Retargierbare Codeerzeugung für digitale Signalprozessoren

Rainer Leupers

Universität Dortmund
Lehrstuhl Informatik 12
D-44221 Dortmund
email: leupers@ls12.cs.uni-dortmund.de

Digitale Signalprozessoren (DSPs) sind programmierbare Bausteine mit speziellen, für rechenintensive Anwendungen optimierten Befehlssätzen, welche vor allem zur Signalverarbeitung unter Echtzeitbedingungen eingesetzt werden. Aufgrund fehlender DSP-spezifischer Optimierungstechniken erzeugen derzeitige Hochsprachen-Compiler für DSPs meist sehr schlechten Code, so daß der Großteil der DSP-Software auch heute noch zeitaufwendig in Assemblersprachen entwickelt werden muß. Dies bedeutet einen erheblichen Flaschenhals in der Entwicklung eingebetteter Systeme. In dieser Arbeit werden neue Compilertechniken vorgestellt, welche die besonderen Randbedingungen im DSP-Bereich berücksichtigen. Hierzu zählen Optimierungstechniken, welche die charakteristischen Hardware-Eigenschaften von DSPs (u.a. spezialisierte Register, parallele Maschinenbefehle, separate Adreßrecheneinheiten) zur Verbesserung der Codequalität ausnutzen, mit dem Ziel, den Einsatz von Compilern auch im DSP-Bereich zu ermöglichen. Gleichzeitig sind diese Techniken hinreichend allgemein gehalten, um auf eine ganze Klasse von DSPs anwendbar zu sein. Diese Eigenschaft wird als Retargierbarkeit bezeichnet. Retargierbare Compiler helfen bei der Optimierung von Prozessorarchitekturen für gegebene Anwendungen. Das in dieser Arbeit vorgestellte Compilersystem RECORD ermöglicht die automatische Anpassung von Compilern an neue Prozessoren auf der Basis von Prozessormodellen, die in einer Hardware-Beschreibungssprache spezifiziert sind. Hierdurch wird die notwendige Brücke zwischen dem Compilerbau und dem computergestützten Entwurf integrierter Schaltungen geschlagen. Experimentelle Ergebnisse für realistische Prozessoren zeigen die praktische Anwendbarkeit der vorgestellten Techniken.

1 Einleitung

Die Integrationsdichte heutiger mikroelektronischer Schaltungen liegt bei mehreren Millionen Transistoren pro Chip. Der korrekte Entwurf derartig komplexer Schaltungen erfordert Rechnerunterstützung mit Hilfe von CAD-Werkzeugen[1]. Während vor einigen Jahren noch der Schaltungsentwurf auf der Ebene logischer Gatter (z.B. UND, ODER, NICHT) üblich war, gestatten moderne CAD-Werkzeuge die automatische Synthese von Schaltungsstrukturen aus abstrakten Verhaltensbeschreibungen, ähnlich zu Programmiersprachen. Auf diese Weise wird eine wesentlich höhere Effizienz beim Schaltungsentwurf erzielt, als dies bei Entwürfen auf Gatterebene möglich wäre.

1.1 Entwurf eingebetteter Systeme

Aufgrund der stetig steigenden Integrationsdichten ist ein wesentliches Ziel der heutigen Forschung im Bereich des Entwurfs mikroelektronischer Schaltungen die Bereitstellung von CAD-Werkzeugen zur Entwurfsautomatisierung auf *Systemebene*. Eine wichtige Klasse von mikroelektronischen Systemen sind dabei die *eingebetteten Systeme*, welche aufgrund der heutigen hohen Verbreitung von elektronischen Geräten in vielen Bereichen des täglichen Lebens zunehmend an Bedeutung gewinnen. Im Gegensatz zu "Allzweckrechnern" wie Workstations und PCs lesen und verarbeiten eingebettete Systeme physikalische Größen für feste Anwendungen. Beispiele sind Mobiltelefone, digitale Anrufbeantworter und Steuerungen in der Automobilelektronik.

Da immer komplexere eingebettete Systeme in immer kürzerer Zeit entworfen und auf den Markt gebracht werden müssen, ist die *Wiederverwendung* von Systemkomponenten eine wichtige Randbedingung beim Systementwurf. Aus diesem Grund ist derzeit ein Trend zu *software-basiertem* Systementwurf zu beobachten, während spezielle Hardware (ASICs[2]) häufig nur noch zur Beschleunigung zeitkritischer Komponenten verwendet wird. Die Software eines Systems wird dabei von programmierbaren *eingebetteten Prozessoren* ausgeführt, welche sich als Makrozellen in einem Entwurf leicht wiederverwenden lassen. Ebenso ist die Wiederverwendung von Softwarekomponenten einfacher als die von ASICs, welche stets mehr oder weniger stark technologieabhängig sind.

---

[1] computer-aided design
[2] application-specific integrated circuits

1.2 Compiler für digitale Signalprozessoren

In diesem Beitrag wird eine spezielle Klasse von eingebetteten Prozessoren behandelt, die *digitalen Signalprozessoren* (DSPs). DSPs unterscheiden sich von herkömmlichen Prozessorklassen wie CISC[3] und RISC[4] durch Maschinenbefehlssätze, welche auf die schnelle Ausführung rechenintensiver Anwendungen (z.B. Fourier-Transformation oder digitale Filter) zugeschnitten sind. Beispielsweise verfügen praktisch alle DSPs über einen *multiply-accumulate*-Befehl zur schnellen Berechung von Summen von Produkten, welche eine Grundoperation in der Signalverarbeitung darstellt.
Da DSPs in erster Linie hohe Effizienz gewährleisten müssen, sind deren Befehlssätze komplizierter als bei anderen Prozessorklassen. Es existieren gewöhnlich spezialisierte Register, welche an bestimmte Funktionseinheiten gekoppelt sind. Des weiteren wird eine Reihe von DSP-spezifischen Speicheradressierungsmodi unterstützt, und verschiedene Befehle können parallel ausgeführt werden (Abb. 1).

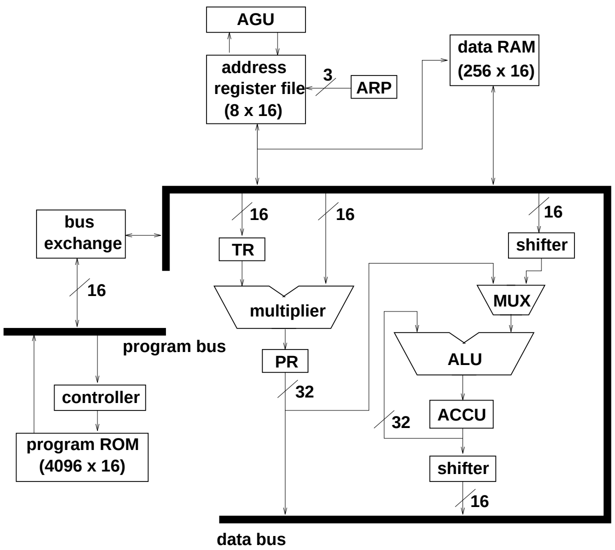


Abbildung 1: *Vereinfachte Architektur eines Texas Instruments TMS320C25 DSP. Die Spezialregister TR, PR und ACCU können teilweise parallel beschrieben werden. Ebenso kann die spezielle Adreßrecheneinheit (AGU) parallel zum übrigen Rechenwerk betrieben werden.*

Diese Eigenschaften führen dazu, daß derzeitige Compiler für Programmiersprachen wie C sehr schlechten Code für DSPs erzeugen. Im Vergleich zu ma-

[3]complex instruction set computer
[4]reduced instruction set computer

nuell geschriebenem Assemblercode ist compiler-generierter Code oft um ein vielfaches langsamer bzw. größer. Unglücklicherweise widerspricht dies völlig den Anforderungen im Bereich des Entwurfs eingebetteter Systeme, welche oft unter Echtzeitbedingungen arbeiten müssen und außerdem nur über kleine Programmspeicher verfügen. Daher wird auch heute noch der größte Teil der DSP-Software in Assembler geschrieben. Assemblerprogrammierung ist bekanntermaßen (und insbesondere bei DSPs) sehr zeitaufwendig und fehleranfällig und erschwert außerdem die Wiederverwendung und Portierung von Softwarekomponenten.
Bessere Compiler für DSPs könnten somit einen Flaschenhals beim Entwurf eingebetteter Systeme beseitigen und zu einer wesentlich höheren Produktivität führen. Hierzu ist die Entwicklung neuer, DSP-spezifischer Compilertechniken notwendig, welche über die klassische Compilertechnologie hinausgehen. Eine gewisse Vereinfachung dieses Problems ergibt sich daraus, daß – im Gegensatz zu herkömmlichen Compilern – bei DSPs keine hohe Übersetzungsgeschwindigkeit erforderlich ist. DSP-Software ist i.a. nicht sehr umfangreich, und viele Benutzer würden längere Übersetzungszeiten im Tausch gegen bessere Codequalität in Kauf nehmen. Im Extremfall könnte ein Compiler auch "über Nacht" laufen, wenn dadurch ein hinreichend kompaktes Maschinenprogramm erzeugt wird, welches gegebenen Echtzeitbedingungen und/oder Programmspeicher-Restriktionen genügt.

1.3 Retargierbare Compiler

In dieser Arbeit werden neue Compilertechniken vorgestellt, welche den besonderen Randbedingungen im DSP-Bereich entsprechen. Neben reinen Codeoptimierungstechniken liegt ein weiterer Schwerpunkt auf der *Retargierbarkeit* von Compilern. Retargierbarkeit bezeichnet die Fähigkeit eines Compilers, für verschiedene *Zielprozessoren* innerhalb einer bestimmten Prozessorklasse Maschinencode erzeugen zu können, ohne daß dazu wesentliche Änderungen am Compiler erforderlich sind. Hierzu wird dem Compiler ein *Modell* des Zielprozessors zur Verfügung gestellt, für den Code zu erzeugen ist. Im Gegensatz zu herkömmlichen Compilern ist dieses Modell nicht im Compiler selbst codiert, sondern liegt extern (i.d.R. als textuelle Beschreibung) vor. Das Modell dient somit, neben dem zu übersetzenden Programm, als zusätzliche Eingabe des Compilers.
Gerade im DSP-Bereich ist die Retargierbarkeit ein sinnvolles Konzept, da man es mit sehr vielen verschiedenen anwendungsspezifischen Prozessoren zu tun hat. Oft ist die Architektur eines Prozessors zu Beginn des Systementwurfs noch nicht völlig festgelegt, sondern es bestehen noch diverse Konfigura-

tionsmöglichkeiten, z.B. bezüglich der Anzahl der Register und Funktionseinheiten oder der Wortbreite. Die Eignung eines bestimmten Zielprozessors bzw. einer bestimmten Prozessorkonfiguration für eine gegebene Anwendung läßt sich erst nach der Erstellung des Maschinencodes detailliert beurteilen. Ein retargierbarer Compiler, welcher ein wiederholtes Übersetzen eines gegebenen Programms auf verschiedene Zielprozessoren ermöglicht, ist dabei offensichtlich ein geeignetes Werkzeug. Ohne Retargierbarkeit müßte jeweils ein spezieller Compiler für jeden Zielprozessor bzw. für jede Prozessorkonfiguration entwickelt werden.

1.4 Übersicht

Im folgenden Abschnitt wird der Stand der Technik im Bereich Compiler für DSPs umrissen. Anschließend wird ein Überblick über das RECORD-Compilersystem gegeben. RECORD ("Retargetable Compiler for DSPs") ist ein retargierbarer Compiler für eine Klasse von DSPs, in welchem die hier vorgestellten Techniken prototypenhaft implementiert sind. In Abschnitt 4 wird dargestellt, wie sich RECORD für einen gegebenen Zielprozessor anpassen läßt. Die Abschnitte 5-7 befassen sich mit den verschiedenen Codeerzeugungs- und Optimierungstechniken. Experimentelle Ergebnisse werden in Abschnitt 8 dargestellt. Zum Abschluß erfolgt eine Zusammenfassung der wesentlichen Resultate. Die detaillierte Beschreibung der Techniken in RECORD findet sich in [8].

2 Stand der Technik

Während man bei anwendungsspezifischen DSPs aufgrund fehlender Compiler meist zur Assemblerprogrammierung gezwungen ist, existieren für eine Reihe von Standard-DSPs C-Compiler. Diese erzeugen allerdings Code von inakzeptabler Qualität. In einer systematischen Untersuchung der RWTH Aachen [1] wurde gezeigt daß die Qualität (bzgl. Code-Ausführungsgeschwindigkeit und -größe) von compiler-generiertem Code oft um mehrere 100 % von der Qualität handgeschriebener Assembler-Referenzcodes abweicht. Industrielle Untersuchungen kommen zu ähnlichen Ergebnissen [2].

Ein Hauptgrund hierfür liegt darin, daß sich Hochsprachen-Konstrukte teilweise nur schwer in effiziente DSP-Assemblerbefehle umsetzen lassen. Auch die Vielzahl aus dem Compilerbau bekannter Standard-Optimierungstechniken (siehe z.B. [3, 4]) hilft hier nur begrenzt weiter, da diese Techniken meist auf einer abstrakten, maschinenunabhängigen Code-Zwischenrepräsentation auf-

setzen, während aber z.B. zur Ausnutzung von parallelen Befehlen eine detailliertere Sicht des Zielprozessors notwendig ist.
Das Konzept der retargierbaren Compiler ist bereits seit längerem bekannt. Der populärste Vertreter solcher Werkzeuge ist vermutlich der GNU C-Compiler [5], welcher erfolgreich auf eine Reihe von CISC- und RISC-Maschinen portiert worden ist. Da dieser Compiler allerdings von vornherein nur für derartige Prozessoren ausgelegt ist, erzeugen Anpassungen für DSPs (z.B. im Falle von Motorola-Prozessoren) ebenfalls schlechten Code. Darüber hinaus ist die Anpassung des GNU C-Compilers an neue Zielprozessoren recht aufwendig, so daß eine Prozessoroptimierung wie in Abschnitt 1.3 angedeutet, kaum möglich ist.
In den letzten Jahren war das Thema "retargierbare Compiler für DSPs" daher international Gegenstand intensiver Forschung, vor allem im Bereich des computergestützten Entwurfs eingebetteter Systeme. Es wurden eine Reihe von Techniken zur Prozessormodellierung und Codeoptimierung vorgestellt, welche die speziellen Gegebenheiten bei DSPs berücksichtigen. Übersichten finden sich in [6, 7, 8]. Gegenüber verwandten Arbeiten weist das im folgenden Abschnitt vorgestellte RECORD-System zwei Besonderheiten auf:

1. Es wird der komplette Übersetzungvorgang, von der Modellierung des Zielprozessors über die Befehlsauswahl bis zur Erzeugung von parallelisiertem Maschinencode realisiert. Alle in RECORD verwendeten Codeerzeugungs- und optimierungstechniken sind aufeinander abgestimmt und sind auf eine ganze Klasse von DSPs anwendbar.

2. Das Modell des Zielprozessors wird in einer echten Hardware-Beschreibungssprache (*hardware description language, HDL*) spezifiziert. Dieses Vorgehen bietet einige wichtige Vorteile. Zum einen sind die potentiellen Benutzer, d.h. Entwurfsingenieure, mit HDLs vertraut, so daß keine Notwendigkeit besteht, werkzeugspezifische Sprachen zu erlernen. Des weiteren können HDL-Modelle auch unmittelbar zur Synthese und Simulation von Prozessoren verwendet werden, was zur Vereinfachung des Systementwurfs beiträgt. Schließlich ermöglichen HDLs eine sehr flexible Modellierung auf verschiedenen Abstraktionsebenen, von der Gatterebene über die Register-Transfer-Ebene bis zur Verhaltensebene, wodurch eine hoher Modellierungskomfort erreicht wird.

3 Das RECORD-Compilersystem

RECORD umfaßt ca. 120.000 Zeilen C++ Code und ist auf einer Workstation-Umgebung unter UNIX implementiert (Abb. 2). Das System besteht aus einer graphischen Benutzeroberfläche sowie zwei Hauptkomponenten:

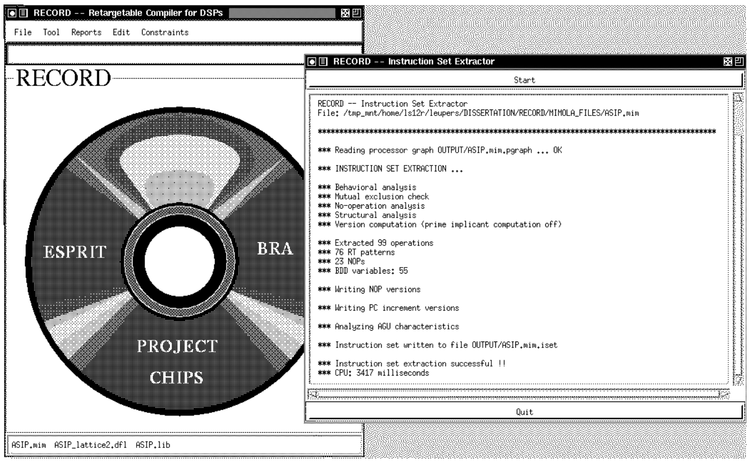


Abbildung 2: *Das RECORD-System: Benutzeroberfläche und Befehlssatz-Extraktor*

Der **Befehlssatz-Extraktor** dient zur Analyse des Zielprozessormodells. Er liest das HDL-Modell, welches auf verschiedenen Abstraktionsebenen (evtl. auch gemischt) gegeben sein kann. In der derzeitigen Version wird die Sprache MIMOLA zur Prozessormodellierung verwendet. Ein Frontend für die Standard-Sprache VHDL ließe sich allerdings auch mit vertretbarem Aufwand realisieren. Je nach Anwendung kann das HDL-Modell strukturelle Komponenten wie Register, Funktionseinheiten, Multiplexer oder Busse beinhalten, welche für die Codeerzeugung im Prinzip irrelevant sind. Hauptaufgabe des Befehlssatz-Extraktors ist neben diversen Korrektheitsprüfungen die Elimination überflüssiger Prozessor-Strukturinformationen und die "Extraktion" eines für die Codeerzeugung geeigneten Verhaltensmodells. Dieses Verhaltensmodell umfaßt alle Maschinenbefehle, die auf dem modellierten Prozessor ausgeführt werden können, sowie Informationen über die Parallelisierbarkeit dieser Befehle.

Eine Reihe von **Codeerzeugungs- und optimierungsalgorithmen** nimmt dann eine möglichst effiziente Abbildung eines gegebenen Programms auf den extrahierten Befehlssatz vor. Als Programmiersprache wird in RECORD die DSP-spezifische Sprache DFL (*data flow language*) verwendet, welche auch kommerziell in CAD-Werkzeugen genutzt wird. Gegenüber C bietet DFL bessere Möglichkeiten zur Programmierung von DSP-Software. Das DFL-Quellprogramm wird in ein Zwischenformat überführt, und es wird eine vorläufige Befehlsauswahl vorgenommen. Anschließend folgt eine Scheduling-Phase, in der die ausgewählten Befehle in eine sequentielle Reihenfolge gebracht werden. In beiden Phasen ist die integrierte Allokation der Spezialregister (vgl.

Abschnitt 1.2) besonders wichtig ("Phasenkopplung") um überflüssige Datentransfers zu vermeiden.
Die nachfolgenden beiden Phasen beschäftigen sich mit der Ausnutzung von parallelen Befehlen. Zunächst werden die Programmvariablen bestimmten Speicherplätzen zugewiesen, mit dem Ziel, die DSP-spezifische Hardware zur parallelen Adreßberechnung gut auszunutzen. Zum Abschluß erfolgt eine *Code-Kompaktierungsphase*, in der die erzeugten sequentiellen Befehle blockweise optimal zu parallelen Befehlen zusammengefaßt werden. Auch hier ist das Konzept der Phasenkopplung zur Verbesserung der Codequalität realisiert, da erst die Kompaktierung die Befehlsauswahl vervollständigt. Das Endergebnis ist ein ausführbares Maschinenprogramm für den modellierten Zielprozessor.
Die Befehlssatz-Extraktion sowie die einzelnen Codeerzeugungsphasen werden im folgenden näher erläutert.

4 Analyse des Prozessormodells

Das Retargieren des RECORD-Compilers erfolgt durch die HDL-Spezifikation des Zielprozessors und die Durchführung der Befehlssatz-Extraktion auf dem HDL-Modell. Wie oben erwähnt, kann das HDL-Modell des Zielprozessors strukturelle Informationen beinhalten, welche für die Codeerzeugung irrelevant sind. Abb. 3 zeigt als Beispiel das Teil-Schaltbild eines Prozessors, welches aus Registern, Funktionseinheiten, Multiplexern und deren Verbindungsstrukturen besteht. Der Befehlssatz-Extraktor analysiert, welche primitiven Maschinenbefehle auf einer solchen (textuell spezifizierten) Schaltungsstruktur ausgeführt werden können. Die gezeigte Struktur erlaubt bspw. die Addition der Register R1 und R4 und die Zuweisung des Resultats an R5. Per Graphdurchlauf durch die spezifizierte Prozessorstruktur werden alle derartigen Elementarbefehle extrahiert. Des weiteren wird festgestellt, welche Kontrollsignale, d.h. binäre Belegungen des Befehlswortes, für die jeweiligen Befehle vorliegen müssen. Für den Befehl "R5 := R1 + R4" müssen bspw. die Multiplexer MUX1, MUX2 und MUX3 passend angesteuert werden.

Aus den Belegungen der Kontrollsignale läßt sich anschließend auch die Information gewinnen, welche Befehle parallel ausführbar sind, denn Konflikte zwischen Befehlen, z.B. aufgrund einer Beanspruchung derselben Prozessorkomponente, spiegeln sich in den Kontrollsignalen wider. Um diese Konflikte zu analysieren, werden die Kontrollsignale jedes Befehls durch eine Boolesche Funktion $f : \{0,1\}^n \rightarrow \{0,1\}$ dargestellt, deren Variablenmenge genau den einzelnen Bits des Prozessor-Befehlswortes entspricht. In dieser Darstellung sind zwei Befehle genau dann parallel ausführbar, wenn die logische UND-

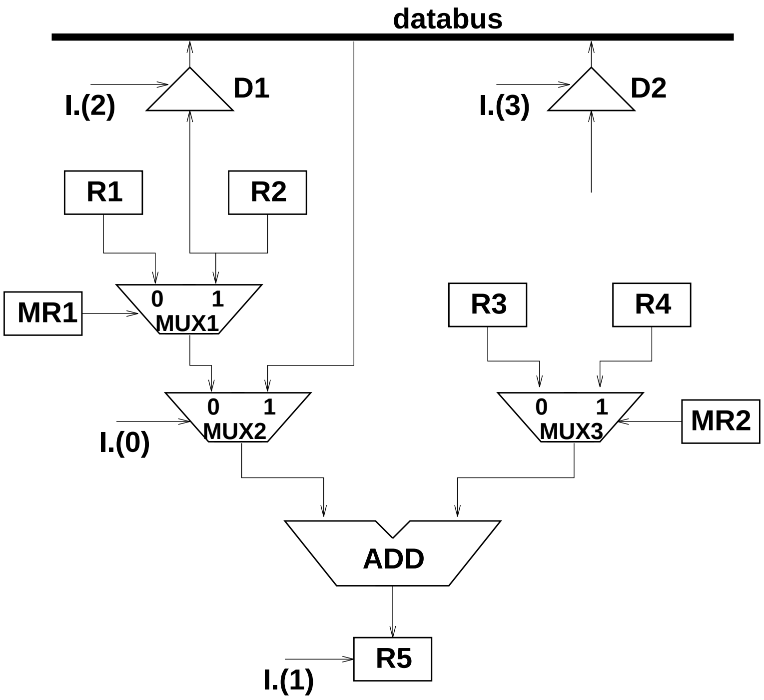


Abbildung 3: *Partielle Prozessorstruktur auf Register-Transfer-Ebene*

Verknüpfung ihrer Booleschen Funktionen erfüllbar ist. Zur effizienten Verwaltung und Manipulation Boolescher Funktionen werden in RECORD binäre Entscheidungsdiagramme (*binary decision diagrams*, BDDs) eingesetzt, welche eine Standard-Datenstruktur für diesen Zweck darstellen (vgl. [9]).

Der extrahierte Befehlssatz bildet die Basis für die im folgenden erläuterte Codeerzeugung. Daneben läßt sich die Technik der Befehlssatz-Extraktion auch für weitere nützliche Zwecke einsetzen, z.B. für einen Äquivalenztest bzgl. des realisierten Befehlssatzes von auf verschiedenen Abstraktionsebenen spezifizierten Modellen desselben Prozessors.

5 Codeerzeugung

Die Codeerzeugung nimmt die Abbildung der Zwischendarstellung des Quellprogramms in eine möglichst "kostengünstige" Menge von Maschinenbefehlen vor. Das Kostenmaß ist dabei proportional zur Ausführungszeit der Befehle. Zur Realisierung einer *retargierbaren* Codeerzeugungstechnik wurde in RECORD auf den Mustervergleich zwischen Bäumen (*tree pattern matching*) zurückgegriffen. Dies ist eine aus dem Standard-Compilerbau bekannte Technik

[4], welche im Kontext der retargierbaren Compiler den wichtigen Vorteil bietet, daß sich baumbasierte Codegeneratoren mittels sog. *Baum-Grammatiken* automatisch generieren lassen. Des weiteren eignet sich das tree pattern matching auch sehr gut zur Codeerzeugung für irreguläre Prozessorstrukturen mit spezialisierten Registern, wie sie bei DSPs meist vorliegen. Der extrahierte Befehlssatz wird von RECORD in eine äquivalente Baum-Grammatik transformiert, und mit Hilfe eines Standardwerkzeugs ("iburg" [10]) wird dann ein ausführbarer Codegenerator erzeugt.
Das Quellprogramm wird in einzelne Bäume zerlegt, welche jeweils einen zu übersetzenden Ausdruck repräsentieren. Da sich die verfügbaren Maschinenbefehle ebenfalls durch "kleine" Bäume repräsentieren lassen, kann man die Codeerzeugung als Überdeckungsproblem auf Bäumen auffassen. Der Codegenerator berechnet eine optimale Überdeckung jedes Ausdrucksbaumes durch Maschinenbefehle (Abb. 4). Zur Steigerung der Codequalität wird in RECORD

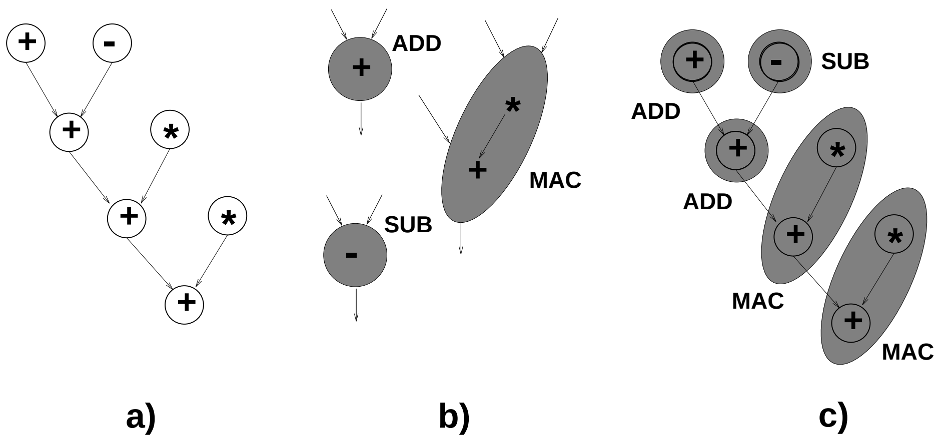


Abbildung 4: *Befehlsauswahl: a) Ausdrucksbaum, b) Baumdarstellung der Maschinenbefehle, c) optimale Überdeckung*

auch auf optionale Baum-Transformationsregeln zurückgegriffen, welche u.a. algebraische Regeln (Kommutativität, Assoziativität) zur Erzeugung alternativer äquivalenter Ausdrucksbäume ausnutzen. Unter den alternativen Ausdrucksbäumen wird dann derjenige ausgewählt, für den der schnellste Code erzeugt werden kann.
Den Abschluß der Codeerzeugung bildet eine Scheduling-Phase, in der Beschränkungen der Registerkapazitäten berücksichtigt werden. Das Ziel dabei ist es, zusätzlichen Code zur temporären Aus- und Einlagerung von Spezialregisterinhalten (*spill code*) zu minimieren. Das Ergebnis ist dann ein sequentielles Maschinenprogramm.

6 Speicheradressierung

Wie in Abschnitt 1.2 erwähnt, besitzen DSPs i.d.R. spezielle Adreßrecheneinheiten (AGUs). Diese AGUs sind insbesondere dazu geeignet, in jedem Befehlszyklus die jeweils nächste im Programm benötigte Speicheradresse parallel zu berechnen, z.B. durch Inkrementieren oder Dekrementieren eines Adreßregisters (*auto-increment/decrement*). Zur Vermeidung überflüssigen Codes zur Adreßrechnung ist es daher naheliegend, die Freiheitsgrade bei der Zuordnung von Programmvariablen zu Speicherplätzen während der Übersetzung dahingehend auszunutzen, daß die Fähigkeiten der AGU berücksichtigt werden. Beispielsweise sollten Variablen mit häufigen direkt aufeinanderfolgenden Zugriffen innerhalb des Programms im Speicher benachbart angeordnet werden, um auto-increment/decrement-Adressierung einsetzen zu können.
Zur Optimierung des Codes zur Speicheradressierung wird in RECORD eine Reihe von speziellen, meist graphbasierten Techniken eingesetzt. Aus Platzgründen wird hier exemplarisch nur die Speicherplatzzuordnung für skalare Variablen vorgestellt. Gegeben sei z.B. ein Programmabschnitt, in dem die Variablenmenge V in der Reihenfolge S zugegriffen wird, mit

$$V = \{a, b, c, d\}, \quad S = (b, d, a, c, d, a, c, b, a, d, a, c, d)$$

Die Variablen aus V sollen so im Speicher permutiert werden, daß Variablenpaare mit häufigen Nachbarschaften in S vorzugsweise auch im Speicher benachbart sind. Dies läßt sich mittels des in Abb. 5 gezeigten Zugriffsgraph-Modells realisieren. Der Zugriffsgraph besitzt einen Knoten pro Variable. Die

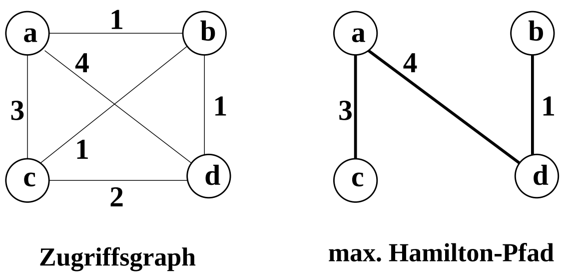


Abbildung 5: *Graphmodell zur Adreßoptimierung für skalare Variablen*

Kantengewichte bezeichnen die Anzahl der Übergänge (v, w) oder (w, v) zwischen zwei Variablen v und w in der Zugriffssequenz S. Man kann zeigen, daß jede optimale Speicheranordnung der Variablen einem maximal gewichteten Pfad über alle Knoten (*Hamilton-Pfad*) im Zugriffsgraphen entspricht. Die Berechnung eines solchen Pfades ist allerdings NP-hart, und in RECORD wird daher auf eine Heuristik zurückgegriffen, welche meist nahezu optimale

Ergebnisse produziert.

Abb. 6 veranschaulicht die Auswirkung dieser Optimierung auf die Codequalität. Das Kostenmaß ist dabei die Anzahl der Befehle, welche *nicht* durch

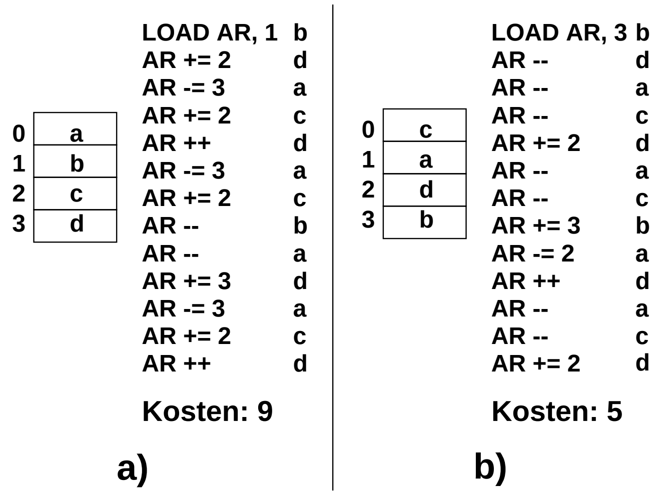


Abbildung 6: *Generierte Befehle zur Adreßrechnung (in "C-Notation") bei nicht-optimierter (a) bzw. optimierter (b) Speicherzuordnung von Variablen. AR bezeichnet ein Adreßregister.*

auto-increment/decrement implementiert werden können. Während bei einer Speicherzuordnung, die die Zugriffssequenz nicht berücksichtigt (z.B. die lexikographische Anordnung in Abb. 6 a) relativ hohe Kosten verursacht, erhält man bei der mittels des Zugriffsgraphen gewonnenen Zuordnung einen wesentlich höheren Anteil an (parallelisierbaren und daher "kostenlosen") auto-increment/decrement-Befehlen.

Für Arrays, welche ein fest vorgegebenes Speicherlayout einhalten müssen, werden in RECORD andere Adreß-Optimierungsalgorithmen eingesetzt. Alle generierten Zusatzbefehle zur Adreßrechnung werden an geeigneten Stellen in den sequentiellen Maschinencode (siehe Abschnitt 5) eingefügt. In der im folgenden beschriebenen Kompaktierungsphase wird dann der sequentielle Code zur Effizienzsteigerung parallelisiert.

7 Code-Kompaktierung

Aufgabe der Code-Kompaktierung ist es, einen gegebenen Block von sequentiellem Maschinencode optimal zu parallelisieren. In RECORD wird die Kompaktierung lokal für sog. *Basisblöcke* (d.h. Programmfragmente mit höchstens

einer Verzweigung am Ende) durchgeführt. Bei der Kompaktierung ist eine Reihe von Randbedingungen zu beachten. Zunächst dürfen nur solche Befehle parallelisiert werden, die bzgl. des Befehlsformates nicht in Konflikt zueinander stehen. Diese Konflikte sind durch die während der Befehlssatz-Extraktion gewonnenen Informationen bekannt (vgl. Abschnitt 4).
Als Folge der Codeerzeugung entstehen weitere gegenseitige Abhängigkeiten zwischen Befehlen, z.B. *Datenabhängigkeiten*. Diese repräsentieren die Tatsache, daß ein Befehl i einen Wert in ein bestimmtes Register schreibt, welcher von einem anderen Befehl j als Argument gelesen wird. Solche Abhängigkeiten resultieren in Vorrangsvorschriften der Form "Befehl i muß vor Befehl j ausgeführt werden", um die Korrektheit des Programms zu erhalten.
Die Berechnung des optimal kompaktierten Codes für einen Basisblock unter Einhaltung der Konfliktrelationen und Vorrangsvorschriften ist NP-hart. Daher wurden für dieses Problem, welches auch im Bereich der VLIW-Prozessoren[5] von Bedeutung ist, verschiedene heuristische Algorithmen entwickelt. Diese sind allerdings für DSPs nur bedingt anwendbar. Zum einen können die oft recht eigentümlichen Befehlsformate von DSPs weitere Randbedingungen (z.B. den notwendigen Ausschluß unerwünschter Seiteneffekte) hervorrufen, zum anderen möchte man gerade bei der für die Codequalität besonders wichtigen Parallelisierung von Befehlen ungern Kompromisse aufgrund suboptimaler Resultate durch Heuristiken eingehen.
Aus diesen Gründen wurde für RECORD ein neuartiges, exaktes Kompaktierungsverfahren entwickelt, welches auf ganzzahliger linearer Programmierung (*integer linear programming, ILP*) basiert. Die einzuhaltenden Randbedingungen sowie das Optimierungsziel (möglichst schneller Code) werden dabei durch ein System von linearen Gleichungen und Ungleichungen codiert, welches mit Hilfe von Standardwerkzeugen (*ILP Solver*) gelöst wird. Die berechnete Belegung der Lösungsvariablen des Gleichungssystems gibt dann exakte Auskunft über die Zuordung jedes Befehls zu einem festen Zeitpunkt im parallelisierten Programm.
So wird z.B. die Zuordnung eines Befehls i zu einem Zeitpunkt t durch eine Variable $x_{i,t} \in \{0,1\}$ repräsentiert. Für $x_{i,t} = 1$ findet die Zuordnung $i \mapsto t$ statt, ansonsten wird i einem anderen Zeitpunkt zugeordnet. Die Korrektheit der berechneten Lösung wird durch geeignete lineare Randbedingungen garantiert. So wird z.B. die Anforderung, daß jeder Befehl i genau einem Zeitpunkt zugeordnet sein muss, durch die Bedingung

$$\forall \quad \text{Befehle} \quad i: \quad \sum_{t=1}^{T} x_{i,t} = 1$$

[5] very long instruction word

ausgedrückt, wobei T die obere Schranke der Programmzeitpunkte ist. Die Anforderung, einen Befehl i vor einem anderen Befehl j auszuführen, kann man durch die Bedingung

$$x_{j,t} \quad \leq \quad \sum_{t'=1}^{t-1} x_{i,t'}$$

darstellen. Die zur Kompaktierung benötigten ILP-Probleminstanzen werden von RECORD automatisch generiert, so daß sich der Benutzer nicht mit dem Erstellen der teilweise recht komplexen Gleichungssysteme beschäftigen muß. Die berechneten Lösungen sind per Konstruktion optimal. Obwohl das ILP-Problem selbst NP-hart ist, ist die Technik aufgrund schneller ILP Solver praktisch anwendbar. Basisblöcke der Größe bis zu 50 werden für realistische Befehlssätze typischerweise innerhalb einer CPU-Minute (SPARC-20) kompaktiert. Darüber hinaus gewährleistet das ILP-basierte Verfahren eine hohe Flexibilität bzgl. des Zielprozessor-Befehlsformates, was der Anforderung der Retargierbarkeit entspricht.

8 Ergebnisse und Ausblick

Für praktische Untersuchungen wurde RECORD auf eine Reihe von anwendungsspezifischen DSPs sowie auf einen Standard-DSP (TI TMS320C25, vgl. Abb. 1) retargiert. Bei der Implementierung von RECORD wurde besonders auf die Effizienz des Retargiervorganges geachtet. Nach Fertigstellung des HDL-Zielprozessormodells benötigt das System (im wesentlichen der Befehlssatz-Extraktor) typischerweise nur wenige CPU-Sekunden zur Erzeugung der prozessorspezifischen Compilerkomponenten. Diese vergleichsweise sehr hohe Geschwindigkeit ermöglicht tatsächlich die in Abschnitt 1.3 genannte Methode der Prozessoroptimierung durch wiederholtes Compilieren auf veränderliche Zielprozessorarchitekturen zur Analyse der gegenseitigen Abhängigkeiten von Hardware und Software.

Ein wesentliches Ergebnis ist auch die verbesserte Codequalität im Vergleich zu herkömmlichen DSP-Compilern. Dies wird zwar durch eine geringe Übersetzungsgeschwindigkeit (ca. 2-5 generierte Befehle pro CPU-Sekunde) erkauft, was jedoch im DSP-Bereich nach Meinung vieler Compilerbenutzer nicht sehr kritisch ist. Für die DSPStone Benchmarks [1] konnte mittels der neuen in RECORD implementierten Optimierungstechniken der Overhead von compilergeneriertem gegenüber handgeschriebenem Assemblercode von durchschnittlich ca. 145 % auf ca. 70 % halbiert werden. Dies läßt zwar noch Raum für Verbesserungen (ideal wären 0 %), bedeutet aber dennoch einen großen Schritt vorwärts im Bezug auf die praktische Verwendbarkeit von Compilern im DSP-Bereich. Da dieses Ergebnis nicht mit prozessorspezifischen, sondern mit retargierbaren und somit für den DSP-Bereich recht allgemeinen Techniken erzielt wurde, bestehen gute Aussichten, unter Einsatz von weiteren innovativen Compileroptimierungen die bisher noch weitverbreitete Praxis der zeitaufwendigen Assemblerprogrammierung von DSPs mittelfristig ablösen zu können.

Literatur

[1] V. Zivojnovic, J.M. Velarde, C. Schläger, H. Meyr: *DSPStone – A DSP-oriented Benchmarking Methodology*, Int. Conf. on Signal Processing Applications and Technology (ICSPAT), 1994

[2] P. Paulin, M. Cornero, C. Liem, et al.: *Trends in Embedded Systems Technology*, in: M.G. Sami, G. De Micheli (eds.): *Hardware/Software Codesign*, Kluwer Academic Publishers, 1996

[3] A. Aho, R. Sethi, J. Ullman: *Compilers: Principles, Techniques and Tools*, Addison-Wesley, 1986

[4] S. Muchnik: *Advanced Compiler Design and Implementation*, Morgan Kaufmann Publishers, 1997

[5] R.M. Stallmann: *Using and Porting GNU CC* V2.4, Free Software Foundation, Cambridge/Massachusetts, 1993

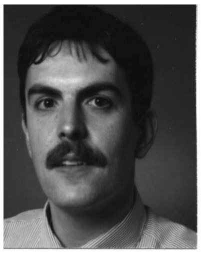
[6] P. Marwedel, G. Goossens (eds.): *Code Generation for Embedded Processors*, Kluwer Academic Publishers, 1995

[7] C. Liem: *Retargetable Compilers for Embedded Core Processors*, Kluwer Academic Publishers, 1997

[8] R. Leupers: *Retargetable Code Generation for Digital Signal Processors*, Kluwer Academic Publishers, 1997

[9] T. Theobald: *Transformationstechniken für Entscheidungsgraphen im rechnergestützten Schaltungsentwurf*, dieser Band

[10] C.W. Fraser, D.R. Hanson, T.A. Proebsting: *Engineering a Simple, Efficient Code Generator Generator*, ACM Letters on Programming Languages and Systems, vol. 1, no. 3, 1992, pp. 213-226



Rainer Leupers, geboren 1967, studierte von 1987-1992 Informatik und Volkswirtschaftslehre an der Universität Dortmund. Seit 1993 ist er dort wissenschaftlicher Mitarbeiter am Lehrstuhl für technische Informatik (Prof. Dr. P. Marwedel), wo er 1997 mit "summa cum laude" promovierte. Dr. Leupers war Stipendiat der Siemens AG und wurde mit dem Hans-Uhde-Preis (1994) und dem Dissertationspreis der Universität Dortmund (1997) ausgezeichnet. Zur Zeit setzt er seine Forschungsarbeiten im Bereich "Compiler für eingebettete Systeme" mit dem Ziel der Habilitation fort. Der Schwerpunkt liegt dabei auf der effizienten Codeerzeugung für neuere, hochgradig parallele digitale Signalprozessoren.