Speicheroptimierungstechniken für Energieeffiziente Eingebettete Systeme[1,2]

Manish Verma

Informatik 12, Universität Dortmund,
Otto-Hahn Strasse 16,
44225 Dortmund, Germany,
manish.verma@uni-dortmund.de

Abstract: Das Speichersystem ist als Flaschenhals der Leistung von Systemen der Informationsverarbeitung erkannt worden. Dementsprechend spricht man auch von der sog. *Memory Wall*, der Wand gegen die man beim Versuch weiterer Geschwindigkeitssteigerungen läuft. Darüber hinaus ist das Speichersystem in vielen Systemen für einen großen Teil des Energieverbrauchs verantwortlich. Dementsprechend wird der Einsatz eines hoch optimierten Speichersystems als kritisch für die Einhaltung von stringenten Entwurfsrandbedingungen insbesondere von Eingebetteten Systemen angesehen, v.a. bei Produkten der Konsumelektronik. Diese Dissertation liefert einen Beitrag zur Milderung des *Memory Wall*-Problems durch den Einsatz neuer Speicherarchitekturen, der durch neue Optimierungsstrategien möglich wird.

1 Einleitung

Innerhalb einer relativ kurzen Zeitspanne haben sich Rechner von großen *Mainframes* zu kleinen und eleganten PCs und nunmehr auch zu energiesparenden, hoch-portablen Geräten entwickelt. Mit jeder Generation wurden die Rechner, bestehend aus Prozessoren, Speichern und Peripheriegeräten, schneller, kleiner und kostengünstiger. Beispielsweise kostete der erste kommerzielle Rechner UNIVAC I 1 Million Dollar, belegte 26 m^3 Platz und konnte 1.905 Operationen pro Sekunde ausführen [Mus]. Heute wird diese Leistung von einem Prozessor in einem elektrischen Rasierapparat überboten.

Neben der Miniaturisierung und den dramatischen Leistungssteigerungen hat auch die starke Reduktion der Preise von Prozessoren zu einer Situation geführt, in der sie in Produkte wie Autos, Fernseher und Telefone integriert werden, die in der Bevölkerung üblicherweise nicht mit Rechnern in Verbindung gebracht werden. Das damit einhergehende Verschwinden der Sichtbarkeit von Rechnern hat zu dem Begriff des verschwindenden

[1]Die Arbeit wurde unterstützt durch die Deutsche Forschungsgemeinschaft im Rahmen der Projekte Ma 943/6-3, 943/8-2 und 943/8-3 sowie durch die Europäische Gemeinschaft im Rahmen des *Networks of Excellence* ARTIST2 (http://www.artist-embedded.org).

[2]Übersetzung aus dem Englischen durch Peter Marwedel.

Rechners (engl. *disappearing computer*) geführt. Dabei verschwindet der Rechner nicht wirklich, sondern er wird „überall“ sein, aber eben weitgehend unsichtbar [Mar07].

Im täglichen Leben kommen wir mit vielen Geräten in Berührung, die digitale Prozessoren beinhalten. Dazu gehören u.a. Mikrowellengeräte, Fernseher, Mobiltelefone, digitale Kameras, MP3-*Player* und Autos. Die informationsverarbeitenden Systeme in diesen Geräten nennen wir **Eingebettete Systeme**. Eingebettete Systeme sind informationsverarbeitende Systeme, die in ein umgebendes System oder ein umgebendes Produkt eingebettet sind. Tatsächlich ist die Anzahl der Eingebetteten Systeme auf dieser Welt bereits größer als die Anzahl der Menschen.

Ein großer Teil der Eingebetteten Systeme findet sich in der Konsumelektronik. Diese Geräte kommen direkt mit technisch nicht besonders vorgebildeten Nutzern in Kontakt, und eine große Bedienungsfreundlichkeit ist daher besonders wichtig. In der vergangenen Dekade hat es ein starkes Wachstum im Bereich der Konsumelektronik gegeben und es wird erwartet, dass dieser Bereich eine der am stärksten treibenden Kräfte darstellt, welche die technische Innovation und die Wirtschaft vorantreiben [The06].

Allerdings gibt es im der Konsumelektronik einen gnadenlosen Wettbewerb, geringe Gewinnspannen pro Gerät und nur kurze Zeitspannen, innerhalb derer Geräte verkauft werden können. Darüber hinaus müssen insbesondere portable Geräte Randbedingungen hinsichtlich des Gewichts, der Batteriekapazität, des Verkaufspreises usw. berücksichtigen. Es müssen daher stringente Entwurfsbeschränkungen, beispielsweise hinsichtlich der Leistung, des Energieverbrauchs, der Vorhersagbarkeit, der Entwicklungskosten, der Fertigungskosten und der Zeit bis zum Markteintritt, eingehalten werden [Vah02]. Die folgenden Kriterien gelten als die drei wichtigsten, da sie eine direkte Auswirkung auf die Nutzerzufriedenheit haben:

1. Rechenleistung
2. Effizienz des Einsatzes elektrischer Leistung oder Energie
3. Vorhersagbarkeit im Sinne der Realzeitfähigkeit

Entwickler optimieren Systeme einschließlich deren Hardware und Software, um diese Kriterien möglichst gut zu erfüllen. Das Speichersystem wurde als ein Flaschenhals von Systemen identifiziert, und daher muss das Potenzial zu seiner Optimierung genutzt werden.

1.1 Das *Memory Wall*-Problem

Während der vergangenen 30 Jahre sind die Geschwindigkeiten von Mikroprozessoren um die beachtliche Rate von 50-100% pro Jahr gestiegen. Dagegen wuchs die Geschwindigkeit von dynamischen Speichern (DRAMs) nur um ca. 7% pro Jahr [Mac02]. Heutzutage verbringen sehr schnelle Prozessoren viele Taktzyklen damit, auf das Eintreffen von Informationen vom langsamen Speicher zu warten. Dieses Problem ist als das *Memory Wall-*

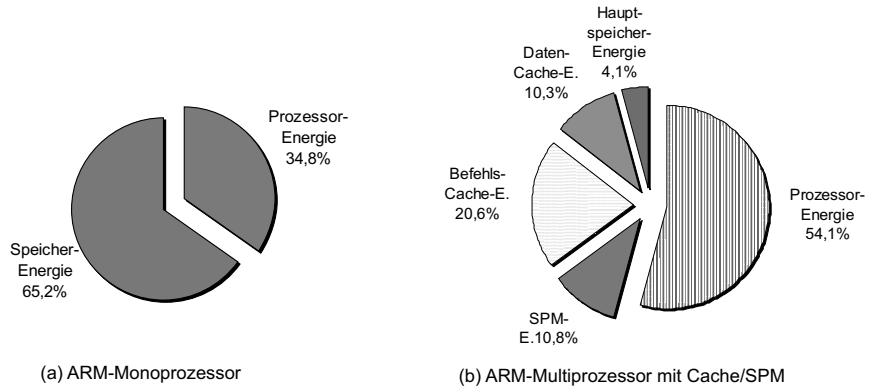

Abbildung 1: Energieverbrauch für Systeme mit (a) einem einzelnen ARM-Prozessor (b) einem ARM Multi-Processor

Problem bekannt geworden, als Problem der Begrenzung der weiteren Leistungssteigerung durch die „Wand“ fast konstanter Speichergeschwindigkeiten [WM95].

Darüber hinaus ist das Speichersystem auch einer der größten Verbraucher von elektrischer Energie bzw. ein Wandler dieser Energie in Wärme. Teilweise werden 50-70% des gesamten Energiebedarfs der Informationsverarbeitung für das Speichersystem aufgewandt [KC02, WCNM96]. Wir haben umfangreiche Analysen durchgeführt, um diese Angaben für unsere Systeme zu überprüfen. Abbildung 1 zeigt eine Übersicht über die Ergebnisse dieser Analysen für Systeme mit einem einzelnen ARM-Prozessor [ARM] sowie mit mehreren ARM-Prozessoren.

Insgesamt wurden 184 bzw. 163 Analysen durchgeführt, um den durchschnittlichen Energieverbrauch in beiden Gruppen von Systemen zu bestimmen. Es wurden Energiemodelle großer Genauigkeit benutzt [SKWM01]. Aus der Abbildung können wir ablesen, dass das Speichersystem 65,2% bzw. 45,9% der Energie der betrachteten Teilsysteme „verbraucht“ (d.h. in Wärmeenergie gewandelt) hat. Der Hauptspeicher für Systeme mit mehreren ARM-Prozessoren ist ein statischer SRAM-Speicher auf dem Chip, wohingegen für einen einzelnen Prozessor ein Speichersystem außerhalb des Prozessorchips angenommen wurde. Daher benötigen die Speicher in den betrachteten Multiprozessorsystemen einen kleineren Teil der aufgewendeten Energie.

Nach allgemeiner Überzeugung gibt es keine einzelne Lösung, die das *Memory Wall*-Problem löst. Ein wichtiger Beitrag zur Reduktion des Einflusses des Problems ist die Einführung von Speicherhierarchien. Bei diesem Ansatz werden kleine und schnelle Speicher nahe des bzw. der Prozessoren platziert und es wird versucht, häufig benötigte Informationen einer Anwendung (im Wesentlichen die sog. Arbeitsmenge (engl. *working set*)) möglichst aus Speichern in der Nähe des bzw. der Prozessoren zu verarbeiten.

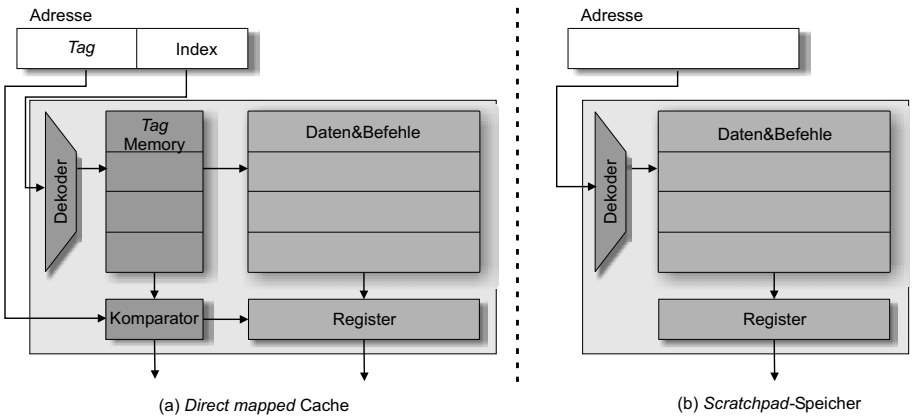

Abbildung 2: Architektur eines *direct mapped* Caches und eines *Scratchpad*-Speichers

1.2 Speicherhierarchien

In den letzten Jahrzehnten wurden Caches praktisch als Synonym für schnelle Komponenten innerhalb einer Speicherhierarchie angesehen, und sie stellen heute die Standardspeicher innerhalb der schnellen Stufen der Speicherhierarchie dar. Ihr Hauptvorteil ist, dass sie autonom arbeiten und in vielen Fällen recht effizient die Arbeitsmengen von Applikationen speichern. Ein Cache besteht neben dem Speicherplatz für die eigentlich zu speichernden Information aus einem *Tag*-Speicher, über den die Informationen zu einer bestimmten Speicheradresse erreicht werden können. Dies geschieht mit Hilfe von Vergleichern, welche überprüfen, ob die zu einer angeforderten Adresse gehörigen Informationen sich im Cache befinden (vgl. Abb. 2 (a)). Bei Eingebetteten Systemen besitzen Caches allerdings einige Nachteile. Das Lesen aus dem *Tag*-Speicher, das teilweise parallele Lesen mehrerer potenziell benötigter Informationen und die Vergleicher führen zu einem hohen Energieverbrauch [KG97], die Leistung ist häufig unzureichend, und Schranken (sog. *worst case execution times, WCETs*) für die maximal möglichen Speicherzugriffszeiten sind relativ schlecht [MWV+04, WM05b], was in Systemen mit einer vorgegebenen Antwortzeit zu Problemen führt.

Auf der anderen Seite gibt es die sog. *Scratchpad*-Speicher (engl. *scratch pad memories*, SPMs), auch *tightly coupled memories, TCMs* genannt. Ein *Scratchpad*-Speicher (vgl. Abb. 2 (b)) besteht nur aus den Speicherzellen zur Speicherung von Informationen (Daten oder Befehle) sowie Adressdekodierungslogik. Aufgrund der Abwesenheit des *Tag*-Speichers, der Vergleicher und des Fehlens von parallelen Zugriffen auf mehrere Datensätze sind SPMs flächen- und energieeffizienter als Caches [BSL+02]. Abb. 3 zeigt den Energieverbrauch pro Zugriff für SPMs verschiedener Größe und von Caches verschiedener Größe und verschiedener Assoziativität. Es werden die Werte für eine 4-fache Assoziativität (d.h. 4 paralleler Zugriffe auf Datensätze), 2-facher Assoziativität und 1-facher Assoziativität (*direct mapping*, mit DM bezeichnet) dargestellt. Aus der Abbildung wird klar, dass der Energiebedarf pro Zugriff bei SPMs immer kleiner ist als derjenige für

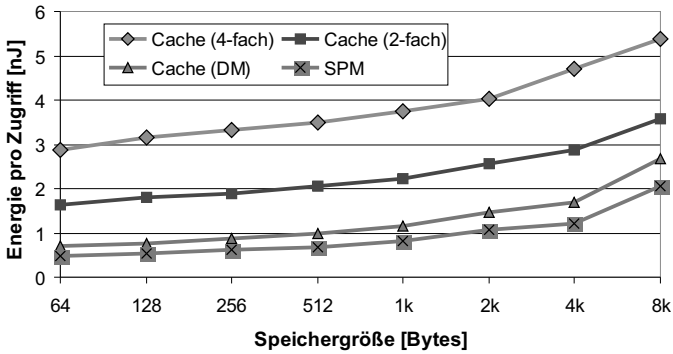

Abbildung 3: Energieverbrauch pro Zugriff für Caches und *Scratchpad*-Speicher

Cache-Speicher. Beispielsweise ist der Energiebedarf eines 2 kByte großen SPMs lediglich ein Viertel desjenigen für einen 2 kByte großen Cachespeicher 4-facher Assoziativität.

SPMs können allerdings nur dann sinnvoll genutzt werden, wenn Software-Optimierungen eingesetzt werden. Es ist eine sorgfältige Abbildung von Befehlen und Daten zu den Adressbereichen, zu denen SPMs vorhanden sind, erforderlich. Eine bereits im Compiler durchgeführte Abbildung erlaubt dabei engere Schranken für die möglichen Laufzeiten, da die Variabilität der Speicherzugriffszeiten gegenüber Cache-basierten Systemen eingeschränkt wird. Eine geeignete Compilerunterstützung ist allerdings bislang nicht vorhanden. Daher stellen wir in der Dissertation [Ver06] eine kohärente Compiler- und Simulationsumgebung vor, welche Optimierungstechniken zur Ausnutzung von SPMs enthält.

1.3 Software-Optimierung

Moderne Eingebettete Systeme führen Informationsverarbeitung mit Hilfe von Software aus. Die drei Kriterien Rechenleistung, Energieeffizienz und Vorhersagbarkeit des zeitlichen Verhaltens hängen direkt von der ausgeführten Software ab. Nach Informationen der *Technology Roadmap for Semiconductors* (ITRS) 2001 [ITR] beläuft sich der Aufwand für die Entwicklung von Eingebetteter Software auf 80 % der Entwicklungskosten eines Systems. Ursprünglich wurde die Software für Eingebettete Systeme vielfach in Assembler programmiert. Aufgrund der zunehmenden Komplexität und der kurzen Zeiten bis zur Markteinführung von Produkten erfolgt die Softwareentwicklung gegenwärtig aber überwiegend in Hochsprachen und mit Hilfe von Compilern.

In den vergangenen Jahren wurde zunehmend versucht, Leistungssteigerungen durch optimierende Compiler zu erzielen. Das beste Beispiel dafür ist der Cell-Prozessor [IBM]. Optimierende Compiler haben eine globalere Sicht auf Applikationen als Hardware-basierte Leistungssteigerungsmaßnahmen. Der nachfolgende Abschnitt gibt eine kurze Übersicht über Optimierungen, die in der Dissertation [Ver06] des Autors vorgestellt werden.

2 Speicher-Optimierungstechniken

In der Dissertation stellen wir Optimierungstechniken am Beispiel der folgenden drei unterschiedlichen Prozessortypen vor, die den Effekt der *Memory Wall* zumindest abschwächen:

1. ein ARM-Einzelprozessorsystem [ARM]
2. ein ARM-Mehrprozessorsystem [BBB+05]
3. der digitale Signalprozessor M5 [CRS+04]

Zwei der drei Prozessortypen, ARM-Einzelprozessorsysteme und der M5-Signalprozessor [Dre06], sind bereits in viele Geräte der Konsumelektronik integriert.

Die Dissertation beschreibt einen weiten Bereich an Optimierungstechniken, die hinsichtlich ihrer Analysetechniken und der unterstützten Architekturen immer komplexer werden. Die vorgeschlagenen Optimierungen transformieren die Anwendungen derart, dass sie SPMs effizient ausnutzen. Das Ziel der Optimierungen ist die Minimierung des Energieverbrauchs bei Gewährleistung einer niedrigen Laufzeit und einer hohen (zeitlichen) Vorhersagbarkeit. Alle vorgeschlagenen Techniken bestimmen den Inhalt des SPMs zur Übersetzungszeit. Ein Programm zur Analyse von Ausführungszeitschranken [Abs04] kann benutzt werden, um Schranken für die Ausführungszeit zu bestimmen. Wehmeyer hat gezeigt [WM05a], dass die Ausführungszeitschranken durch den Einsatz unserer Techniken reduziert werden können, in einem Fall um einen Faktor 8 gegenüber einem Cache-basierten System.

Die vorgeschlagenen Optimierungen sind innerhalb zweier Compiler-*Backends* sowie als Quellcodetransformationen implementiert. Der Vorteil der ersten Implementierungstechnik ist, dass genaue Informationen etwa hinsichtlich der benötigten Anzahl von Maschinenbefehlen vorliegen und so genaue Optimierungen möglich sind. Allerdings können *Backends* immer nur für eine begrenzte Anzahl von Prozessoren bereitgestellt werden. Daher wurden in jüngster Zeit zunehmend Quellcodetransformationen eingesetzt, bei denen der Compiler in einer Schleife eingesetzt werden kann, um Informationen über den jeweiligen Aufwand an Maschinenbefehlen und Speicherzugriffen zu bekommen.

2.1 Nicht überlagernde SPM-Allokation für eine SPM/Hauptspeicher-Hierarchie

Bei der einfachsten Optimierung wird eine Speicherhierarchie bestehend aus einem *Level-1 Scratchpad*-Speicher sowie einem Hauptspeicher genutzt. Es werden zur Laufzeit keine Informationen aus dem Hauptspeicher in den SPM nachgeladen. Dieser Fall wird daher nicht überlagernde Speicher-Allokation (*non overlaying memory allocation*) genannt. Das Optimierungsproblem kann je nach Randbedingungen auf ein Rucksack- oder ein *Fractional Knapsack*-Problem abgebildet werden. Das Rucksack-Problem kann mittels Ganzzahliger Programmierung gelöst werden. Für das zweite Problem wird in der Dissertation ein Greedy-Algorithmus vorgestellt.

Die experimentellen Ergebnisse sind für die beiden Fälle fast identisch. Für ein ARM-Einzelprozessor-System ergeben sich Energieeinsparungen zwischen 29% und 64% im Vergleich zu Systemen ohne *Scratchpad*. Für das ARM-Multiprozessorsystem ergeben sich Energieeinsparungen von bis zu 90%. Beim Signalprozessor M5 wurden hinsichtlich der Datenzugriffe Einsparungen von 15% bis 22% erzielt.

2.2 Nicht überlagernde SPM-Allokation für eine SPM/Cache/Hauptspeicher-Hierarchie

Eine zweite Optimierung betrachtet eine Speicherhierarchie bestehend aus einem *Scratchpad*-Speicher, einem Cache und einem Hauptspeicher. Der Scrachpad-Speicher wird als Befehlspuffer genutzt, und das Verhalten des Caches wird durch einen Konfliktgraphen modelliert. Ziel ist es, Befehlssegmente so dem SPM zuzuordnen, dass die gesamte Energieaufnahme von SPM, Cache und Hauptspeicher minimiert wird. Optimale und näherungsweise optimale Algorithmen zur Speicherallokation ohne Nachladen aus dem Hauptspeicher werden vorgestellt.

Für eine Architektur bestehend aus einem Befehlscache und einem SPM ergibt sich für eine Applikation eine 40%-ige Energieeinsparung gegenüber einer Architektur mit einem Befehlscache, wobei die Fläche für den SPM nur ein Viertel der Cachefläche betrug. Es konnte weiter gezeigt werden, dass ein SPM in Kombination mit der vorgestellten Optimierung sogar spezialiserte *preloaded loop caches* [GRV02] übertrifft. Schließlich werden beim ARM-Multiprozessorsystem Energieeinsparungen gegenüber einfacheren Optimierungen bzw. im Vergleich zum *preloaded loop cache* von 23 % bzw. 29 % erzielt.

2.3 Überlagernde SPM-Allokation für eine SPM/Hauptspeicher-Hierarchie

Bei fortgeschritteneren Optimierungsstrategien erzeugen Compiler Kopieranweisungen, welche zur Laufzeit für ein Kopieren zwischen SPM und Hauptspeicher sorgen. Auf diese Weise kann ausgenutzt werden, dass in unterschiedlichen Programmausführungsphasen unterschiedliche Speicherobjekte referenziert werden. Man kommt so zu *Overlays*, die im Gegensatz zu früheren manuell verwalteten *Overlays* durch den Compiler verwaltet werden.

In der Dissertation zeigen wir, dass die Erzeugung von SPM-Overlays Ähnlichkeiten mit dem Problem einer globalen Registerallokation innerhalb von Compilern besitzt. Das Optimierungsproblem wird in zwei Teilprobleme zerlegt. Wie in den vorangegangenen Fällen beschreiben wir optimale wie auch approximative Lösungsverfahren. Beim optimalen Verfahren wird die Ganzzahlige Programmierung zur Lösung beider Teilprobleme eingesetzt. Bei approximativen Verfahren wird ein Teilproblem mittels Ganzzahliger Programmierung, das andere mittels der *first-fit*-Heuristik gelöst.

Für ARM-Einzelprozessoren ergeben sich Energiereduktionen um 24% bzw. 20% im Vergleich zum vorherigen Ansatz bzw. im Vergleich zu einem Cache-basierten System. Für

ein ARM-Multiprozessorsystem konnte der Energiebedarf bei einem 4kByte großen SPM auf 10% der Energie eines Systems ohne SPM reduziert werden. Für den Signalprozessor M5 ergab sich eine Reduktion der Energieaufnahme in der Datenspeicherhierarchie um durchschnittlich 31%.

2.4 Datenpartitionierung und *Loop Nest Splitting*

Die nächste Optimierung unterteilt Array-Variablen in kleinere Array-Partitionen und führt anschließend eine nicht-überlagernde Zuordnung von Befehlssegmenten und Array-Partitionen zum *Scratchpad* durch. Die Optimierung bietet eine Verbesserung gegenüber bisherigen Verfahren, die Arrays nur als Ganzes dem SPM zuordnen konnten. Im Unterschied zu anderen Verfahren können reguläre wie auch irreguläre Indexausdrücke genutzt werden.

Die Partitionierung fügt in den Quellcode *if*-Anweisungen ein, um jeweils auf eine bestimmte Partition zuzugreifen. Diese zusätzlichen *if*-Anweisungen führen zu einem komplexeren Kontrollfluss und reduzieren die Leistung des Prozessor-Fließbandes. Daher wird die Datenpartitionierung mit der als *loop nest splitting* bezeichneten Transformation von H. Falk [Fal04] verknüpft, die *if*-Anweisungen entfernen bzw. in weiter außen liegende Schleifen verschieben kann. Die kombinierten Optimierungen ergeben Reduktionen des Energiebedarfs und der Laufzeit von bis zu 50 % gegenüber der nicht überlagernden Allokation für eine SPM/Hauptspeicherhierarchie.

2.5 Nutzung von SPMs für Mehrprozess-Anwendungen

Während die meisten Geräte der Konsumelektronik Mehrprozess-Anwendungen ausführen, sind die bislang bekannten Speicheroptimierungen v.a. für Einzelprozess-Anwendungen ausgelegt. Daher schlagen wir drei Ansätze vor, *Scratchpad*-Speicher von den verschiedenen Prozessen einer Applikation gemeinsam nutzen zu lassen. Bei diesen Ansätzen werden den Prozessen disjunkte und überlappende SPM-Bereiche zur Übersetzungszeit zugewiesen. Gegenüber der exklusiven Zuweisung des SPM an den energiehungrigsten Prozess werden bis zu 30% an Energie eingespart.

3 Zusammenfassung

In dieser Arbeit werden Compileroptimierungen vorgestellt, welche *Scratchpad*-basierte Speicherhierarchien für drei unterschiedliche Prozessortypen, nämlich einen ARM-Einzelprozessor, eine ARM-Multiprozessorarchitektur und einen Digitalen Signalprozessor, ausnutzen. Ein Vorteil der vorgestellten Techniken gegenüber bekannten anderen Techniken ist die Unterstützung sowohl von Daten- wie auch Befehlssegmenten, so dass die gesamte Energie bei beiden Arten von Speicherzugriffen optimiert wird. Die bekannten Ansätze zur Optimierung von Datenzugriffen betrachten die Auswirkungen der Befehlsspeicherhierar-

chie nicht ausreichend. Wir konnten beispielsweise zeigen [VM06], dass unsere Ergebnisse besser sind als die von Brockmeyer et al. [BMCC03]. Das Ziel der Optimierungen ist die Minimierung des Energiebedarfs des Speichersystems. Eine positive Wirkung haben diese Optimierungen auch hinsichtlich einer Reduktion der Programmausführungszeit sowie hinsichtlich einer Verbesserung der (zeitlichen) Vorhersagbarkeit.

Literatur

[Abs04] AbsInt Angewandte Informatik GmbH. *aiT: Worst Case Execution Time Analyzers*. http://www.absint.com/ait, 2004.

[ARM] ARM. *Advanced RISC Machines Ltd. - ARM7TDMI Reference Manual*. http://www.arm.com/pdfs/DDI0210B_7TDMI_R4.pdf.

[BBB+05] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli und M. Olivieri. MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. *Springer Journal of VLSI Signal Processing*, 41(2):169–182, Sep. 2005.

[BMCC03] E. Brockmeyer, M. Miranda, H. Corporaal und F. Cathoor. Layer Assignment Techniques for Low Energy in Multi-Layered Memory Organization. In *Proceedings of Design Automation and Test in Europe (DATE'03)*, Munich, Germany, Mar. 2003.

[BSL+02] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan und P. Marwedel. Scratchpad Memory: A Design Alternative for Cache On-chip Memory in Embedded Systems. In *Proceedings of 10th International Symposium on Hardware/Software Codesign (CODES'02)*, Colorado, USA, May 2002.

[CRS+04] G. Cichon, P. Robelly, H Seidel, M. Bronzel und G. Fettweis. Synchronous Transfer Architecture (STA). In *Proceedings of Fourth International Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS'04)*, Samos, Greece, Jul 2004.

[Dre06] Dresden Silicon. *Samira Prototype DSP*. http://www.dresdensilicon.com, 2006.

[Fal04] Heiko Falk. *Source Code Optimization Techniques for Data Flow Dominated Embedded Software*. Kluwer Academic Publishers, 2004.

[GRV02] A. Gordon-Ross und F. Vahid. Dynamic loop caching meets preloaded loop caching-a hybrid approach. *Proceedings. 2002 IEEE International Conference on Computer Design*, Seiten 446–449, 2002.

[IBM] IBM. *Cell Broadband Engine resource center*. http://www-128.ibm.com/developerworks/power/cell/.

[ITR] ITRS. *Information Technology Roadmap for Semiconductors*. http://public.itrs.net.

[KC02] M. Kandemir und A. Choudhary. Compiler-Directed Scratch Pad Memory Hierarchy Design and Management. In *Proceedings of Design Automation Conference (DAC'02)*, New Orleans, USA, Jun. 2002.

[KG97] M. Kamble und K. Ghosh. Analytical Energy Dissipation Models for Low Power Caches. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'97)*, Monterey, CA, USA, Aug. 1997.

[Mac02] P. Machanick. Approaches to Addressing the Memory Wall. Technical report, School of IT and Electrical Engineering, University of Queensland, Nov. 2002.

[Mar07] P. Marwedel. *Eingebettete Systeme*. Springer Verlag, Heidelberg, 1. Auflage, 2007.

[Mus] Computer History Museum. *Timeline of Computers*. http://www.computerhistory.org/.

[MWV+04] P. Marwedel, L. Wehmeyer, M. Verma, S. Steinke und U. Helmig. Fast, Predictable and Low Energy Memory References Through Architecture-Aware Compilation. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC'04)*, Yokohama, Japan, Jan. 2004.

[SKWM01] S. Steinke, M. Knauer, L. Wehmeyer und P. Marwedel. An Accurate and Fine Grain Instruction-Level Energy Model Supporting Software Optimizations. In *Proceedings of International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS'01)*, Yverdon-Les-Bains, Switzerland, Sep. 2001.

[The06] The Economist. *Not just a flash in the pan*. http://www.economist.com/displaystory.cfm?story_id=E1_VVSTVQQ, 2006.

[Vah02] F. Vahid. *Embedded System Design - A Unified Hardware/Software Introduction*. John Wiley & Sons, New York, USA, 2002.

[Ver06] M. Verma. *Advanced Memory Optimization Techniques for Low-Power Embedded Processors*. PhD Thesis, University of Dortmund, Dortmund, Germany, 2006.

[VM06] M. Verma und P. Marwedel. Overlay of Scratchpad Memory for Low Power Embedded Processors. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 14(8), Aug. 2006.

[WCNM96] S. Wuytack, F. Catthoor, L. Nachtergaele und H. D. Man. Power Exploration for Data Dominated Video Applications. In *Proceedings of the International Symposium of Low-Power Electronics and Design (ISLPED'96)*, Monterey, CA, USA, Aug. 1996. ACM.

[WM95] W. A. Wulf und S. A. McKee. Hitting the Memory Wall: Implications of the Obvious. *ACM Computer Archtiecture News*, 23(1), Mar. 1995.

[WM05a] L. Wehmeyer und P. Marwedel. *Fast, Efficient and Predictable Memory Accesses - Optimization Algorithms for Memory Architecture Aware Compilation*. Springer, Dordrecht, The Netherlands, 2005.

[WM05b] L. Wehmeyer und P. Marwedel. Influence of Memory Hierarchies on Predictability for Time Constrained Embedded Software. In *Proceedings of Design Automation and Test in Europe (DATE'05)*, Munich, Germany, Mar. 2005.

Manish Verma wurde am 4.9.1979 in Delhi (Indien) geboren. Im Jahre 2001 erhielt er den Grad eines *B.Tech. in Computer Science and Engineering* am *Indian Institute of Technology* in Delhi. 2006 promovierte er mit *summa cum laude* an der Universität Dortmund. Gegenwärtig arbeitet er als Compilerentwicklungsingenieur am *Altera European Technology Center* in London.

Seine Forschungsinteressen beinhalten Mikroarchitektursynthese und optimierende Compiler für Einzel- und Mehrprozessorsysteme. Er hat zahlreiche Artikel sowie ein Buch publiziert.