Quellsprachorientiertes Debugging optimierter Programme

Roland Wismüller

20.12.1994

Warum Fehlersuche in optimierten Programmen?

Optimierungen (Codetransformationen) sind essentiell für:

- optimale Nutzung von RISC Architekturen
- einfache und effiziente Programmierung paralleler Architekturen

$\Rightarrow$ Produktionscode sollte optimiert sein!

Aber:

- Auch Produktionscode hat Fehler!
- Fehlersuche in unoptimierter Version nicht adäquat:
  - $\leadsto$ kein Post-mortem-Debugging möglich
  - $\leadsto$ Reproduzieren des Fehlers notwendig:
    sporadische Fehler, durch Optimierung “enttarnte” Fehler,
    Fehler in der Optimierung, (Real-)Zeitbedingungen

Beispiel

```
1   for ( i =0; i<10;  i ++) {
2        y = y + a[i]
3        f  = g + h
4        w = a[i] * 2
5        z = z + w
6   }
```

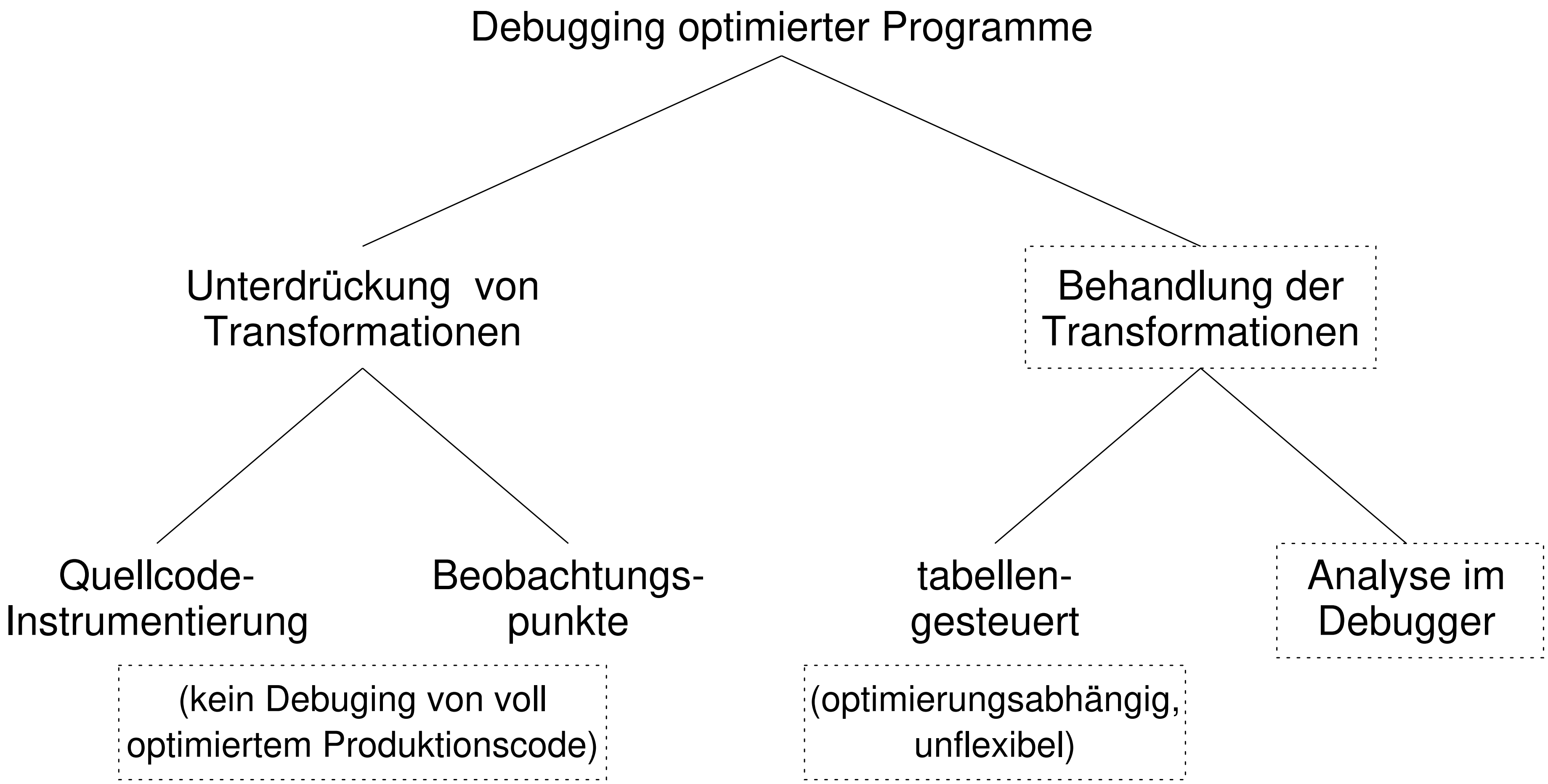
```
1    add    %o0,%i5,  %l6
2    mov    0,  %i3
3    add    %fp,-48,%i2
4    ld     [%i2],%i5
5    sll    %i5,1,  %i1
6    add    %i4,%i1,  %i4
7    inc    %i3
8    cmp    %i3,10
9    add    %l7,%i5,  %l7
10   bl     4
11   inc    4,%i2
```

Herkömmliche Symboltabelle (Ausschnitt):

Zeile	Adr.
3	1

Var.	Reg.
i	%i3
g	%i5

Methoden zum Debugging optimierter Programme



Anforderungen

- Keine fest vorgegebene Haltepunktabbildung
- Ausnutzung von Laufzeitinformation
- Realistischer Einsatzbereich (globale Optimierungen, Zeiger)
- Formales Modell (Korrektheit)

Untersuchte Fragestellungen

- Ermittlung von Variablenwerten
  (ohne bzw. mit Berücksichtigung von Schleifentransformationen)
- Erzielung eines besseren Debuggerverhaltens
  (Laufzeitinformation, Aufzeichnung von Variablenwerten, *recovery*)
- Optimale Abbildung von Haltepunkten
- Umsetzung von Ablaufspuren (*traces*)

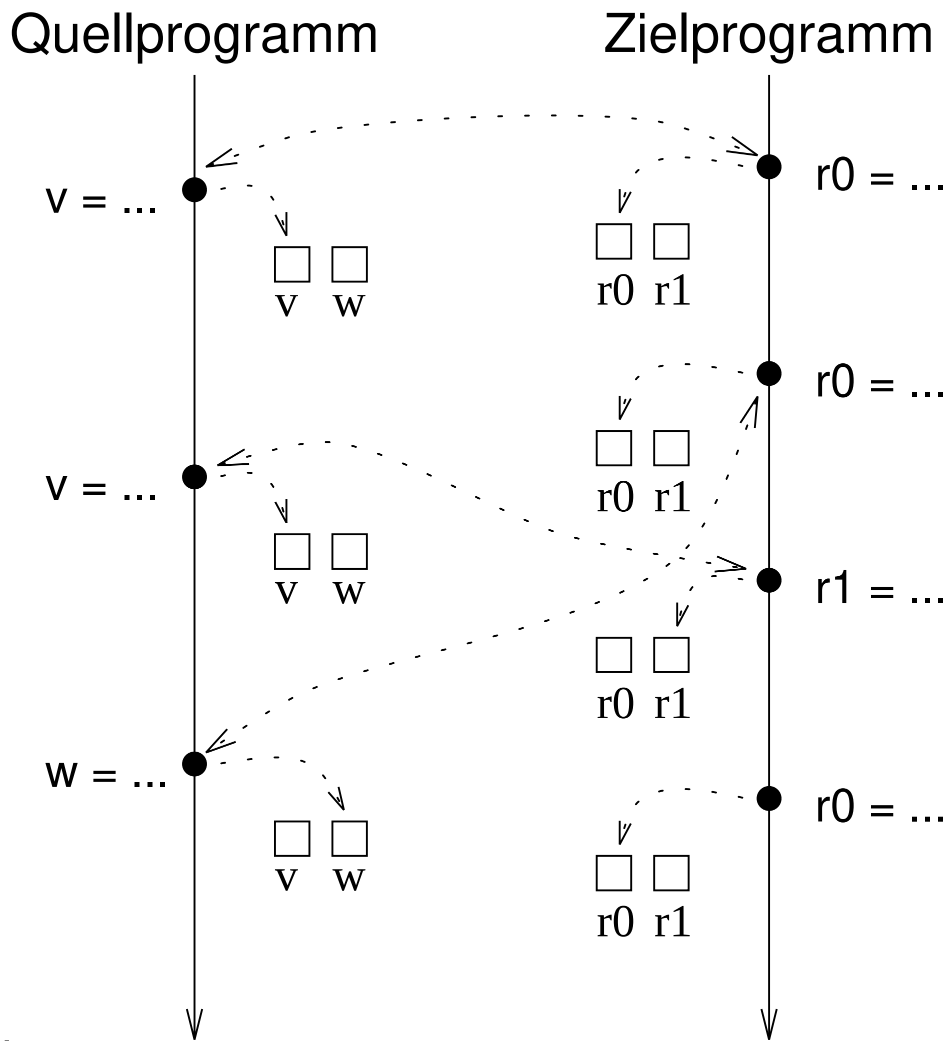
Ermittlung von Variablenwerten

Zwei Teilfragen:

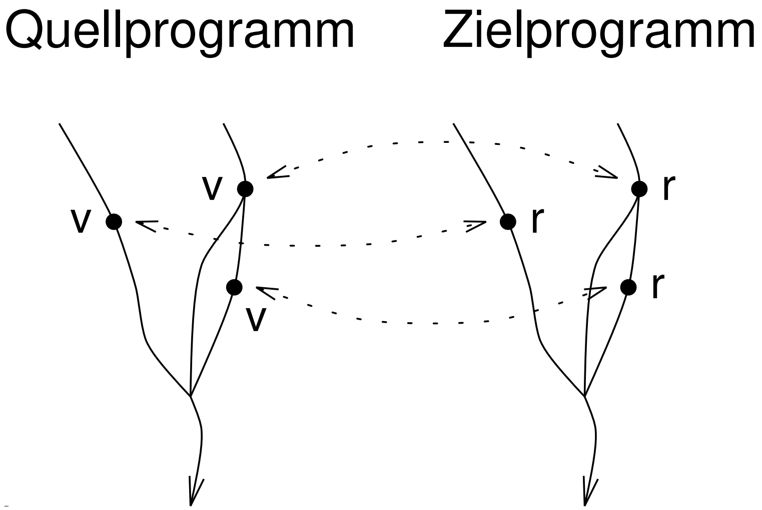
1. Welches Register ist der Variablen am Haltepunkt zugeordnet?

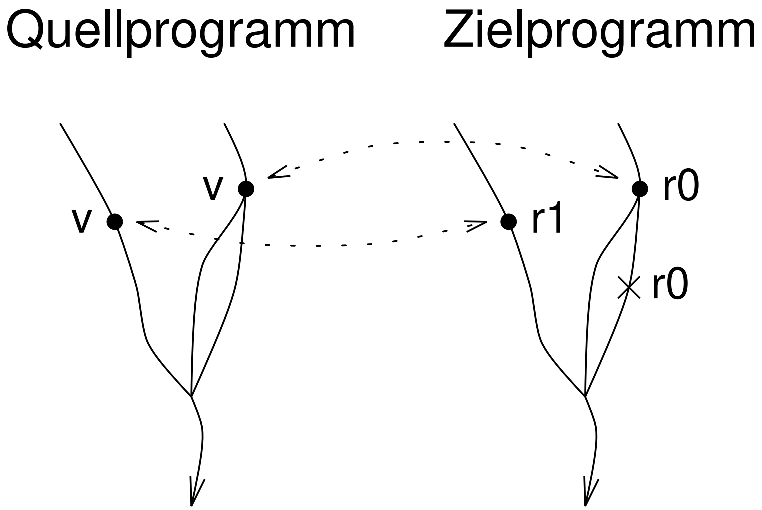
2. Stimmt der Inhalt des Registers mit dem erwarteten Wert der Variablen überein?

Beantwortung durch Analyse von Programmablauf*paaren*



Ermittlung von Variablenwerten





Betrachte alle den Haltepunkt erreichenden Zuweisungen an v:

Alle zugehörigen Instruktionen schreiben auf Register r	$\Rightarrow$	r ist zugehöriges Register zu v
Alle zugehörigen Schreibzugriffe auf r erreichen den Haltepunkt im Zielprogramm entlang aller zulässigen Pfade	$\Rightarrow$	r enthält erwarteten Wert von v

Zu lösende Probleme

- Einschränkung der Analyse auf die zulässigen Pfade
- Unterscheidung von Instanzen derselben Anweisung
- Behandlung indirekter Zuweisungen
- Terminierung des Verfahrens

Implementierung und Ergebnisse

- Erweiterung des SUN C-Compilers für den Sparc-Prozessor

  $\leadsto$ Abbildung kann lokal in jedem Optimierungsschritt aktualisiert werden
- Realisierung der Analyse zur Ermittlung von Variablenwerten:
  integriert in Debugger DETOP, Modul für statistische Untersuchungen

  $\leadsto$ An beliebigem Haltepunkt 70 - 80% aller Variablen korrekt inspizierbar

Zusammenfassung und Ausblick

- Quellsprach-Debugging ist in Verbindung mit üblichen Zwischencode-Optimierungen praktikabel:
  - $\leadsto$ Analyse ist in allgemeiner, compiler-unabhängiger Form möglich
  - $\leadsto$ Unterstützung im Compiler ist sehr einfach
  - $\leadsto$ Resultate weisen darauf hin, daß brauchbares Debuggerverhalten erzielt werden kann
- Zukunft: Praktische Erfahrungen

- Bei Anwendung von Schleifentransformationen ist die Analyse zwar im Prinzip möglich, jedoch in bisheriger Form zu aufwendig
- Zukunft: Realisierbare Analyse für Schleifentransformationen