Seminarausarbeitung
Zur Lehrveranstaltung 185.272
„Grundlagen methodischen Arbeitens“
im WS 2006/2007

# Automatic Generation of Peephole Optimizations

bearbeitet von

Alexander Eisl
Matrikelnummer: 0250266, Studienkennzahl: E531

7. Jänner 2007

Technische Universität Wien
Fakultät für Informatik
Institut für Computersprachen
Arbeitsbereich Programmiersprachen und Übersetzer

Lehrveranstaltungsleiter: A.o. Univ.Prof. Dr. Anton Ertl

## Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe, und ich diese Arbeit zuvor keiner anderen Stelle oder Institution als Studiums- oder Prüfungsleistung vorgelegt habe.

Wien, 7. Jänner 2007

## Einleitung

Ziel dieser Arbeit ist es dem Leser einen Einblick in den Aufbau und die Struktur wissenschaftlicher Arbeiten zu geben. Dies geschieht anhand einer im Rahmen der PLDI/CC-Konferenz veröffentlichten Arbeit von Jack. W Davidson und Christopher W. Fraser mit dem Titel „Automatic Generation of Peephole Optimiziations“.

Diese Seminararbeit gliedert sich wie folgt in 3 Teile. Im ersten Teil werde ich auf die von den Autoren behandelte Problemstellung eingehen. Der Zweite Teil beschäftigt sich mit dem Hauptteil des ursprünglichen Artikels und bringt dem Leser die Methoden näher, mit denen die Autoren die Problemstellung behandelt haben. Die im Rahmen einer „Best Paper“ Serie zusätzlich erschienene Retrospektive bildet den dritten und abschließenden Teil der Arbeit. Hier werde ich auf die von den Autoren genannte Bedeutung ihrer ursprünglichen Arbeit für die wissenschaftliche Forschung eingehen.

## Übersetzungen

Der in dieser Seminararbeit behandelte Artikel ist in englischer Sprache abgefasst. Soweit englische Fachausdrücke gängiger und verständlicher sind als deutsche Übersetzungen werden diese verwendet.

## Motivation

Der Artikel „Automatic Generation of Peephole Optimizations“ beschäftigt sich mit der automatischen Erstellung von Mustern für peephole optimizer (PO). Peephole optimizer werden dazu verwendet, suboptimale Code-Sequenzen durch optimale zu ersetzen um so Redundanzen zu vermeiden. Diese entsteht unter anderem dadurch, dass durch das Aneinanderfügen von Befehlen ungewollte Redundanzen entstehen.

So kann beispielsweise die Code-Sequenz

`i = j; if (i >0)` ... in folgenden Code kompiliert werden:

```
move j, i
test i
bne L1
```

Für Maschinen bei denen im Rahmen des Move-Befehls condition codes gesetzt werden, ist der Test allerdings redundant (Davidson und Fraser 1984).

Der Artikel baut auf grundlegenden Arbeiten über klassische POs auf (vgl. McKeeman). Das Problem klassischer POs besteht allerdings darin, dass die Optimierungen händisch erstellt werden müssen, und es außerdem schwierig ist, derartige Verbesserungsmöglichkeiten zu bestimmen. Die Autoren nennen als Schwierigkeiten unter anderem die Programmiersprachenabhängigkeit derartiger Verbesserungsmethoden. Weiters sind die Verbesserungsmöglichkeiten von der Architektur abhängig, weshalb die vorgeschlagene Lösung auch einen wesentlich geringen Arbeitsaufwand bei der Erstellung von Optimierungen für neue Architekturen zur Folge hat.
Das Problem wird dabei von den Autoren hauptsächlich anhand von Beispielen dargestellt. Dies geschieht allerdings bereits im Hauptteil der Arbeit, in dem auch kurz die Funktionsweise des von Davidson und Fraser entwickelten „retargetable peephole optimizers“ vorgestellt wird. Der von ihnen vorgestellte neue Ansatz beruht auf dieser Lösung. Diese Art von PO erstellt die Optimierungen zwar selbständig, benötigt dadurch aber wesentlich mehr Zeit beim Kompilieren. Der hauptsächliche Vorteil der von Autoren vorgeschlagenen Lösung besteht nun darin, dass dieser zusätzliche Aufwand zur compile-compile-Zeit durchgeführt wird, und beim Kompilieren neuer Programme auf gespeicherte Patterns zugegriffen wird. Somit werden die Vorteile beider Systeme verbunden, es kommt zu einem erheblich verringerten Arbeitsaufwand und die Zeit, die zum Kompilieren benötigt wird, ist nicht viel schlechter als bei klassischen POs.

## Hauptteil

Der Hauptteil der Arbeit gliedert sich in mehrere Abschnitte. Im ersten Abschnitt wird kurz die Funktionsweise des retargetable peephole optimizers dargestellt, der zum Verständnis der folgenden Teile notwendig ist. Diese Darstellung geschieht hauptsächlich anhand von Beispielen, für einen detaillierten Überblick wird der Leser auf weitere Literaturquellen verwiesen [3,4]. Weiters behandeln die Autoren das Geschwindigkeitsproblem, das bei der Verwendung dieser Art von POs auftritt, ohne allerdings Beispiele oder Tests zu nennen, es wird lediglich auf Literaturquellen verwiesen.

Im darauf folgenden Teil behandeln die Autoren die automatische Erstellung der oben erwähnten Patterns. Die Erstellung dieser Patterns geschieht durch das Verhalten von PO auf einem „Trainings-Set“ (vgl. Davidson an Fraser 1984). Diese Darstellung der Funktionsweise wird hauptsächlich durch Beispiele, sprich mögliche Patterns, dargestellt. Die Korrektheit ist relativ einfach nachvollziehbar, auf Beweise wird allerdings verzichtet. Außerdem gibt es keine generellen Theoreme über die Möglichkeit der Erstellung derartiger Patterns und ihre Korrektheit.

Der vierte Abschnitt der Arbeit behandelt das Hash-Verfahren, das HOP zum Vergleichen von Codezeilen mit gespeicherten Patterns verwendet.

Um die Effektivität ihrer Implementierung zu demonstrieren, haben die Autoren einen Testlauf durchgeführt und die Ergebnisse dargestellt. Aufgrund der „Einfachheit typischer Programme“ (Knuth 191) ist es laut den Autoren möglich, mit einer ausreichend großen Testumgebung genügend viele Patterns zu generieren. Aufbauend auf dieser Erkenntnis vergleichen die Autoren das Ergebnis von HOP mit dem von PO.

Die Arbeit lässt sich daher als eine mehr praktisch ausgerichtete Arbeit klassifizieren. Es gibt keine theoretische Beweisführung, der Beitrag des vorgestellten Verfahrens wird anhand von Versuchen und Vergleichen dargestellt.

## Zusammenfassung und Schlussfolgerung

Da die Arbeit keine explizite Zusammenfassung enthält, und die Autoren auch im Hauptteil nicht darauf eingehen, kann eine Aussage, wie die Autoren die Bedeutung ihrer Arbeit einschätzen nur im Rahmen der Retrospektive sinnvoll beantwortet werden. Die Autoren beenden ihre Arbeit mit einem Überblick über alternative Anwendungsmöglichkeiten des vorgestellten Verfahrens.

## Verwandte Arbeiten

Die Arbeit ist die letzte in einer Reihe von Arbeiten, die von den Autoren zu diesem Thema angefertigt wurden. Folglich bezieht sie sich recht häufig auf früher erstellte Arbeiten. Weiters baut sie auf einigen Arbeiten zu klassischen POs auf, welche die Grundlage für das von den Autoren entwickelte Verfahren bilden. Ein weiterer Teil der Arbeiten bezieht sich auf retargetable peephole optimizer, die den Ausgangspunkt für die automatische Erstellung der Patterns liefern.

Zu den Arbeiten über die genannten klassischen PO zählen zum Beispiel:

- J. T. Bagwell, Jr. in „Local Optimizations“
- D.A. Lamb in „Construction of a Peephole Optimizer“
- W. M. McKeeman „Peephole Optimizer“
- W. Wulf, R. K. Johnsson, C. B. Weinstock, S. O Hobbs und C. M. Geschke
  „The Design of an Optimizing Compilier“

Die automatische Erstellung von Optimierungen und die anschließende Verwendung schneller, klassischer POs ist eine Möglichkeit um die Vorteile von retargetable peephole optimizern (RPOs) und klassischen peephole optimizern zu vereinen. Es gibt allerdings auch andere Ansätze um schnellere RPOs zu erstellen, wobei von den Autoren nur auf die relevante Literatur verwiesen wird. Generell geben die Autoren keine besonderen Stellungnahmen zu verwandten Werken ab, sie werden nur im

laufe der Arbeit erwähnt, wenn auf Ergebnisse anderer Artikel verwiesen wird. Allgemein ist anzumerken, dass die Arbeit den für wissenschaftliche Arbeiten aus dem Bereich der Informatik typischen Abschnitt über verwandte Arbeiten nicht enthält.

## Retrospektive

Die Arbeit der Autoren ist insbesondere beachtlich, als sie Einzug in einige bekannte Compiler, darunter den weit verbreitet GNU C Compiler GCC gefunden hat. Obwohl bereits einige Jahre seit dieser Arbeit vergangen sind, stellt dieser Ansatz zum automatischen Generieren von Patterns für peephole optimizer immer noch einen wichtigen Bestandteil dieses Compilers dar. Das von den Autoren entwickelte Verfahren zeichnet sich durch mehrere interessante Vorteile aus. Erstens ist der verwendete Mechanismus plattformunabhängig und erlaubt daher maschinen-unabhängigen Optimierern das Manipulieren und Optimieren maschinen-abhängigen Codes (vgl. Davidson und Fraser). Zweitens wurde mit dieser Arbeit ein wichtiger Beitrag zu den retargetable POs geleistet, da durch das Speichern von Patterns die Geschwindigkeit signifikant erhöht wurde.

Die heutige Forschung beschäftigt sich immer noch mit der Arbeit an peephole optimizern. Die Autoren nennen unter anderem Arbeiten, die sich mit spezifischen Nachteilen von PO und HOP auseinandersetzten, unter anderem langsamen Kompiliergeschwindigkeiten. (vgl. Fraser und Wendt, McKenzie).

Ein weiterer Compiler der auf den Erkenntnissen dieser Arbeit aufbaut ist VPO von der University of Virginia. VPO wurde unter anderem für einige kommerzielle Compiler genutzt und findet auch heute noch Verwendung. VPO wurde außerdem von einigen Forschern verwendet und für Codegenerierung auf den verschiedensten Architekturen verwendet. Außerdem verwendet die Real-Time Comunity VPO zur „timing analysis“ und „cache performance“ (vgl. Retrospektive).

Die Autoren finden es besonders interessant, dass eben diese alten Technologien (die Arbeit über peephole optimizer wurde im Jahr 1984 verfasst) noch heute verwendet werden, obwohl dieses Gebiet einem enormen technischen Fortschritt ausgesetzt ist. Es bleibt zu beobachten, wie sich speziell die beiden Compiler GCC und VPO in Zukunft entwickeln werden.

## Anhang des Artikels

Die Autoren haben einige der Beispiele in den Anhang ausgelagert. Auch hier wird allerdings kein allgemeiner Fall vorgestellt, sondern die Funktionsweise der Optimierung für einen bestimmten Fall vorgestellt. So zeigen sie die Übersetzung und anschließende Optimierung von

```
j = i + 4
```

Sie geben dabei den *postfix intermediate code* und den entsprechenden *object code* an (vgl. Davidson und Fraser 1984)

```
     postfix              object code
1.   push i          r[2] = m[i]
2.   pushc 4         r[3] = 4
3.   add             r[2] = r[2] + r[3] (r[3] dead)
4.   pop j           m[j] = r[2] (r[2] dead)
```

und zeigen anschließend die Optimierung dieses Codes. Diese Vorgehensweise ist den Beispielen in der restlichen Arbeit ähnlich.

## Zusammenfassung

Der Artikel „Automatic Generation of Peephole Optimizations“ ist eine sehr praktisch ausgerichtete Arbeit, mit wenig bis gar keinen theoretischen Elementen. Die Arbeit vereint die Vorteile von zwei in früheren Arbeiten vorgestellten Verfahren, den klassischen und den „retargetable“ POs.

Obwohl die Arbeit bereits vor, für die Informatik, vielen Jahren verfasst wurde, wird das vorgestellte Verfahren noch heute verwendet, zum Beispiel im Gnu C Compiler.

### Literatur


J. T. Bagwell, Jr. *Local Optimizations,* STGPLAN Notices 5, 7 (July 1970), 52-66.

J.W. Davidson und C.W. Fraser, *The Design and Application of a Retargetable Peephole Optimizer,* ACM Trans. Prog. Lang. and Systems 2, 2 (April 1980), 191-202.

J.W. Davidson und C.W. Fraser, *Automatic Generation of Peephole Optimazations,* ACM SIGPLAN, Best of PLDI 1979-1999, 1999

J. W. Davidson, *Simplifying Code Generation Through Peephole Optimization,* PhD dissertation, University of Arizona, December 1981.

D. E. Knuth, *An Empirical Study of Fortran Programs,* Software-Practice & Expirience 11 (1981) ,638-647

D.A. Lamb, *Construction of a Peephole Optimizer,* Software-Practice & Experience 11 (1981), 638-647

W.M. McKeeman, *Peephole Optimization,* Comm. ACM 8, 7 (July 1965), 443-444