeitscheduling

2.1.1 Modellierung

Formal lässt sich das betrachtete Scheduling-Problem folgendermaßen beschreiben. Der Prozessor auf dem die Tasks ausgeführt werden sollen, stellt m verschiedene Operationsmodi $S = \{s_1, ..., s_m\}$ mit $s_i = (f_i, v_i)$ bereit. Jedes Tupel (f_i, v_i) beschreibt eine Frequenz f_i und Spannung v_i mit $f_i < f_j$ für $1 \leq i < j \leq m$. Ein Task τ_i wird spezifiziert durch die Anzahl der maximalen Ausführzyklen $\check{C}_i$ für den Worst-Case und einer Periode T_i. Für die relative Deadline D_i eines Tasks wird angenommen, dass sie der Periode T_i entspricht. Zur Laufzeit gibt $\check{R}_i$ die Anzahl der Restzyklen bis zur Terminierung einer Taskinstanz an. Gesucht ist nun ein Schedulingverfahren, welches für eine gegebene Menge $\Gamma = \{\tau_i(\check{C}_i, T_i), i = 1, .., n\}$ von n Tasks den

Energieverbrauch minimiert. Der Scheduler soll dabei online[1] und präemptiv[2] auf einem Prozessor mit diskreten Frequenzen ausgeführt werden.

2.1.2 Grundlegende Verfahren

In der Literatur findet man zwei unterschiedliche Ansätze für dynamische Frequenzskalierung (DFS): Inter-Task- und Intra-Task-DFS, z.B. [2] und [6]. Bei Inter-Task-DFS wird die Prozessorfrequenz immer bei Taskwechsel bzw. Kontextwechsel (z.B. Präemptionen) angepasst. Wird bei der Beendigung einer Taskinstanz beispielsweise festgestellt, dass sie weniger Ausführzeit in Anspruch genommen hat als für den Worst-Case vorgesehen, so kann diese ungenutzte Rechenzeit bzw. dieser *Slack* einem nachfolgenden Task weitergereicht werden. Intra-Task-DFS Techniken berücksichtigen ungenutzte Rechenzeit schon während der Taskausführung, die durch unterschiedliche Ausführpfade bei der Laufzeit auftreten können. Wenn beispielsweise aufgrund bedingter Ausführung Codeblöcke ausgelassen werden oder Schleifen früher terminieren, lässt sich die Anzahl der gesparten Prozessorzyklen zur Verringerung der Frequenz ausnutzen. Intra-Task-Techniken im Gegensatz zu Inter-Task-Verfahren haben den Vorteil, dass sie ungenutzte Rechenzeit nicht erst nach Beendigung einer Taskinstanz feststellen und damit potentiell effizienter arbeiten können, allerdings eine aufwändige Veränderung des auszuführenden Codes voraussetzen. Eine solche Instrumentierung wird durch den Einsatz von so genannten *Frequency Scaling Points* (FSPs) erreicht, die im ausführbaren Code verankert werden. Ein FSP signalisiert hierbei dem Scheduler, wie viele Ausführzyklen durch die Wahl eines Ausführpfades eingespart wurden. Eine weitere Besonderheit ergibt sich für beide Verfahren bei diskreten Frequenzen: Da die optimalen Frequenzen (wie bei kontinuierlichen Frequenzspektren) nur selten zur Verfügung stehen, muss auf andere Frequenzen ausgewichen werden, was sich wiederum auf die Ausführzeit des Tasks auswirkt.

2.1.3 Kombiniertes Inter- und Intra-Task Frequency Scaling

Die Frage ist nun, ob durch ein hybrides Verfahren die Energieeffizienz verbessert werden kann. Die Schwierigkeit ist hierbei, dass bisherige Intra-Task-Techniken wegen eines statischen Instrumentierungsschemas nicht ohne Weiteres auf Systeme mit variabler Anzahl von Tasks angewendet werden können: Bisherige Intra-Task-Techniken führen nach Verzweigungen des Kontrollflusses, die eine Einsparung von Rechnenzyklen gegenüber dem worst-case nach sich ziehen, feste, zur Compilezeit berechnete Frequenzänderungen durch. Das feste Skalierungsschema verhindert aber, dass ungenutzte Rechenzeit vorausgehender Tasks berücksichtigt werden kann.

Um dieses Problem zu lösen, haben wir ein verändertes Verfahren vorgeschlagen, welches die Restausführzyklen $\check{R}_i$ mit Hilfe eines prozessorinternen Zyklenzählers $\check{C}_{act}$ berechnet und die zu erreichende Deadline anhand der relativen Deadline der aktuellen Task und der bisher ungenutzten Rechenzeit vorangehender Tasks gemäß eines übergeordneten Taskschedulings bestimmt. Diese Vorgehensweise bietet weitere wesentliche Vorteile gegenüber bestehenden Intra-Task-Verfahren. Es ist hiermit möglich, auch gemeinsam genutzten Code zu instrumentieren, was für Standardbefehlsbibliotheken von Betriebssystemen besonders wichtig ist. Außerdem kann selbst innerhalb geschachtelter Schleifen die Restzyklenzahl effizient bestimmt werden, welche bei bisherigen Intra-Task-Verfahren aufwändig berechnet werden musste. Für das übergeordnete Taskscheduling

[1] Die Schedulingentscheidung wird zur Laufzeit (*online*) vorgenommen, um auf ein variierendes Laufzeitverhalten der Tasks und auf dynamisch hinzukommende Tasks reagieren zu können.

[2] *Präemption:* Möglichkeit eines Schedulers, den laufenden Task zu unterbrechen, um eine Aufgabe mit höherer Priorität auszuführen.

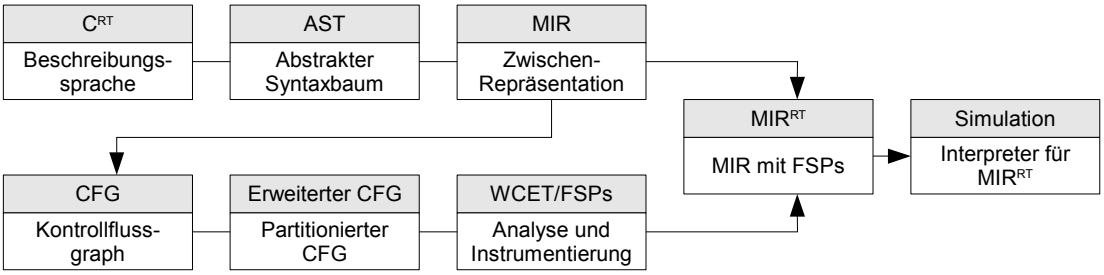

Abbildung 1: Compiler- und Simulationsumgebung: Die Abbildung zeigt die wesentlichen Schritte wie aus einer Taskbeschreibung in der Sprache C^{RT} eine interpretierbare Zwischensprache MIR^{RT} für die Echtzeit-Simulation erzeugt wird.

wurde das *Earliest-Deadline-First*-Verfahren (EDF) ausgewählt [3], um auch bei hoher Prozessorauslastung Schedulbarkeit garantieren zu können.

2.1.4 Optimistische Frequenzwahl

Da im betrachteten Modell nur diskrete Frequenzen vorhanden sind, kann eine vom Scheduler berechnete Frequenz f in der Regel nicht direkt ausgewählt werden. Im ursprünglichen Verfahren wurde in diesem Fall stets die nächst höhere Frequenz $f_i = \min\{f_i \mid (f_i, v_i) \in S, f_i \geq f\}$ benutzt, um weiterhin Echtzeitausführung im Worst-Case garantieren zu können. Dieses Vorgehen hat allerdings zwei Nachteile: zum Einen entsteht durch Wahl einer höheren Frequenz wieder unerwünschter Slack und zum Anderen wird nicht berücksichtigt, dass Tasks in einem realen System nur selten Worst-Case-Verhalten aufzeigen.

Hier konnte eine deutliche Verbesserung durch eine *optimistische Frequenzwahl* erzielt werden, indem die Taskausführung in zwei Abschnitte mit einer zuerst niedrigen und dann gegebenenfalls mit einer höheren Frequenz aufgeteilt wird. Benötigt ein Task seine komplette Rechenzeit so ist weiterhin gewährleistet, dass Deadlines eingehalten werden. Terminiert die Taskinstanz früher als erwartet, so kann durch Benutzung einer zuerst niedrigeren Frequenz Energie eingespart werden.[3]

2.1.5 Scheduler-Overhead

Bei vielen Modellen wird der Zusatzaufwand, der für eine Schedulingentscheidung notwendig ist, nicht betrachtet. Allerdings kann eine solche Idealisierung bei harten Echtzeitsystemen gravierende Folgen mit sich bringen. Um ein solch unerwünschtes Verhalten ausschließen zu können, wurde das Schedulingmodell entsprechend erweitert. Ist die Anzahl der Zyklen $\check{C}_{\mathrm{CS}}$ bekannt, die bei einem Kontextwechsel im schlimmsten Fall ausgeführt werden, so kann mit Hilfe der Taskperioden T_i eine Zusatzauslastung U_{CS} nach oben abgeschätzt werden:

$$U_{\mathrm{CS}} = \frac{2\check{C}_{\mathrm{CS}}}{f_{\min}} \sum_{i=1}^{n} \frac{1}{T_i}$$

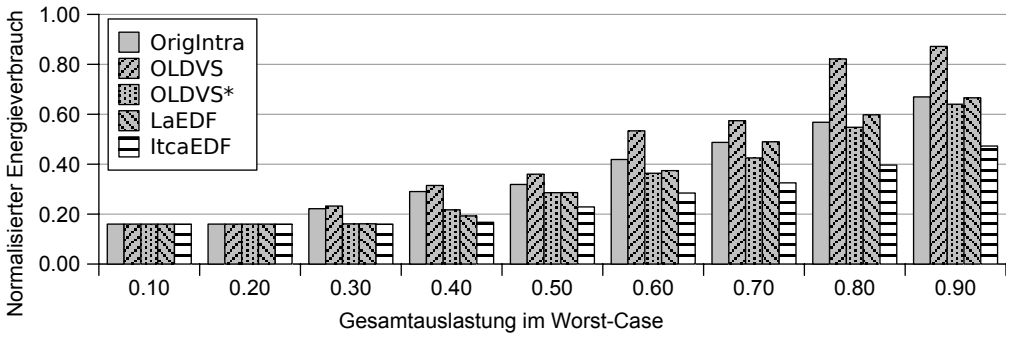

Abbildung 2: Normalisierter Energieverbrauch für zwei Tasks und je einer Durchnittslaufzeit mit dem Faktor 0.4 bis 0.8 zur Worst-Case-Ausführzeit.

2.2 Experimente und Ergebnisse

2.2.1 Compiler- und Simulationsumgebung

Für die Experimente wurde eine Compiler- und Simulationsumgebung implementiert, welche die Energieeffizienz für verschiedene Taskmengen und Prozessorkonfigurationen ermittelt (Abbildung 1). Da unser Ansatz unbenutzte Rechenzeit innerhalb des Tasks (Intra-Task) aufgrund ausgelassener Ausführpfade ausnutzt, war es notwendig den Kontrollfluss der Tasks zu modellieren. Das Taskverhalten wird deshalb in einer Sprache C^{RT} (aufbauend auf ANSI C) beschrieben, um Kontrollfluss, Tasksemantik und zeitliche Eigenschaften spezifizieren zu können. Verschiedene Syntaxerweiterungen erlauben es Quellcode mit so genannten *Flow Facts* wie Iterationsobergrenzen für Schleifen zu annotieren. Nach zahlreichen Compilerschritten, pfadbasierter WCET-Analyse und Instrumentierung der Tasks mit FSPs entsteht eine interpretierbare Repräsentation MIR^{RT} eines Tasks, die für die Simulation benutzt werden kann. Es sei hier angemerkt, dass dieses Vorgehen für spezifische Architekturen angepasst werden kann, indem das kanonische MIR^{RT}-Format weiter in Assembler übersetzt und daraus das Zeitverhalten bestimmt wird.

2.2.2 Ergebnisse

In der Simulation wurde unser Schedulingverfahren *ItcaEDF (Intra-Task Characteristics Aware EDF)* [8] mit zahlreichen bestehenden Ansätzen aus der Literatur verglichen, unter anderem mit OrigIntra [6], OLDVS [2], OLDVS* [1], LaEDF [4]. Bei einer Simulation mit beispielsweise zwei Tasks und einem Verhältnis zwischen 0.4 und 0.8 der tatsächlichen Ausführzeit zur Worst-Case-Ausführzeit verbesserte sich bei einer Worst-Case Auslastung des Systems von $U = 0.8$ der normalisierte Energieverbrauch um $28\,\%$ gegenüber OLDVS* und $34\,\%$ gegenüber LaEDF (Abbildung 2). Selbst bei einem hoch ausgelasteten System mit $U = 0.8$, acht Tasks und Ausführzeiten der Tasks mit einem Faktor zwischen 0.8 und 1.0 sind noch Einsparungen um $7\,\%$ gegenüber OLDVS* und $14\,\%$ gegenüber LaEDF möglich.

2.3 Ausblick

Die Ergebnisse der bisherigen Experimente innerhalb der Simulationsumgebung waren überaus vielversprechend. Dennoch sind noch weitere Anpassungen wünschenswert, um das vorgestellte

[3] Da der Energieverbrauch superlinear mit der Frequenz wächst, ist zur Minimierung der Gesamtenergie prinzipiell eine *gleichmäßig* niedrige Frequenzwahl anzustreben.

Verfahren für eine größere Anzahl realer Architekturen besser einsetzbar zu machen. Für Prozessoren, bei denen der Zeitaufwand für die Frequenzskalierung nicht vernachlässigbar ist, muss dieser zusätzlich zu dem schon berücksichtigten Aufwand für die Schedulingentscheidungen betrachtet werden. In der verbleibenden Zeit werde ich mich mit dafür notwendigen Modellerweiterungen und einer Erweiterung der Algorithmen zur Energieminimierung unter Berücksichtigung der Frequenzskalierungskosten beschäftigen.

3 Zusammenarbeit

Im Rahmen des Blimp-Projekts kam es zu einer engen Zusammenarbeit mit den Stipendiaten Matthias Sippel aus der Mikrosystemtechnik und Axel Rottmann aus der Informatik. Unser Ziel war es für die unterschiedliche Ansteuerung der angeschlossenen Sensoren und Aktoren dieses Luftschiffes, eine generische Schnittstelle für die Verfahren und Algorithmen aus der künstlichen Intelligenz bereitzustellen. Hierfür wurde eine *Hardware Abstraction Layer* (HAL) implementiert, welche den Zugriff auf die angeschlossenen Geräte durch allgemeine, einfache Routinen innerhalb einer Bibliothek bereitstellt. Auf diese Weise bleibt unsere Forschungsplattform flexibel und erweiterbar sowohl in der Konfiguration der Hardware als auch in der Anwendung der Software. Die Arbeit zum Blimp-Projekt mit Ergebnissen zur autonomen Navigation wurden in [7] und [5] veröffentlicht.

Literatur

[1] M.-S. Gong, Y. R. Seong, and C.-H. Lee. “On-line dynamic voltage scaling on processor with discrete frequency and voltage levels”. In *ICCIT ’07: Proceedings of the 2007 International Conference on Convergence Information Technology*. Washington, DC, USA, 2007, pp. 1824–1831.

[2] C.-H. Lee and K. G. Shin. “On-line dynamic voltage scaling for hard real-time systems using the EDF algorithm”. In *RTSS ’04: Proceedings of the 25th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 319–327.

[3] C. L. Liu and J. W. Layland. “Scheduling algorithms for multiprogramming in a hard-real-time environment”. *Journal of the Association for Computing Machinery (ACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[4] P. Pillai and K. G. Shin. “Real-time dynamic voltage scaling for low-power embedded operating systems”. In *SOSP ’01: Proceedings of the eighteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2001, pp. 89–102.

[5] A. Rottmann, M. Sippel, T. Zitterell, W. Burgard, L. Reindl, and C. Scholl. “Towards an experimental autonomous blimp platform”. In *Proceedings of the 3rd European Conference on Mobile Robots (ECMR)*. Freiburg, Germany, 2007, pp. 19–24.

[6] D. Shin, J. Kim, and S. Lee. “Intra-task voltage scheduling for low-energy hard real-time applications”. *Design & Test of Computers, IEEE*, vol. 18, pp. 20–30, 2001.

[7] M. Sippel, A. Rottmann, T. Zitterell, B. Steder, C. Scholl, W. Burgard, and L. Reindl. “Multisensor-Navigation für autonome Flugroboter”. In *Sensoren und Messsysteme 2008*. ser. VDI-Berichte 2011, Ludwigsburg, Germany, 2008, pp. 667–676.

[8] T. Zitterell and C. Scholl. “Improving energy-efficient real-time scheduling by exploiting code instrumentation”. In *Proceedings of the International Multiconference on Computer Science and Information Technology*. 2008, accepted.