Codegenerierung für anwendungsspezifische Prozessoren mit evolutionären Algorithmen

Holger Arnold*

Hochschule für Technik, Wirtschaft und Kultur Leipzig,
Fachbereich Informatik, Mathematik und Naturwissenschaften

Zusammenfassung: Wir beschreiben ein Verfahren zur Codegenerierung für Prozessoren mit Befehlsparallelismus und komplexen Befehlen. Es basiert auf zwei gekoppelten evolutionären Algorithmen, von denen einer die Befehlsauswahl vornimmt und der andere Registerzuordnung und Befehlsplanung optimiert. Die Optimierung erfolgt dabei vollständig auf einer Graphendarstellung des Eingabeprogramms.

1 Einleitung

Die Zielarchitekturen die wir mit unserem Codegenerator erreichen wollen sind Prozessoren mit explizitem Befehlsparallelismus und komplexen Befehlen. Solche Prozessoren besitzen häufig sehr spezialisierte Einheiten und Register, was zu entsprechenden Einschränkungen bei der Codeerzeugung führt. Konventionelle Compiler sind für diese Prozessoren normalerweise nicht geeignet, da der von ihnen erzeugte Code die speziellen Eigenschaften solcher Architekturen nur schlecht ausnutzt [1].

Die Zielarchitektur für unseren Codegenerator soll variabel sein. Statt für jeden Prozessor einen eigenen Codegenerator entwickeln oder anpassen zu müssen, soll eine Spezifikation der Zielarchitektur Teil der Eingabedaten für den Codegenerator sein. Trotzdem für alle Architekturen der gleiche Codegenerierungs-Algorithmus verwendet wird, soll der erzeugte Code die speziellen Architekturmerkmale eines Zielprozessors möglichst gut ausnutzen.

Folgende Codegenerierungsprobleme werden von uns betrachtet: Bei der *Befehlsauswahl* werden die abstrakten Operationen des Eingabeprogramms auf konkrete Befehle des Zielprozessors abgebildet. Die *Registerzuordung* ordnet den Variablen des Eingabeprogramms und eventuellen Zwischenergebnissen die Prozessorregister zu in denen sie gespeichert werden. Bei der *Befehlsplanung* wird festgelegt, zu welchen Zeitschritten die ausgewählten Befehle ausgeführt werden.

Verfahren zur Codegenerierung, die bei den von uns angepeilten Zielarchitekturen eine hohe Codequalität erreichen wollen, müssen zwei wesentliche Bedingungen erfüllen:

1. Die Optimierung muss direkt auf der Graphendarstellung des Eingabeprogramms

Datum: 13. Oktober 2002.
*Kontaktdaten unter `http://harnold.org/`.

erfolgen. Konventionelle Compiler zerlegen den Graphen der das Eingabeprogramm darstellt in Teilbäume. Zwischen den Teilbäumen entstehen zusätzliche Datenabhängigkeiten, durch die Parallelisierungspotential verschenkt wird.

2. Die Teilprobleme Befehlsauswahl, Registerzuordnung und Befehlsplanung müssen gemeinsam optimiert werden. Konventionelle Compiler lösen die Teilprobleme sequentiell. Die spezialisierten Einheiten und Register der von uns betrachteten Architekturen führen jedoch zu starken Abhängigkeiten zwischen Befehlsauswahl, Registerzuordnung und Befehlsplanung, die bei einer getrennten Behandlung der Teilaufgaben nicht berücksichtigt werden können.

2 Bisherige Arbeiten

Es existieren bereits einige Ansätze zur Codegenerierung mit den genannten Merkmalen: In [2] und [3] werden intelligente Datenstrukturen und Heuristiken verwendet um die Codegenerierungsprobleme teilweise gekoppelt zu lösen. Das erste vollständig integrierte Optimierungsmodell für die Codegenerierung wird in [4] beschrieben. Es ist als ganzzahliges lineares Programm formuliert und wird mit Standardverfahren gelöst. In [5] wird ein interessantes Verfahren beschrieben, das auf logischer Programmierung mit Constraints basiert.

3 Ansatz

Unser Ansatz basiert auf evolutionären Algorithmen [6]. Diese haben gegenüber problemspezifischen Optimierungsverfahren den Vorteil, dass keine Heuristiken erforderlich sind, die sich bei manchen Zielarchitekturen möglicherweise als ungeeignet herausstellen. Der Vorteil gegenüber anderen allgemeinen Optimierungsstrategien wie ganzzahliger linearer Programmierung ist die wesentlich einfachere Formulierung des Optimierungsmodells.

Wir verwenden zwei gekoppelte Optimierungsprozesse. Der äußere Prozess optimiert die Befehlsauswahl. Eine Lösung der Befehlsauswahl wird als Eingabe an den inneren Prozess übergeben, der die Registerzuordnung und Befehlsplanung für diese Lösung optimiert. Das Ergebnis wird als Bewertung der Lösung wieder an den äußeren Prozess übergeben, der auf dieser Basis neue Lösungen erzeugt.

Beide Optimierungsprozesse, die Befehlsauswahl und die kombinierte Registerzuordnung und Befehlsplanung, sind als evolutionäre Algorithmen implementiert. Der Programmgraph und die Maschinenbeschreibung sind die Eingabedaten für die Befehlsauswahl. Diese arbeitet als evolutionärer Algorithmus mit einer Population von Individuen, wobei jedes Individuum eine eigene Befehlsauswahl repräsentiert. Um die Individuen der Befehlsauswahlpopulation zu bewerten, wird für jedes Individuum ein eigener Optimierungsprozess gestartet, der die Registerzuordnung und Befehlsplanung für die von diesem Individuum repräsentierte Befehlsauswahl optimiert. Diese Optimierung erfolgt wie beschrieben ebenfalls mit einem evolutionären Algorithmus, so dass für jedes Befehlsauswahlindividuum eine eigene Population für die Registerzuordnung

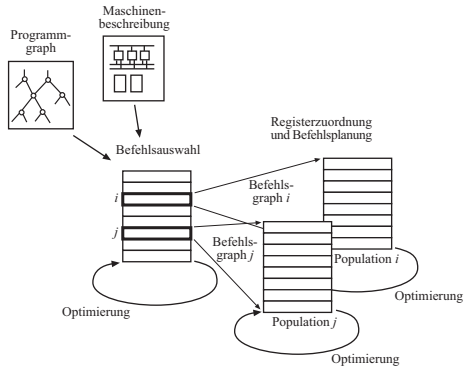

Abbildung 1: Aufbau des Codegenerators

und Befehlsplanung erstellt wird. In Abb. 1 ist dieses Verfahren für die Individuen i und j dargestellt. Die Details unseres Codegenerators werden in [7] beschrieben.

4 Auswertung

Wir haben für unsere Untersuchungen verschiedene Prozessorarchitekturen modelliert, die sich an realen Vorbildern aus dem DSP-Bereich orientieren. Sie unterscheiden sich im Grad der möglichen Parallelität und den vorhandenen Beschränkungen bezüglich der Kombination von Befehlen und Registern.

Unsere Ergebnisse zeigen, dass der Codegenerator in der Lage ist, die spezifischen Architektureigenschaften eines Prozessors gut auszunutzen. Der erzeugte Code ist entsprechend den Möglichkeiten des Zielprozessors parallelisiert. Vorhandene komplexe Befehle werden gut in den Code eingebaut. Die Codequalität ist allerdings stark von der eingesetzten Rechenzeit abhängig.

Der Rechenzeitbedarf unseres Codegenerators steigt in etwa linear mit der Problemgröße an. Diese hängt von der Größe und Struktur des Eingabeprogramms und der Komplexität der Maschinenbeschreibung ab. Der absolute Zeitbedarf ist deutlich höher als der eines konventionellen Compilers. Für die von uns betrachteten Anwendungen sind die Optimierungszeiten aber akzeptabel.

5 Zusammenfassung

Unsere Untersuchungen haben gezeigt, dass evolutionäre Algorithmen prinzipiell zur Codegenerierung geeignet sind. Anwendungen für unser Verfahren sehen wir in erster Linie bei der Codegenerierung für anwendungsspezifische Prozesoren in eingebetteten Systemen und in der Exploration des Entwurfsraumes bei der Systemsynthese.

Literatur

[1] V. Zivojnovic, J. M. Velarde, S. Christian und H. Meyr: DSPstone: A DSP-oriented Benchmarking Methodology. In: *Proceedings of the International Conference on Signal Processing Applications and Technology*, Seiten 715–720, 1994.

[2] Silvana Hanono und Srinivas Devadas: Instruction Selection, Resource Allocation, and Scheduling in the Aviv Retargetable Code Generator. In: *Proceedings of the 35th Design Automation Conference*, Seiten 510–515, 1998.

[3] Johan Van Praet, Dirk Lanneer, W. Geurts und Gert Goossens: *Processor Modeling and Code Selection for Retargetable Compilation*. ACM Transactions on Design Automation of Electronic Systems, 6[3], 2001.

[4] Tom Wilson, Gary Grewal, Ben Halley und Dilip Banerji: An Integrated Approach to Retargetable Code Generation. In: *Proceedings of the 7th International Symposium on High-Level Synthesis*, Seiten 70–75, 1994.

[5] Rainer Leupers und Steven Bashford: *Graph Based Code Selection Techniques for Embedded Processors*. ACM Transactions on Design Automation of Electronic Systems, 5[4]:794–814, 2000.

[6] David E. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[7] Holger Arnold: *Codegenerierung für variable Zielarchitekturen mit evolutionären Algorithmen*. Diplomarbeit, Fachbereich Informatik, Mathematik und Naturwissenschaften, HTWK Leipzig, 2001.