

### Optimierende Compiler Rückwandlung aus der SSA-Form

#### Andreas Koch

FG Eingebettete Systeme und ihre Anwendungen Informatik, TU Darmstadt

Sommersemester 2011

### OptComp

A. Koch

Lost Copy

Vertauschung

. Platzierung

Briggs-



### Überblick



 Interaktion zwischen SSA-Rückwandlung und Optimierung

- Lost-Copy-Problem
- Swap-Problem
- Unnötige Kopieranweisungen
- Probleme bei Platzierung der Kopieranweisungen
- Lösung: Algorithmus nach Briggs, Cooper, Harvey und Simpson
  - Paper liegt auf Web-Seite!
  - Eine Korrektur wird hier in VL besprochen

OptComp

A. Koch

Lost Conv

Vertauschur

Abiauipianun

latzierung

### Überblick



 Interaktion zwischen SSA-Rückwandlung und Optimierung

- Lost-Copy-Problem
- Swap-Problem
- Unnötige Kopieranweisungen
- Probleme bei Platzierung der Kopieranweisungen
- Lösung: Algorithmus nach Briggs, Cooper, Harvey und Simpson
  - Paper liegt auf Web-Seite!
  - Eine Korrektur wird hier in VL besprochen

OptComp

A. Koch

Lost Conv

Vertauschung

Ablautplanung

- .



# Grundlagen

OptComp

A. Koch

Grundlagen

Vortauschung

Ablaufolanung

Platzierung

Briggs-Maorithmus

### Entfernen von Phi-Knoten

Ersetzen durch Kopieranweisungen





A. Koch

Grundlagen

ost Copy

Vertauschun

waupianu

Duinen

Algorith



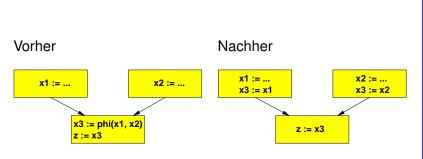
z := x3

### Nachher

### Entfernen von Phi-Knoten

Ersetzen durch Kopieranweisungen





OptComp

A. Koch

Grundlagen

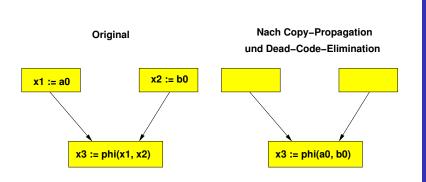
osi Copy

vertauschung

Platziarung

### Optimierung durch Copy Propagation





OptComp

A. Koch

Grundlagen

Vertauschung

latzierung



# Das "Lost-Copy"-Problem

OptComp

A. Koch

Lost Copy

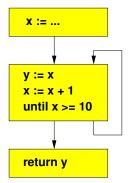
Vertauschung

Ablaufplanun

Platzierun



### Normale Form

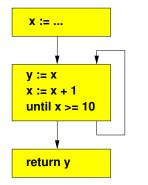


Hierbei keine Überraschungen



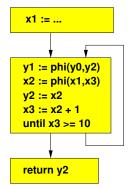


### Normale Form



Hierbei keine Überraschungen

#### SSA-Form



OptComp

A. Koch

Lost Copy

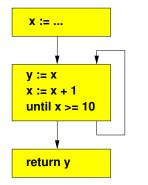
Ablaufplanung

Platzierunç

Briggs-Algorithi

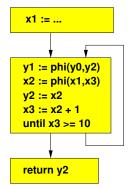


### Normale Form



Hierbei keine Überraschungen

#### SSA-Form



OptComp

A. Koch

Lost Copy

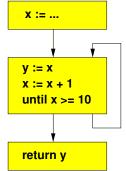
Ablaufplanung

Platzierunç

Briggs-Algorithi

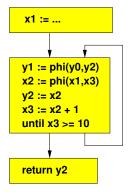


### Normale Form



Hierbei keine Überraschungen

#### SSA-Form



OptComp

A. Koch

Lost Copy

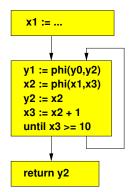
Vertauschung Ablaufplanung

Piatzieruni

Algorithr



#### SSA-Form



• Immer noch keine Überraschungen



A. Koch

Lost Copy

/ertausch

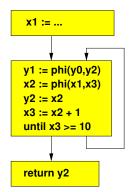
Ablaufplanung

r iaiziei ui i

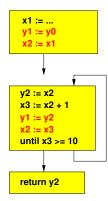
Algorithi







#### Rücktransformation aus SSA-Form



Immer noch keine Überraschungen

OptComp

A. Koci

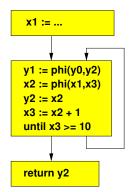
Lost Copy

/ertauschung

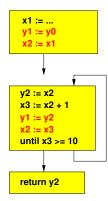
Platzierung







# Rücktransformation aus SSA-Form



Immer noch keine Überraschungen

OptComp

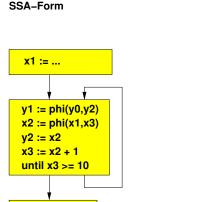
A. Koci

Lost Copy

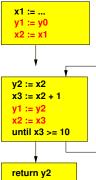
/ertauschung

Platzierung





Rücktransformation aus SSA-Form



Immer noch keine Überraschungen

return y2

OptComp

A. Kocl

Lost Copy

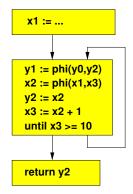
/ertauschung Ablaufplanung

Briggs-

inggs-.lgorithmi



#### SSA-Form



y2:=x2 propagiert



A. Koch

arunulager

Lost Copy

vertauscrium

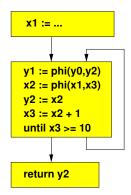
Platziarung

Briggs-

Algorithmu

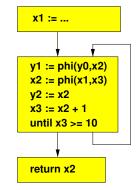


#### SSA-Form



v2:=x2 propagiert

# SSA-Form nach Copy-Propagation



OptComp

A. Koch

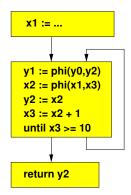
Lost Copy

Vertauschung Ablaufplanung

Platzierung

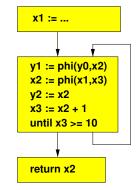


#### SSA-Form



v2:=x2 propagiert

# SSA-Form nach Copy-Propagation



OptComp

A. Koch

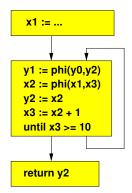
Lost Copy

Vertauschung Ablaufplanung

Platzierung

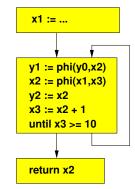


#### SSA-Form



y2:=x2 propagiert

# SSA-Form nach Copy-Propagation



OptComp

A. Koch

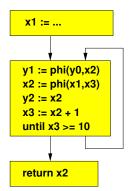
Lost Copy

Ablaufplanung

Platzierung



SSA-Form nach Copy-Propagation



Unnötige Kopieranweisung und Phi-Funktion für y1 entfernt



A. Koch

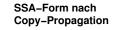
Lost Copy

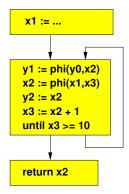
Ablaufolanur

Platzierung

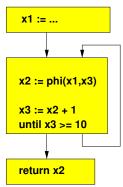
Briggs-Algorithr







SSA-Form nach Copy-Propagation und Dead-Code Elim.



Unnötige Kopieranweisung und Phi-Funktion für y1 entfernt

OptComp

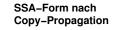
A. Koch

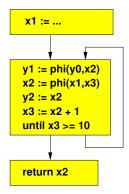
Lost Copy

Vertauschung

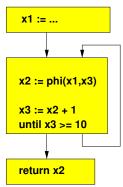
Platzierung







SSA-Form nach Copy-Propagation und Dead-Code Elim.



Unnötige Kopieranweisung und Phi-Funktion für y1 entfernt

OptComp

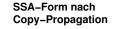
A. Koch

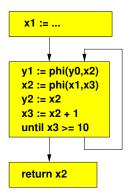
Lost Copy

Vertauschung

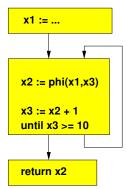
Platzierung







SSA-Form nach Copy-Propagation und Dead-Code Elim.



 Unnötige Kopieranweisung und Phi-Funktion für y1 entfernt OptComp

A. Koch

Lost Copy

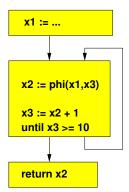
Vertauschung Ablaufplanung

latzierung

Algorithn



SSA-Form nach Copy-Propagation und Dead-Code Elim.



• Fehler: Falscher Wert zurückgegeben



A. Koch

Lost Copy

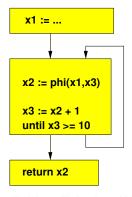
Vertausch

Ablaufplanun

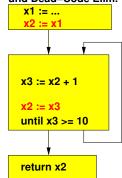
rlatzierung



SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



Fehler: Falscher Wert zurückgegeben



A. Koch

Lost Copy

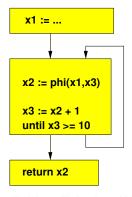
Vertauschung Ablaufplanung

Platzierung
Briggs-

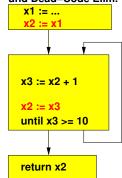




SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



Fehler: Falscher Wert zurückgegeben



A. Koch

Lost Copy

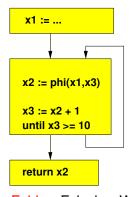
Vertauschung Ablaufplanung

Platzierung
Briggs-

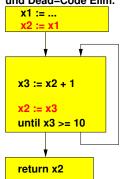




SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



Fehler: Falscher Wert zurückgegeben!

OptComp

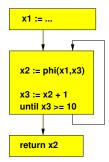
A. Koch

Lost Copy

Vertauschung Ablaufplanung



SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Jetzt richtig
- Aufspalten kritischer Kanten nicht immer möglich oder wünschenswert
- Andere Lösung?



A. Koch

Lost Copy

Vertauschi

A la la colonia la la colonia

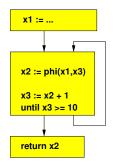
Platzierung



OptComp

Lost Copy

SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.

x1 := ...
x2 := x1

x3 := x2 + 1

until x3 >= 10

return x2

- Jetzt richtig
- Aufspalten kritischer Kanten nicht immer möglich oder wünschenswert
- Andere Lösung?

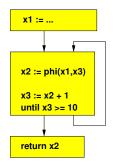




OptComp

Lost Copy

SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.

x1 := ...
x2 := x1

x3 := x2 + 1

until x3 >= 10

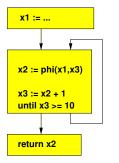
return x2

- Jetzt richtig
- Aufspalten kritischer Kanten nicht immer möglich oder wünschenswert
- Andere Lösung?





SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.

x1 := ...
x2 := x1

x3 := x2 + 1

until x3 >= 10

return x2

- Jetzt richtig
- Aufspalten kritischer Kanten nicht immer möglich oder wünschenswert
- Andere Lösung?



OptComp

A. Koch

Lost Copy

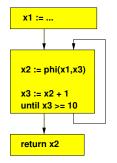
Vertauschun

Ablaufplanung

Platzierung



SSA-Form nach Copy-Propagation und Dead-Code Elim.

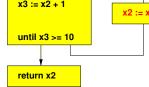


Rücktransformation aus
SSA-Form nach
Copy-Propagation
und Dead-Code Elim.

X1 := ...
X2 := x1

Kritische Kante aufgeteilt

X3 := x2 + 1



- Jetzt richtig
- Aufspalten kritischer Kanten nicht immer möglich oder wünschenswert
- Andere Lösung?



Grundlagen Lost Copy

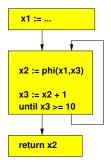
Vertauschung Ablaufplanung

Piatzierung Briaas-

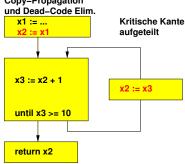
Algorithm



SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim. x1 := ...



OptComp

A Koch

Lost Copy

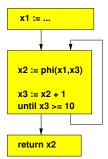
- Jetzt richtig
- Aufspalten kritischer Kanten nicht immer möglich oder wünschenswert
- Andere Lösung?



### Lösung: Wert vor Überschreiben sichern



SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Erkenne solche Fälle (→ Live Variables)
- Füge Sicherheitskopie ein
- Ersetze spätere Verwendungen durch Sicherheitskopie



A. Koch

Lost Copy

vertauschur

Platzierung

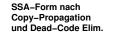
Briggs-Algorithn

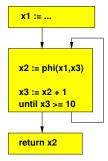
### Lösung: Wert vor Überschreiben sichern



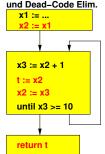
OptComp

A Koch





Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Erkenne solche Fälle (→ Live Variables)
- Füge Sicherheitskopie ein
- Ersetze spätere Verwendungen durch Sicherheitskopie

Grundlagen

Lost Copy

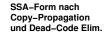
Vertauschung

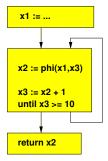
Platzierung

Briggs-Algorithr

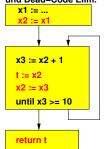
#### Lösung: Wert vor Überschreiben sichern







Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Erkenne solche Fälle (→ Live Variables)
- Füge Sicherheitskopie ein
- Ersetze spätere Verwendungen durch Sicherheitskopie

OptComp

A. Koch

Lost Copy

Vertauschung

. Platzierung

Briggs-Algorithr

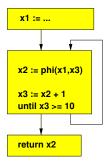
#### Lösung: Wert vor Überschreiben sichern



OptComp

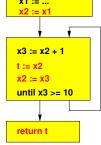
A Koch

SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.

x1 := ...
x2 := x1



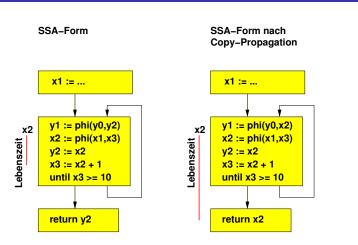
- Erkenne solche Fälle (→ Live Variables)
- Füge Sicherheitskopie ein
- Ersetze spätere Verwendungen durch Sicherheitskopie

Lost Copy
Vertauschung
Ablaufplanun

Flatzierung Briggs-Algorithmu:

## Allgemeines Lost-Copy-Problem 1





OptComp

A. Koch

Lost Copy

Vertauschur

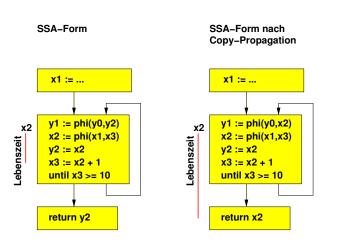
Platzierung

Briggs-Jgorithmus

Lebenszeit von Variablen über bestimmende Phi-Funktion hinaus ausgedehnt

## Allgemeines Lost-Copy-Problem 1





OptComp

A. Koci

Lost Copy

Vertauschung Ablaufplanung

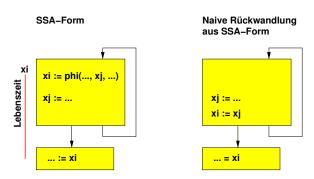
latzierung

Briggs-Algorithm

Lebenszeit von Variablen über bestimmende Phi-Funktion hinaus ausgedehnt

## Allgemeines Lost Copy-Problem 2





A. Koch
Grundlagen
Lost Copy

OptComp

Vertauschung
Ablaufplanung

Platzierung

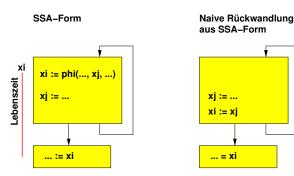
Briggs-Algorithn

- Naive Umwandlung liefert fehlerhafte Ergebnisse
- Kopieranweisung auf xi mitten in dessen Lebenszeit hineingesetzt
- Liefert immer xj statt xi



## Allgemeines Lost Copy-Problem 2





OptComp A. Koch

Lost Copy

Vertauschung Ablaufplanung

Platzierung

Briggs-Algorithn

- Naive Umwandlung liefert fehlerhafte Ergebnisse
- Kopieranweisung auf xi mitten in dessen Lebenszeit hineingesetzt
- Liefert immer xj statt xi



### Lösung für "Lost Copy"-Problem



- Für genau welche Variablen müssen Sicherheitskopien erstellt werden?
- Wo müssen die jeweiligen Sicherheitskopien angelegt werden?

OptComp

A Koch

A. Koch

Lost Copy

Vertauschung

Platzierung



# Fehlerhafte Vertauschung

OptComp

A. Koch

arundlagen Lost Copy

Vertauschung

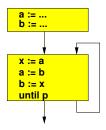
Ablaufplanun

Briggs-



OptComp

#### **Normale Form**



Hierbei keine Überraschungen



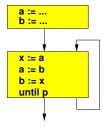
Vertauschung Ablaufplanung

Briggs-

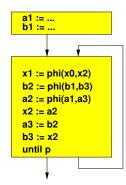




#### **Normale Form**



#### SSA-Form



Hierbei keine Überraschungen

OptComp

A. Koch

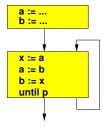
Lost Copy

Vertauschung

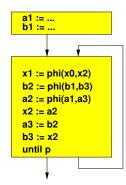
Platzierung



#### **Normale Form**



#### SSA-Form



Hierbei keine Überraschungen

OptComp

A. Koch

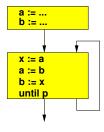
Lost Copy

Vertauschung

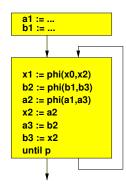
Platzierung



#### **Normale Form**



#### SSA-Form



Hierbei keine Überraschungen

OptComp

A. Koch

rundlagen

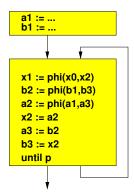
Vertauschung

Ablaufplanur

Briggs-



#### SSA-Form



Immer noch keine Überraschungen



A. Koch

ındlager

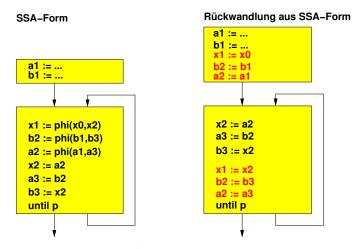
ost Copy

Vertauschung

Ablaufplanur

Prigas





OptComp

A. Koch

Loet Copy

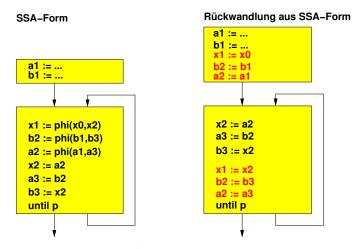
Vertauschung

Platzierung

Briggs-Algorithmus

Immer noch keine Uberraschungen





OptComp

A. Koch

Loet Copy

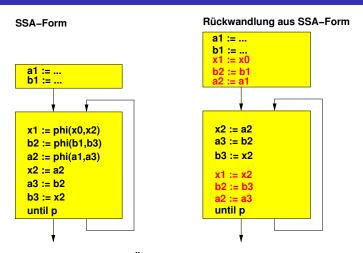
Vertauschung

Platzierung

Briggs-Algorithmus

Immer noch keine Uberraschungen





OptComp

A. Koch

arundlager

Vertauschung

Ablaufplanung

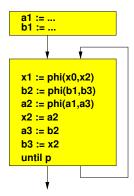
Platzierung

Briggs-Algorithmus

Immer noch keine Überraschungen



#### SSA-Form



Propagiert: x2:=a2, a3:=b2, b3:=x22



A. Koch

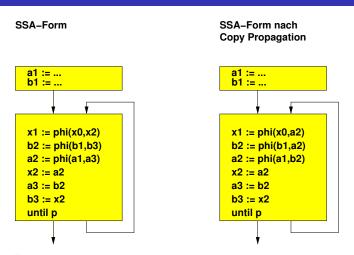
undlagen

ost Copy

Vertauschung

Platzierung





OptComp

A. Koch

Lost Copy

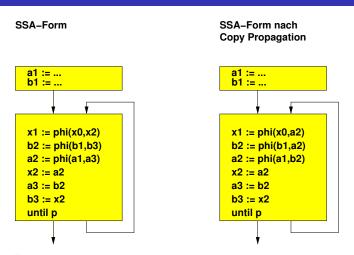
Vertauschung

Platzierung

Briggs-Algorithmus

Propagiert: x2:=a2, a3:=b2, b3:=x2





OptComp

A. Koch

Lost Copy

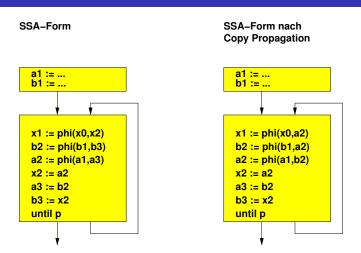
Vertauschung

Platzierung

Briggs-Algorithmus

Propagiert: x2:=a2, a3:=b2, b3:=x2





OptComp

A. Koch

Grundlagen
Lost Copy

Vertauschung

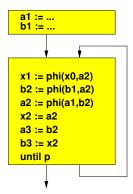
Platzierung

Briggs-Algorithmus

• Propagiert: x2:=a2, a3:=b2, b3:=x2



#### SSA-Form nach Copy Propagation



 Unnotige Kopieranweisungen und Phi-Funktion für x1 entfernt OptComp

A. Koch

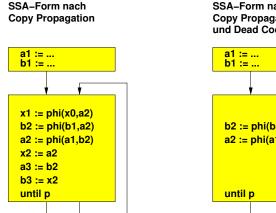
ost Copy

Vertauschung

lblaufplanun

Briggs-





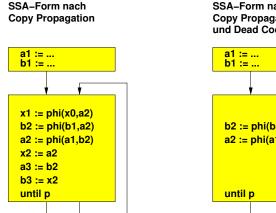
SSA-Form nach Copy Propagation und Dead Code-Elim.

b2 := phi(b1,a2)a2 := phi(a1,b2)

OptComp

Vertauschung





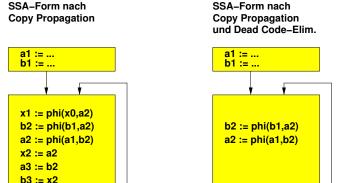
SSA-Form nach Copy Propagation und Dead Code-Elim.

b2 := phi(b1,a2)a2 := phi(a1,b2)

OptComp

Vertauschung





OptComp A Koch

Lost Copy

Vertauschung Ablaufplanung

Briggs-

Unnötige Kopieranweisungen und Phi-Funktion für x1 entfernt

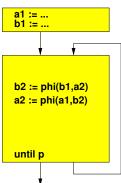
until p



until p



SSA-Form nach Copy Propagation und Dead Code-Elim.



- until p

  Fehler: Keine Vertauschung r
- Formal werden alle Phi-Funktionen parallel ausgeführt
- Kopieranweisungen aber sequentiell



A. Koch

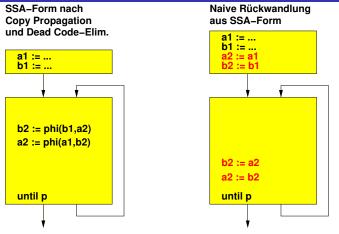
ost Copy

Vertauschung

Ablaufplanun

riatzierung





OptComp

A. KOCI

and Canu

Vertauschung

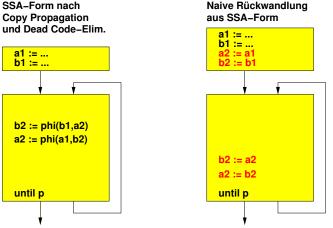
Ablaufplanung

Platzierung

- Fehler: Keine Vertauschung mehr
- Formal werden alle Phi-Funktionen parallel ausgeführt
- Kopieranweisungen aber seguentiell







OptComp

A. Kocr

rundlager

Lost Copy

Vertauschung

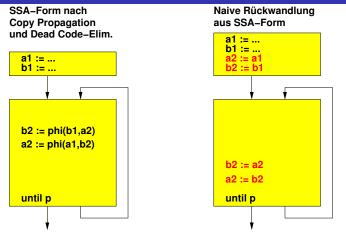
- Diaurpianur

Briggs-

- Fehler: Keine Vertauschung mehr
- Formal werden alle Phi-Funktionen parallel ausgeführt
- Kopieranweisungen aber seguentiell







OptComp

A. Kocl

arunulayen

Vertauschung

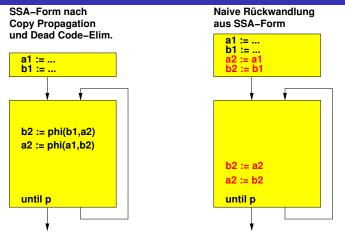
Ablaufplanung

Platzierung

- Fehler: Keine Vertauschung mehr
- Formal werden alle Phi-Funktionen parallel ausgeführt
- Kopieranweisungen aber sequentiell







OptComp

A. Kocl

Lost Conv

Vertauschung

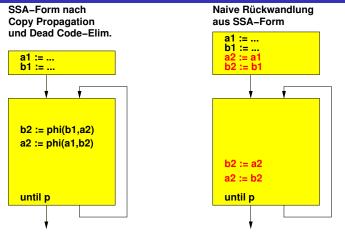
Ablaufplanung

Platzierung

- Fehler: Keine Vertauschung mehr
- Formal werden alle Phi-Funktionen parallel ausgeführt
- Kopieranweisungen aber sequentiell







OptComp

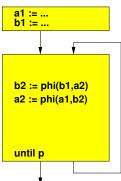
Vertauschung

- Fehler: Keine Vertauschung mehr
- Formal werden alle Phi-Funktionen parallel ausgeführt
- Kopieranweisungen aber sequentiell





SSA-Form nach Copy Propagation und Dead Code-Elim.



- Korrektes Vorgehen: Zyklus aufbrechen
- Einen der benötigten Werte kopieren, dann Kopie benutzen



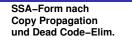
A. Koch

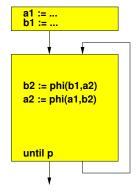
Lost Copy

Vertauschung

Abiauipianui







#### definiert-über



### definiert-über

- Kopieranweisungen voneinander abhängig
- Welche zuerst ausführen?
- Korrektes Vorgehen: Zyklus aufbrechen
- Einen der benötigten Werte kopieren, dann Kopie benutzen



A. Kod

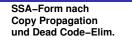
undlagen

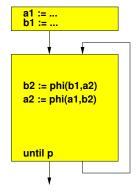
Vertauschung

Platzierung









#### definiert-über



### definiert-über

- Kopieranweisungen voneinander abhängig
- Welche zuerst ausführen?
- Korrektes Vorgehen: Zyklus aufbrechen
- Einen der benötigten Werte kopieren, dann Kopie benutzen



A. Kod

undlagen

Vertauschung

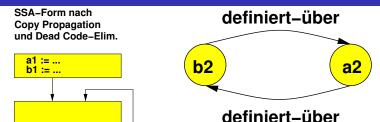
Platzierung



b2 := phi(b1,a2) a2 := phi(a1,b2)

until p





- Kopieranweisungen voneinander abhängig
- Welche zuerst ausführen?
- Korrektes Vorgehen: Zyklus aufbrechen
- Einen der benötigten Werte kopieren, dann Kopie benutzen



A. Koch

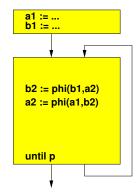
undlagen st Copy

Vertauschung

Platzierung



SSA-Form nach Copy Propagation und Dead Code-Elim.



definiert-über



#### definiert-über

- Kopieranweisungen voneinander abhängig
- Welche zuerst ausführen?
- Korrektes Vorgehen: Zyklus aufbrechen
- Einen der benötigten Werte kopieren, dann Kopie benutzen



A. Koch

undlagen

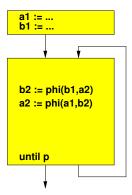
Vertauschung Ablaufplanung

Platzierung

### Korrekte SSA-Rückwandlung nach CFG



SSA-Form nach Copy Propagation und Dead Code-Elim.



Rechnet nun korrekt



A. Koch

Lost Copy

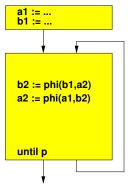
Vertauschung

Platzierung

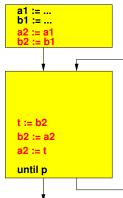
# Korrekte SSA-Rückwandlung nach CFG



SSA-Form nach Copy Propagation und Dead Code-Elim.



Korrekte Rückwandlung aus SSA-Form



Rechnet nun korrek

OptComp

A. Kocl

Loet Conv

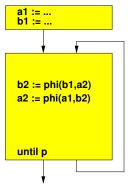
Vertauschung

Platzierung

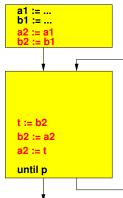
# Korrekte SSA-Rückwandlung nach CFG



SSA-Form nach Copy Propagation und Dead Code-Elim.



Korrekte Rückwandlung aus SSA-Form



Rechnet nun korrek

OptComp

A. Kocl

Loet Conv

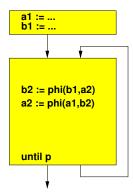
Vertauschung

Platzierung

# Korrekte SSA-Rückwandlung nach CFG

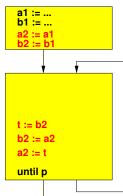


SSA-Form nach Copy Propagation und Dead Code-Elim.



Rechnet nun korrekt

# Korrekte Rückwandlung aus SSA-Form



OptComp

A. KOCI

Lost Conv

Vertauschung

Platzierung



Abhängigkeitszyklen durch Einführen einer block-lokalen Kopie eines Phi-Funktion-Parameters aufbrechen.

- Unterschied zur Lösung des "Lost Copy"-Problems
- Dort block-übergreifende Sicherheitskopie von Phi-Funktions-Ergebnis

Sind bei allen Arten von Abhängigkeiten zwischen Phi-Funktionen block-lokale Kopien erforderlich?

OptComp

A. Koch

rundlagen ost Copy

Vertauschung
Ablaufplanung

Platzierunç



Abhängigkeitszyklen durch Einführen einer block-lokalen Kopie eines Phi-Funktion-Parameters aufbrechen.

- Unterschied zur Lösung des "Lost Copy"-Problems
- Dort block-übergreifende Sicherheitskopie von Phi-Funktions-Ergebnis

Sind bei allen Arten von Abhängigkeiten zwischen Phi-Funktionen block-lokale Kopien erforderlich?

OptComp

A. Koch

ost Copy

Vertauschung

Platzierung



Abhängigkeitszyklen durch Einführen einer block-lokalen Kopie eines Phi-Funktion-Parameters aufbrechen.

- Unterschied zur Lösung des "Lost Copy"-Problems
- Dort block-übergreifende Sicherheitskopie von Phi-Funktions-Ergebnis

Sind bei allen Arten von Abhängigkeiten zwischen Phi-Funktionen block-lokale Kopien erforderlich?

OptComp

A. Koch

ost Copy

Vertauschung

Platzierung



Abhängigkeitszyklen durch Einführen einer block-lokalen Kopie eines Phi-Funktion-Parameters aufbrechen.

- Unterschied zur Lösung des "Lost Copy"-Problems
- Dort block-übergreifende Sicherheitskopie von Phi-Funktions-Ergebnis

Sind bei allen Arten von Abhängigkeiten zwischen Phi-Funktionen block-lokale Kopien erforderlich?

OptComp

A. Koch

ost Copy

Vertauschung

Platzierung



Abhängigkeitszyklen durch Einführen einer block-lokalen Kopie eines Phi-Funktion-Parameters aufbrechen.

- Unterschied zur Lösung des "Lost Copy"-Problems
- Dort block-übergreifende Sicherheitskopie von Phi-Funktions-Ergebnis

Sind bei allen Arten von Abhängigkeiten zwischen Phi-Funktionen block-lokale Kopien erforderlich?

OptComp

A. Koch

ost Copy

Vertauschung Ablaufplanung

Platzierung



# Ablaufplanung statt Kopieren

OptComp

A. Koch

Lost Copy

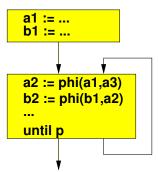
Vertauschung

Ablaufplanung

Priggs



### SSA-Form



- b2 ist von a2 abhängig ightarrow a2 lokal kopieren
- Rechnet korrekt
- Kopie ist aber unnötiger Aufwand!



A. Koch

ost Copy

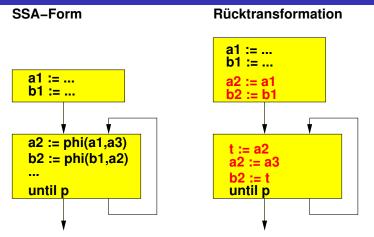
Vertauschu

Ablaufplanung

Platzierun

Briggs-Algorithn





OptComp

A. Koch

undlager

Lost Copy

Vertauschung

Ablaufplanung

n.:...

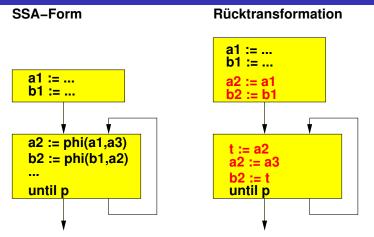
riggs-Ilgorithn

lacksquare b2 ist von a2 abhängig ightarrow a2 lokal kopieren

- Rechnet korrekt
- Kopie ist aber unnötiger Aufwand!







OptComp

A. Koch

undlager

Lost Copy

Vertauschung

Ablaufplanung

n.:...

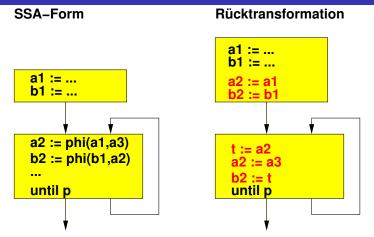
riggs-Ilgorithn

lacksquare b2 ist von a2 abhängig ightarrow a2 lokal kopieren

- Rechnet korrekt
- Kopie ist aber unnötiger Aufwand!







- OptComp
  - A. Koch
- Grundlagen Lost Copy
- Vertauschung
- Ablaufplanung
- Briaas-
- ariggs-Algorithr

- b2 ist von a2 abhängig → a2 lokal kopieren
- Rechnet korrekt
- Kopie ist aber unnötiger Aufwand!





OptComp

# SSA-Form Rücktransformation a2 := a1 b2 := b1a2 := phi(a1,a3)t := a2b2 := phi(b1,a2)a2 := a3b2 := tuntil p until p

- Lost Copy
  Vertauschung
  Ablaufplanung
  - riggs-
- anggs-Algorithr

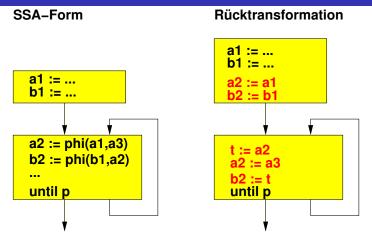
- b2 ist von a2 abhängig → a2 lokal kopieren
- Rechnet korrekt
- Kopie ist aber unnötiger Aufwand!





OptComp

A Koch



Vertauschung

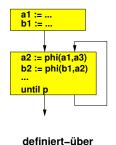
Ablaufplanung

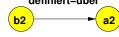
Platzierung

- b2 ist von a2 abhängig → a2 lokal kopieren
- Rechnet korrekt
- Kopie ist aber unnötiger Aufwand!



#### SSA-Form





- Keine Kopie nötig, da keine zirkuläre Abhängigkei
- Geschickte Abiaurpianung der Kopien
- Topologische Reihenfolge im Graphen



A. Koch

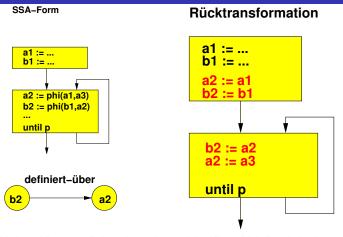
Lost Copy

Vertauschi

Ablaufplanung

Platzierung



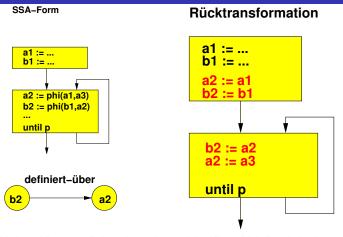


OptComp

Ablaufplanung





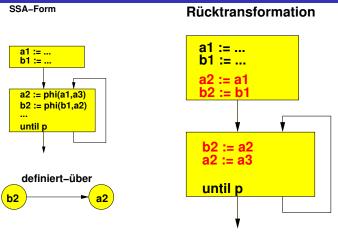


OptComp

Ablaufplanung







OptComp

A. Koch

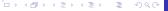
Lost Copy

Vertauschung

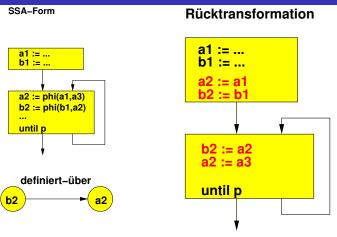
Ablaufplanung
Platzierung

riggs-

- Keine Kopie nötig, da keine zirkuläre Abhängigkeit
- Geschickte Ablaufplanung der Kopien
- Topologische Reihenfolge im Graphen







OptComp

A. Koch

Lost Copy

Vertauschung Ablaufplanung

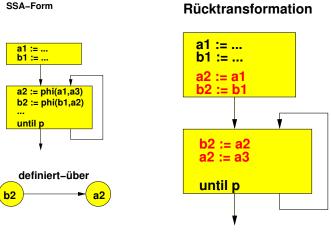
Platzierung

iriggs-.lgorithn

- Keine Kopie nötig, da keine zirkuläre Abhängigkeit
- Geschickte Ablaufplanung der Kopien
- Topologische Reihenfolge im Graphen







OptComp

A. Koch

Grundlagen Lost Copy

Vertauschung
Ablaufplanung

Platzierur

iriggs-.lgorithn

- Keine Kopie nötig, da keine zirkuläre Abhängigkeit
- Geschickte Ablaufplanung der Kopien
- Topologische Reihenfolge im Graphen





# Platzierung der Phi-Funktionen

OptComp

A. Koch

ost Copy

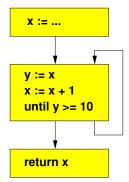
Vertauschung

N-1-i----

Platzierung



#### **Normale Form**



Keine Uberraschungen



A. Koch

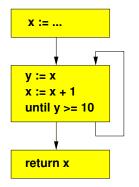
ost Copy

Vertauschun

Platzierung

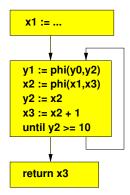


### Normale Form



Keine Überraschungen

#### SSA-Form



OptComp

A. Koch

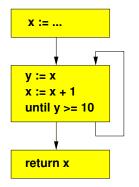
Grundlagen Lost Copy

Vertauschung

Platzierung

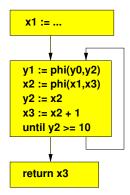


### Normale Form



Keine Überraschungen

#### SSA-Form



OptComp

A. Koch

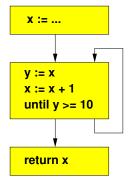
Grundlagen Lost Copy

Vertauschung

Platzierung

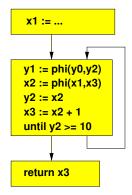


#### **Normale Form**



Keine Überraschungen

#### SSA-Form



OptComp

A. Koch

rundlagen est Copy

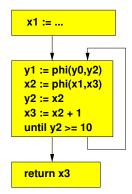
/ertauschung

Platzierung

Briggs-Algorithr



#### SSA-Form



Keine Überraschungen

OptComp

A. Koch

undlagen

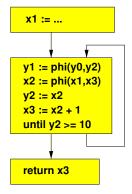
ost Copy

Ablaufalaau

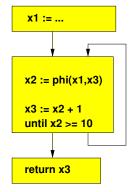
Platzierung







SSA-Form nach Copy-Propagation und Dead-Code Elim.



OptComp

A. Koch

undlagen

Vertauschung

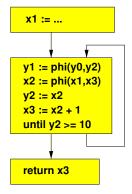
Ablaufplanur Platzierung

Briggs-

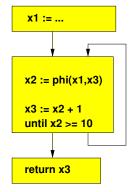
Keine Überraschungen







SSA-Form nach Copy-Propagation und Dead-Code Elim.



OptComp

A. Koch

undlagen

Vertauschung

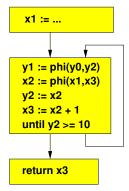
Ablaufplanur Platzierung

Briggs-

Keine Überraschungen

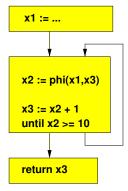






Keine Überraschungen

SSA-Form nach Copy-Propagation und Dead-Code Elim.



OptComp

A. Koch

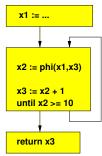
irundiagen ost Copy

Vertauschung

Platzierung



SSA-Form nach
Copy-Propagation
und Dead-Code Elim.



- Fehler: Wert für nächste Iteration
- ... wird schon am Ende der aktuellen Iteration verwendet
- a Dhi Kaniaranyyaisyna in Labanasai
- Phi-Kopieranweisung in



A. Koch

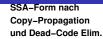
ost Copy

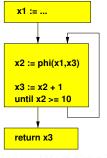
Vertauschu

Abiauipianun

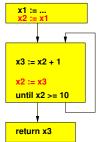
Platzierung







Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Fehler: Wert für nächste Iteratior
- ... wird schon am Ende der aktuellen Iteration verwendet
- e Phi Kanjaranyajauri
- Phi-Kopieranweisung in Leber

OptComp

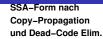
A. Koch

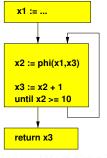
Lost Copy

Vertauschung

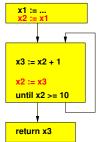
Platzierung







Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Fehler: Wert für nächste Iteratior
- ... wird schon am Ende der aktuellen Iteration verwendet
- e Phi Kanjaranyajauri
- Phi-Kopieranweisung in Leber

OptComp

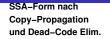
A. Koch

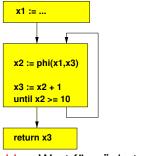
Lost Copy

Vertauschung

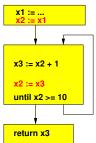
Platzierung







Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Fehler: Wert f
   ür n
   ächste Iteration.
  - ... aus aufgelöster Phi-Funktion
- ... wird schon am Ende der aktuellen Iteration.
- Phi-Kopieranweisung in Lebenszeit von x2-eingefügt!

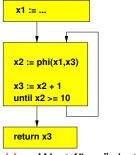
OptComp

A Koch

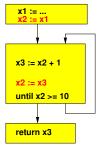
Platzierung



SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Fehler: Wert für nächste Iteration
  - ... aus aufgelöster Phi-Funktion
- ... wird schon am Ende der aktuellen Iteration verwendet
  - ... in bedingtem Sprung am Block-Ende
- Phi-Kopieranweisung in Lebenszeit, von, x2\_eingefügt! nac

OptComp

A. Koch

rundlagen

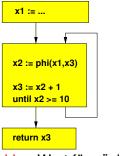
Vertauschung

Platzierung

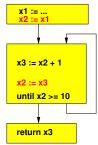
Platzierung



SSA-Form nach
Copy-Propagation
und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Fehler: Wert für nächste Iteration
  - ... aus aufgelöster Phi-Funktion
- ... wird schon am Ende der aktuellen Iteration verwendet
  - ... in bedingtem Sprung am Block-Ende
- Phi-Kopieranweisung in Lebenszeit, von, x2\_eingefügt! noch

OptComp

A. Koch

rundlager

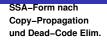
Vertauschung

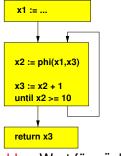
Districums

Platzierung

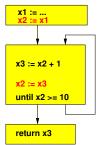
#### Rückwandlung aus SSA-Form







Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Fehler: Wert für nächste Iteration
  - ... aus aufgelöster Phi-Funktion
- ... wird schon am Ende der aktuellen Iteration verwendet
  - ... in bedingtem Sprung am Block-Ende
- Phi-Kopieranweisung in Lebenszeit, von x2-eingefügt!

OptComp

A. Koch

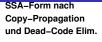
Grundlagen Lost Copy

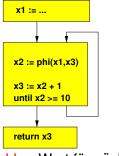
Vertauschung

Platzierung

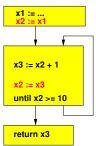
#### Rückwandlung aus SSA-Form







Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Fehler: Wert für nächste Iteration
  - ... aus aufgelöster Phi-Funktion
- ... wird schon am Ende der aktuellen Iteration verwendet
  - ... in bedingtem Sprung am Block-Ende
- Phi-Kopieranweisung in Lebenszeit von x2 eingefügt!



A. Koch

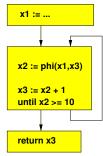
Grundlagen ₋ost Copy

Vertauschung Ablaufplanung

Platzierung



SSA-Form nach
Copy-Propagation
und Dead-Code Elim.



Gleiches Vorgehen wie bei Lost-Copy

Dafür auch bereits vorher angelegte Konie benutzbag



A. Koch

Grundlager

Vertauschu

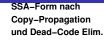
Ablaufplanu

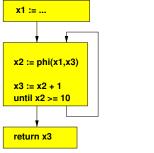
Platzierung



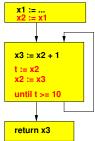
OptComp

Platzierung





Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



Gleiches Vorgehen wie bei Lost-Copy

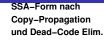
Dafür auch bereits vorher angelegte Kopie benutzba

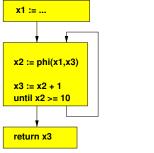




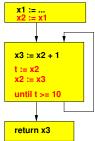
OptComp

Platzierung





Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



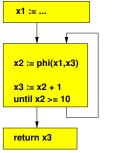
Gleiches Vorgehen wie bei Lost-Copy

Dafür auch bereits vorher angelegte Kopie benutzba

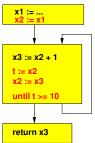




SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Gleiches Vorgehen wie bei Lost-Copy
  - Sicherheitskopie von aktuellem Wert vor Überschreiben anlegen
  - Dann spätere Verwendungen durch Kopie austauschen
- Dafür auch bereits vorher angelegte Kopie benutzbar



A. Koch

rundlager

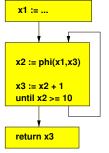
Lost Copy

vortadooridrig

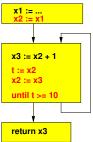
Platzierung



SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Gleiches Vorgehen wie bei Lost-Copy
  - Sicherheitskopie von aktuellem Wert vor Überschreiben anlegen
  - Dann spätere Verwendungen durch Kopie austauschen
- Dafür auch bereits vorher angelegte Kopie benutzbar
   Eventuell bereits wegen Inter-Black Liveness angelege

OptComp

A. Koch

rundlagen

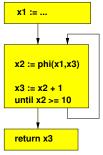
Vertauschun

Ablaufplanun

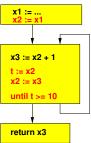
Platzierung



SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Gleiches Vorgehen wie bei Lost-Copy
  - Sicherheitskopie von aktuellem Wert vor Überschreiben anlegen
  - Dann spätere Verwendungen durch Kopie austauschen
- Dafür auch bereits vorher angelegte Kopie benutzbar

OptComp

A. Koch

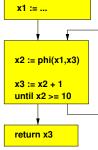
Grundlagen

Vertauschung

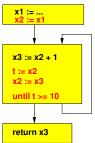
Platzierung



SSA-Form nach Copy-Propagation und Dead-Code Elim.



Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Gleiches Vorgehen wie bei Lost-Copy
  - Sicherheitskopie von aktuellem Wert vor Überschreiben anlegen
  - Dann spätere Verwendungen durch Kopie austauschen
- Dafür auch bereits vorher angelegte Kopie benutzbar
  - Eventuell bereits wegen Inter-Block Liveness angelegt new part of the second sec

OptComp

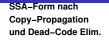
A. Koch

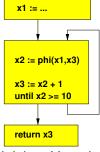
arundlagen ₋ost Copy

Vertauschung

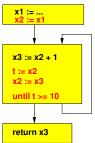
Platzierung







Rücktransformation aus SSA-Form nach Copy-Propagation und Dead-Code Elim.



- Gleiches Vorgehen wie bei Lost-Copy
  - Sicherheitskopie von aktuellem Wert vor Überschreiben anlegen
  - Dann spätere Verwendungen durch Kopie austauschen
- Dafür auch bereits vorher angelegte Kopie benutzbar
  - Eventuell bereits wegen Inter-Block Liveness angelegt

OptComp

A. Koch

irundlagen ost Copy

Vertauschung

Platzierung



- Füge Inter-Block Kopien ein, wenn Kopieranweisungen von aufgelösten Phi-Funktionen in Lebenszeit von Phi-Ergebnis liegen
- Füge lokale Kopien von Phi-Parametern ein, um zyklische Abhängigkeiten aufzubrechen
- Ordne sonstige Kopierfunktionen in richtiger Reihenfolge an
  - Sequentielle Abarbeitung muss gleiches Ergebnis wie parallele Phi-Funktionen ergeben
- Falls Ziel einer Phi-Funktion in bedingten Sprüngen am Block-Ende benutzt wird
  - Verwende dort eine eventuell bereits angelegte Inter-Block-Kopie des Ergebnisses
  - Oder lege neue lokale Kopie an und verwende diese

OptComp

A. Koch

arundiagen ₋ost Copv

Vertauschung

Platzierung



- Füge Inter-Block Kopien ein, wenn Kopieranweisungen von aufgelösten Phi-Funktionen in Lebenszeit von Phi-Ergebnis liegen
- Füge lokale Kopien von Phi-Parametern ein, um zyklische Abhängigkeiten aufzubrechen
- Ordne sonstige Kopierfunktionen in richtiger Reihenfolge an
  - parallele Phi-Funktionen ergeben
- Falls Ziel einer Phi-Funktion in bedingten Sprüngen am Block-Ende benutzt wird
  - Verwende dort eine eventuell bereits angelegte Inter-Block-Kopie des Ergebnisses
     Oder lege neue lokale Kopie an und verwende des

OptComp

A. Koch

Grundlagen Lost Copy

Vertauschung

Platzierung



- Füge Inter-Block Kopien ein, wenn Kopieranweisungen von aufgelösten Phi-Funktionen in Lebenszeit von Phi-Ergebnis liegen
- Füge lokale Kopien von Phi-Parametern ein, um zyklische Abhängigkeiten aufzubrechen
- Ordne sonstige Kopierfunktionen in richtiger Reihenfolge an
  - Sequentielle Abarbeitung muss gleiches Ergebnis wie parallele Phi-Funktionen ergeben
- Falls Ziel einer Phi-Funktion in bedingten Sprüngen am Block-Ende benutzt wird

Verwende dort eine eventuell bereits angelegte
 Inter-Block-Kopie des Ergebnisses
 Oder lege neue lokale Kopie an und verwende die

OptComp

A. Koch

Grundlagen Lost Copv

Vertauschung

Platzierung



- Füge Inter-Block Kopien ein, wenn Kopieranweisungen von aufgelösten Phi-Funktionen in Lebenszeit von Phi-Ergebnis liegen
- Füge lokale Kopien von Phi-Parametern ein, um zyklische Abhängigkeiten aufzubrechen
- Ordne sonstige Kopierfunktionen in richtiger Reihenfolge an
  - Sequentielle Abarbeitung muss gleiches Ergebnis wie parallele Phi-Funktionen ergeben
- Falls Ziel einer Phi-Funktion in bedingten Sprüngen am Block-Ende benutzt wird

OptComp

A. Koch

Grundlagen ₋ost Copy

Vertauschung Ablaufplanung

Platzierung



- Füge Inter-Block Kopien ein, wenn Kopieranweisungen von aufgelösten Phi-Funktionen in Lebenszeit von Phi-Ergebnis liegen
- Füge lokale Kopien von Phi-Parametern ein, um zyklische Abhängigkeiten aufzubrechen
- Ordne sonstige Kopierfunktionen in richtiger Reihenfolge an
  - Sequentielle Abarbeitung muss gleiches Ergebnis wie parallele Phi-Funktionen ergeben
- Falls Ziel einer Phi-Funktion in bedingten Sprüngen am Block-Ende benutzt wird
  - Verwende dort eine eventuell bereits angelegte Inter-Block-Kopie des Ergebnisses
  - Oder lege neue lokale Kopie an und verwende diese

OptComp

A. Koch

arundlagen ost Copy

Vertauschung Ablaufplanung

Platzierung



- Füge Inter-Block Kopien ein, wenn Kopieranweisungen von aufgelösten Phi-Funktionen in Lebenszeit von Phi-Ergebnis liegen
- Füge lokale Kopien von Phi-Parametern ein, um zyklische Abhängigkeiten aufzubrechen
- Ordne sonstige Kopierfunktionen in richtiger Reihenfolge an
  - Sequentielle Abarbeitung muss gleiches Ergebnis wie parallele Phi-Funktionen ergeben
- Falls Ziel einer Phi-Funktion in bedingten Sprüngen am Block-Ende benutzt wird
  - Verwende dort eine eventuell bereits angelegte Inter-Block-Kopie des Ergebnisses
  - Oder lege neue lokale Kopie an und verwende diese

OptComp

A. Koch

arundlagen ₋ost Copy

Vertauschung Ablaufplanung

Platzierung



- Füge Inter-Block Kopien ein, wenn Kopieranweisungen von aufgelösten Phi-Funktionen in Lebenszeit von Phi-Ergebnis liegen
- Füge lokale Kopien von Phi-Parametern ein, um zyklische Abhängigkeiten aufzubrechen
- Ordne sonstige Kopierfunktionen in richtiger Reihenfolge an
  - Sequentielle Abarbeitung muss gleiches Ergebnis wie parallele Phi-Funktionen ergeben
- Falls Ziel einer Phi-Funktion in bedingten Sprüngen am Block-Ende benutzt wird
  - Verwende dort eine eventuell bereits angelegte Inter-Block-Kopie des Ergebnisses
  - Oder lege neue lokale Kopie an und verwende diese

OptComp

A. Koch

Grundlager Lost Copy

Vertauschung
Ablaufplanung

Platzierung



# Algorithmus nach Briggs, Cooper, Harvey und Simpson

OptComp

A. Koch

ındlagen

Vertauschung

Ablaufplanun

Platzierung

## Grundlagen des Algorithmus



Verwaltete Daten

- Durch welche Inter-Block Kopie soll eine Variable ersetzt werden?
- Durch welche lokale (Intra-Block) Kopie soll eine Variable ersetzt werden?

OptComp

A. Koch

arundlagen ost Copy

Vertauschung

Platzierung

## Grundlagen des Algorithmus



- Verwaltete Daten
- Durch welche Inter-Block Kopie soll eine Variable ersetzt werden?
- Durch welche lokale (Intra-Block) Kopie soll eine Variable ersetzt werden?

OptComp

A. Koch

arundlagen

Vertauschung

Platzierung



## Grundlagen des Algorithmus



- OptComp

  A. Koch
- rundlagen
- Lost Copy
- Ablaufplanun
- Platzierung
- Briggs-Algorithmus

- Verwaltete Daten
- Durch welche Inter-Block Kopie soll eine Variable ersetzt werden?
- Durch welche lokale (Intra-Block) Kopie soll eine Variable ersetzt werden?



- Algorithmus bearbeitet Blöcke in Pre-Order-Reihenfolge im Dominatorbaum
  - Bei Abstieg: Inter-block Daten aus Vorgänger übernehmen
  - Bei Aufstieg: Inter-block Daten zurücksetzen
- ⇒Analog zu Geltungsbereichen von Symboltabellen
  - ullet Globale Hash-Map Variable o Stack: Stacks [v]
  - Bei Anlegen einer neuen Kopie

a Sparie verwertuungen van vanadis duren leizies z

Bei Verlassen eines Geltungsbereichse

OptComp

A. Koch

ost Copy

Vertauschun

noiauipiaiiuii

Platzierung





OptComp

A Koch

Briggs-Algorithmus

- Algorithmus bearbeitet Blöcke in Pre-Order-Reihenfolge im Dominatorbaum
  - Bei Abstieg: Inter-block Daten aus Vorgänger übernehmen
  - Bei Aufstieg: Inter-block Daten zurücksetzen
- ⇒Analog zu Geltungsbereichen von Symboltabellen
  - Globale Hash-Map Variable → Stack: Stacks [v]
    - Bei Anlegen einer neuen Kopie

erseizen

Rei Verlassen eines Geltungshereichs



- Algorithmus bearbeitet Blöcke in Pre-Order-Reihenfolge im Dominatorbaum
  - Bei Abstieg: Inter-block Daten aus Vorgänger übernehmen
  - Bei Aufstieg: Inter-block Daten zurücksetzen

⇒Analog zu Geltungsbereichen von Symboltabellen

Globale Hash-Map Variable → Stack: Stacks [v]
 Bei Anlegen einer neuen Kopie

OptComp

A. Koch

oet Copy

Vertauschung

Abiauipianung

Prings



- Algorithmus bearbeitet Blöcke in Pre-Order-Reihenfolge im Dominatorbaum
  - Bei Abstieg: Inter-block Daten aus Vorgänger übernehmen
  - Bei Aufstieg: Inter-block Daten zurücksetzen

⇒Analog zu Geltungsbereichen von Symboltabellen

Globale Hash-Map Variable → Stack: Stacks [v]
 Bei Anlegen einer neuen Kopie

OptComp

A. Koch

oet Copy

Vertauschung

Abiauipianung

Prings



- Algorithmus bearbeitet Blöcke in Pre-Order-Reihenfolge im Dominatorbaum
  - Bei Abstieg: Inter-block Daten aus Vorgänger übernehmen
  - Bei Aufstieg: Inter-block Daten zurücksetzen
- ⇒Analog zu Geltungsbereichen von Symboltabellen
  - Globale Hash-Map Variable → Stack: Stacks [v]
  - Bei Anlegen einer neuen Kopie
    - Push des Ziels auf Stack von Ursprungsvariable
       Spätere Verwendungen von Variable durch letztes Ziel ersetzen
  - Bei Verlassen eines Geltungsbereichs
    - Alle in diesem Block gemachten Einträge entfernen
       Ursprungsvariablen lokal je Block in pushed merken



A. Koch

arundlagen ost Copy

Ablaufplanung

latzierung .





- Algorithmus bearbeitet Blöcke in Pre-Order-Reihenfolge im Dominatorbaum
  - Bei Abstieg: Inter-block Daten aus Vorgänger übernehmen
  - Bei Aufstieg: Inter-block Daten zurücksetzen
- ⇒Analog zu Geltungsbereichen von Symboltabellen
  - Globale Hash-Map Variable → Stack: Stacks [v]
  - Bei Anlegen einer neuen Kopie
    - Push des Ziels auf Stack von Ursprungsvariable
    - Spätere Verwendungen von Variable durch letztes Ziel ersetzen
  - Bei Verlassen eines Geltungsbereichs
    - Alle in diesem Block gemachten Einträge entfernen
       Ursprungsvariablen lokal je Block in pushed merkei



A. Koch

ost Copy

Vertauschung

Ablaufplanung

latzierung





- Algorithmus bearbeitet Blöcke in Pre-Order-Reihenfolge im Dominatorbaum
  - Bei Abstieg: Inter-block Daten aus Vorgänger übernehmen
  - Bei Aufstieg: Inter-block Daten zurücksetzen
- ⇒Analog zu Geltungsbereichen von Symboltabellen
  - Globale Hash-Map Variable → Stack: Stacks [v]
  - Bei Anlegen einer neuen Kopie
    - Push des Ziels auf Stack von Ursprungsvariable
    - Spätere Verwendungen von Variable durch letztes Ziel ersetzen
  - Bei Verlassen eines Geltungsbereichs
    - Alle in diesem Block gemachten Einträge entfernenUrsprungsvariablen lokal je Block in pushed merken



A. Koch

arundlagen ost Copy

Vertauschung Ablaufplanung

latzierung .





- Algorithmus bearbeitet Blöcke in Pre-Order-Reihenfolge im Dominatorbaum
  - Bei Abstieg: Inter-block Daten aus Vorgänger übernehmen
  - Bei Aufstieg: Inter-block Daten zurücksetzen
- ⇒Analog zu Geltungsbereichen von Symboltabellen
  - Globale Hash-Map Variable → Stack: Stacks [v]
  - Bei Anlegen einer neuen Kopie
    - Push des Ziels auf Stack von Ursprungsvariable
    - Spätere Verwendungen von Variable durch letztes Ziel ersetzen
  - Bei Verlassen eines Geltungsbereichs
    - Alle in diesem Block gemachten Einträge entfernenUrsprungsvariablen lokal je Block in pushed merken



A. Koch

rundlagen ost Copy

Vertauschung
Ablaufplanung

latzierung .



- Algorithmus bearbeitet Blöcke in Pre-Order-Reihenfolge im Dominatorbaum
  - Bei Abstieg: Inter-block Daten aus Vorgänger übernehmen
  - Bei Aufstieg: Inter-block Daten zurücksetzen
- ⇒Analog zu Geltungsbereichen von Symboltabellen
  - Globale Hash-Map Variable → Stack: Stacks[v]
  - Bei Anlegen einer neuen Kopie
    - Push des Ziels auf Stack von Ursprungsvariable
    - Spätere Verwendungen von Variable durch letztes Ziel ersetzen
  - Bei Verlassen eines Geltungsbereichs
    - Alle in diesem Block gemachten Einträge entfernen
    - Ursprungsvariablen lokal je Block in pushed merken



A. Koch

rundlagen ost Copy

Vertauschung Ablaufplanung

iaiziei urig



- Algorithmus bearbeitet Blöcke in Pre-Order-Reihenfolge im Dominatorbaum
  - Bei Abstieg: Inter-block Daten aus Vorgänger übernehmen
  - Bei Aufstieg: Inter-block Daten zurücksetzen
- ⇒Analog zu Geltungsbereichen von Symboltabellen
  - Globale Hash-Map Variable → Stack: Stacks[v]
  - Bei Anlegen einer neuen Kopie
    - Push des Ziels auf Stack von Ursprungsvariable
    - Spätere Verwendungen von Variable durch letztes Ziel ersetzen
  - Bei Verlassen eines Geltungsbereichs
    - Alle in diesem Block gemachten Einträge entfernen
    - Ursprungsvariablen lokal je Block in pushed merker



A. Koch

rundlagen ost Copy

Vertauschung Ablaufplanung

iaiziei urig



- Algorithmus bearbeitet Blöcke in Pre-Order-Reihenfolge im Dominatorbaum
  - Bei Abstieg: Inter-block Daten aus Vorgänger übernehmen
  - Bei Aufstieg: Inter-block Daten zurücksetzen
- ⇒Analog zu Geltungsbereichen von Symboltabellen
  - Globale Hash-Map Variable → Stack: Stacks[v]
  - Bei Anlegen einer neuen Kopie
    - Push des Ziels auf Stack von Ursprungsvariable
    - Spätere Verwendungen von Variable durch letztes Ziel ersetzen
  - Bei Verlassen eines Geltungsbereichs
    - Alle in diesem Block gemachten Einträge entfernen
    - Ursprungsvariablen lokal je Block in pushed merken



A. Koch

rundlagen ost Copy

vertauschung Ablaufplanung

Brigas-



Einfachere Struktur: Je Block

- Lokale Hash-Map von Ursprungsvariable auf
- Spätere Verwendungen von Variable durch aktuelle
- Flag, ob Variable als Phi-Parameter verwendet wird:

OptComp

A Koch





Einfachere Struktur: Je Block

 Lokale Hash-Map von Ursprungsvariable auf Zielvariable: Map [v]

- Spätere Verwendungen von Variable durch aktuelle Kopie ersetzen
- Flag, ob Variable als Phi-Parameter verwendet wird:
   IsPhiParam[v]
  - Dann geschickte Ablaufplanung erforderlich
  - Oder sogar Aufbrechen von Zyklus

OptComp

A. Koch

Lost Copy

Ablaufplanung

rians-



Einfachere Struktur: Je Block

 Lokale Hash-Map von Ursprungsvariable auf Zielvariable: Map [v]

- Spätere Verwendungen von Variable durch aktuelle Kopie ersetzen
- Flag, ob Variable als Phi-Parameter verwendet wird: IsPhiParam[v]
  - Dann geschickte Ablaufplanung erforderlich
     Oder sogar Aufbrechen von Zyklus

OptComp

A. Koch

Lost Conv

Vertauschung

Platzierung



Einfachere Struktur: Je Block

 Lokale Hash-Map von Ursprungsvariable auf Zielvariable: Map [v]

- Spätere Verwendungen von Variable durch aktuelle Kopie ersetzen
- Flag, ob Variable als Phi-Parameter verwendet wird: IsPhiParam[v]
  - Dann geschickte Ablaufplanung erforderlich
  - Oder sogar Aufbrechen von Zyklus

OptComp

A. Koch

ost Copy

Vertauschung Ablaufplanung

Platzierung

### Lokale Intra-Block Daten



- Einfachere Struktur: Je Block
- Lokale Hash-Map von Ursprungsvariable auf Zielvariable: Map [v]
- Spätere Verwendungen von Variable durch aktuelle Kopie ersetzen
- Flag, ob Variable als Phi-Parameter verwendet wird: IsPhiParam[v]
  - Dann geschickte Ablaufplanung erforderlich
  - Oder sogar Aufbrechen von Zyklus

OptComp

A. Koch

arundiagen

Vertauschung
Ablaufplanung

Platzierung

# Lokale Intra-Block Daten



- Einfachere Struktur: Je Block
- Lokale Hash-Map von Ursprungsvariable auf Zielvariable: Map [v]
- Spätere Verwendungen von Variable durch aktuelle Kopie ersetzen
- Flag, ob Variable als Phi-Parameter verwendet wird: IsPhiParam[v]
  - Dann geschickte Ablaufplanung erforderlich
  - Oder sogar Aufbrechen von Zyklus

OptComp

A. Koch

⊿rundiagen ₋ost Copy

Vertauschung
Ablaufplanung

latzierung

# Hauptprozedur: replace\_phi\_nodes (cfg)



Bearbeitet gesamten CFG.

Bestimme Live-Variablen durch Datenflussanalyse for alle Variablen v im CFG do

 $\texttt{Stacks}[v] \leftarrow \emptyset$ 

end for

insert\_copies(cfg.entryBlock())

alle Phi-Funktionen " $x \leftarrow \Phi(\dots)$ " im ganzen CFG löschen

OptComp

A. Koch

ost Copy

Vertauschung Ablaufplanung

Platzierung

# insert\_copies(block)



OptComp

A Koch

Brigas-

Algorithmus

# Bearbeitet rekursiv einen Block und die von ihm dominierten Blöcke in Pre-Order des Dominatorbaums.

```
Pushed \leftarrow \emptyset
for alle Anweisungen i in block do
   for alle Variablen v in i do
       ersetze alle v durch Stacks[v].top, wenn dies \neq nil
   end for
end for
schedule_copies(block)
for k ist Kind vom block im Dominator-Baum do
    insert_copies(k)
end for
for Variable v in Pushed do
   pop(Stacks[v])
end for
```

1. Pass: Initialisierung



```
Copyset \leftarrow \emptyset
for alle Nachfolger s von block do
     i \leftarrow \text{whichPred}(s, block)
    for alle Phi-Funktionen "dest := \Phi(...)" in s do
          src ← j-ter Operand der Phi-Funktion
          Copyset \leftarrow Copyset \cup \{(src, dest)\}
         Map[src] \leftarrow src
         Map[dest] \leftarrow dest
          IsPhiParam[src] \leftarrow true
    end for
end for
```

OptComp

A. Koch

A. ROCI

Grundlagen

Vertauschun

Ablaufolooup

Platzierung

2. Pass: Abarbeitungsreihenfolge bestimmen



OptComp

A Koch

- Bestimme Worklist so, dass zunächst nur konfliktfreie Kopien erzeugt werden
- Also nur solche, deren Ziel nicht als Parameter einer anderen Phi-Funktion verwendet wird.
  - Dann ist die Reihenfolge der Kopieranweisungen egal
- Sonst besteht Gefahr von Vertauschungsproblem!

```
Ablaufplanung
Platzierung
Briggs-
```

```
for alle in diesem Block zu erzeugender for alle Kopien (src, dest) ∈ Copyset do
    if ¬IsPhiParam[dest] then
        Worklist ← Worklist ∪ (src, dest)
        Copyset ← Copyset − {(src, dest)}
    end if
end for
```

2. Pass: Abarbeitungsreihenfolge bestimmen



- Bestimme Worklist so, dass zunächst nur konfliktfreie Kopien erzeugt werden
- Also nur solche, deren Ziel nicht als Parameter einer anderen Phi-Funktion verwendet wird.
  - Dann ist die Reihenfolge der Kopieranweisungen egal
- Sonst besteht Gefahr von Vertauschungsproblem!

```
OptComp
```

A. Koch

oet Copy

Vertauschung

atzieruna

```
for alle Kopien (src, dest) ∈ Copyset do

if ¬IsPhiParam[dest] then

Worklist ← Worklist ∪ (src, dest)

Copyset ← Copyset − {(src, dest)}

end if

end for
```

2. Pass: Abarbeitungsreihenfolge bestimmen



- Bestimme Worklist so, dass zunächst nur konfliktfreie Kopien erzeugt werden
- Also nur solche, deren Ziel nicht als Parameter einer anderen Phi-Funktion verwendet wird
  - Dann ist die Reihenfolge der Kopieranweisungen egal
- Sonst besteht Gefahr von Vertauschungsproblem!

```
OptComp
```

A Koch

Brigas-

Algorithmus



2. Pass: Abarbeitungsreihenfolge bestimmen



- Bestimme Worklist so, dass zunächst nur konfliktfreie Kopien erzeugt werden
- Also nur solche, deren Ziel nicht als Parameter einer anderen Phi-Funktion verwendet wird.
  - Dann ist die Reihenfolge der Kopieranweisungen egal
- Sonst besteht Gefahr von Vertauschungsproblem!

```
OptComp
```

A. Koch

Grundlagen

Vertauschung

atzierung

```
for alle in diesem Block zu erzeugenden for alle Kopien (src, dest) \in Copyset do if \neg IsPhiParam[dest] then Worklist \leftarrow Worklist \cup (src, dest) \cup Copyset \leftarrow Copyset - \{(src, dest)\} end if end for
```

2. Pass: Abarbeitungsreihenfolge bestimmen



- Bestimme Worklist so, dass zunächst nur konfliktfreie Kopien erzeugt werden
- Also nur solche, deren Ziel nicht als Parameter einer anderen Phi-Funktion verwendet wird.
  - Dann ist die Reihenfolge der Kopieranweisungen egal
- Sonst besteht Gefahr von Vertauschungsproblem!

```
OptComp
```

A. Koch

Grundlagen

Vertauschung

atzierung

```
for alle in diesem Block zu erzeugenden for alle Kopien (src, dest) \in Copyset do if \neg IsPhiParam[dest] then Worklist \leftarrow Worklist \cup (src, dest) \cup Copyset \leftarrow Copyset - \{(src, dest)\} end if end for
```

2. Pass: Abarbeitungsreihenfolge bestimmen



OptComp

A Koch

- Bestimme Worklist so, dass zunächst nur konfliktfreie Kopien erzeugt werden
- Also nur solche, deren Ziel nicht als Parameter einer anderen Phi-Funktion verwendet wird.
  - Dann ist die Reihenfolge der Kopieranweisungen egal
- Sonst besteht Gefahr von Vertauschungsproblem!

```
ertauschung
```

atzierung

```
// gehe alle in diesem Block zu erzeugenden Kopier

for alle Kopien (src, dest) ∈ Copyset do

if ¬IsPhiParam[dest] then

Worklist ← Worklist ∪ (src, dest)

Copyset ← Copyset − {(src, dest)}

end if

end for

// Worklist onthält interpret die einfachen Fälle.
```



3. Pass: Benötigte Kopien erzeugen



### Es müssen alle Kopien abgearbeitet werden

- Erstmal die "einfachen" aus der Worklist
- Später auch die komplizierteren, die noch in Copyset stehen

```
while Worklist ≠ ∅ ∨ Copyset ≠ ∅ do

// Bearbeite erst eintache Fälle, kann auch komplizierte vereinfachen
while Worklist ≠ ∅ do

⇒einfacher Fall, Pass 3a, Teil 1+2
end while

// Nun die komplizierten Fälle, erzeugt wiederum neue einfache Fälle
if Copyset ≠ ∅ then

// zirkulare Abhängigkeit, muss durch Kopieren aufgebrochen werden

⇒komplizierter Fall, Pass 3b
end if
end while
```

OptComp

A. Koch

irundlagen

Vertauschung

. Platzierung

3. Pass: Benötigte Kopien erzeugen



- Es müssen alle Kopien abgearbeitet werden
- Erstmal die "einfachen" aus der Worklist
- Spater auch die komplizierteren, die noch in Copyset stehen

```
while Worklist ≠ ∅ ∨ Copyset ≠ ∅ do

// Bearbeite erst einfache Fälle, kann auch komplizierte vereinfachen
while Worklist ≠ ∅ do

⇒einfacher Fall, Pass 3a, Teil 1+2
end while

// Nun die komplizierten Fälle, erzeugt wiederum neue einfache Fälle
if Copyset ≠ ∅ then

// zirkuläre Abhängigkeit, muss durch Kopieren aufgebrochen werden
⇒komplizierter Fall, Pass 3b
end if
```

OptComp

A. Koch

arundiagen

Vertauschung

. Platzierung

3. Pass: Benötigte Kopien erzeugen



- Es müssen alle Kopien abgearbeitet werden
- Erstmal die "einfachen" aus der Worklist
- Später auch die komplizierteren, die noch in Copyset stehen

```
⇒einfacher Fall, Pass 3a, Teil 1+2
⇒komplizierter Fall, Pass 3b
```



3. Pass: Benötigte Kopien erzeugen



- Es müssen alle Kopien abgearbeitet werden
- Erstmal die "einfachen" aus der Worklist
- Später auch die komplizierteren, die noch in Copyset stehen

```
⇒einfacher Fall, Pass 3a, Teil 1+2
⇒komplizierter Fall, Pass 3b
```



3. Pass: Benötigte Kopien erzeugen



- Es müssen alle Kopien abgearbeitet werden
- Erstmal die "einfachen" aus der Worklist
- Später auch die komplizierteren, die noch in Copyset stehen

```
while Worklist ≠ ∅ ∨ Copyset ≠ ∅ do

// Bearbeite erst einfache Fälle, kann auch komplizierte vereinfachen

while Worklist ≠ ∅ do

⇒einfacher Fall, Pass 3a, Teil 1+2

end while

// Nun die komplizierten Fälle, erzeugt wiederum neue einfache Fälle

if Copyset ≠ ∅ then

// zirkuläre Abhängigkeit, muss durch Kopieren aufgebrochen werden

⇒komplizierter Fall, Pass 3b

end if
end while
```

OptComp

A. Koch

arundiagen

Lost Copy

Ablaufplanung

latzierung

3a. Pass: Erzeuge Kopien für einfache Fälle aus Worklist, Teil 1



OptComp

A Koch

Briggs-Algorithmus

# Es müssen alle Kopien abgearbeitet werden

Erstmal die "einfachen" aus der Worklist

3a. Pass: Erzeuge Kopien für einfache Fälle aus Worklist, Teil 1



OptComp

A Koch

- Es müssen alle Kopien abgearbeitet werden
- Erstmal die "einfachen" aus der Worklist

3a. Pass: Erzeuge Kopien für einfache Fälle aus Worklist, Teil 1



OptComp

A Koch

- Es müssen alle Kopien abgearbeitet werden
- Erstmal die "einfachen" aus der Worklist

3a. Pass: Erzeuge Kopien für einfache Fälle aus Worklist, Teil 1



OptComp

A Koch

Brigas-

Algorithmus

- Es müssen alle Kopien abgearbeitet werden
- Erstmal die "einfachen" aus der Worklist

```
wähle eine beliebige anstehende Kopie (src. dest) ∈ Worklist
Worklist \leftarrow Worklist - \{(src, dest)\}
if dest ∈ block.liveOut() then
     erzeuge neue temporäre Variable "tempN"
     erzeuge Kopieranweisung "tempN := dest" vor Blockende
     // Merken für Inter-Block-Umbenennung von dest nach "tempn"
     Stacks [dest].push(''tempN'')
     Pushed ← Pushed U { dest}
     if dest wird verwendet in bedingtem Sprung am Blockende then
         ersetze Auftreten von dest durch "tempn" in allen Sprungbedingungen
     end if
else
     if dest wird verwendet in bedingtem Sprung am Blockende then
          erzeuge neue temporäre Variable "tempM"
          erzeuge Kopieranweisung "tempM := dest" vor Blockende
          ersetze Auftreten von dest durch "tempM" in allen Sprungbedingungen
    end if
end if
⇒Pass 3a. Teil 2.
```

3a. Pass: Erzeuge Kopien für einfache Fälle aus Worklist, Teil 2



# Sonderfälle vorher erledigt

- Variable war Live-Out aus Block
- Variable in Sprungbedingung am Blockende
- Jetzt Kopieranweisungen für eigentliche Phi-Funktionen erzeugen

```
# Auflösung dieses Teils der Phi-Funktion erzeuge Kopieranweisung "dest := \operatorname{Map}[src]" vor Block-Ende # merken, wo Wert von src jetzt verfügbar ist \operatorname{Map}[src] \leftarrow dest # wurde so ein Konflikt aus Copyset aufgelöst? for alle Kopien (s,d) \in Copyset mit d=src do # es gibt also eine vorher zurückgestellte Kopie mit src als Ziel # da oben der Wert von src in dest kopiert wurde, # kann src selbst nun überschrieben werden: geschickte Ablaufplanung # Worklist \leftarrow Worklist \cup \{(s,d)\} Copyset \leftarrow Copyset - \{(s,d)\} end for
```

OptComp

A. Koch

Lost Copy

Ablaufplanun

Platzierung

3a. Pass: Erzeuge Kopien für einfache Fälle aus Worklist, Teil 2



- Sonderfälle vorher erledigt
  - Variable war Live-Out aus Block
  - Variable in Sprungbedingung am Blockende
- Jetzt Kopieranweisungen für eigentliche Phi-Funktionen erzeugen

```
## Auflösung dieses Teils der Phi-Funktion erzeuge Kopieranweisung "dest := \operatorname{Map}[src]" vor Block-Ende ### merken, wo Wert von src jetzt verfügbar ist \operatorname{Map}[src] \leftarrow dest ### wurde so ein Konflikt aus Copyset aufgelöst? ** for alle Kopien (s,d) \in Copyset mit d=src do ### with also eine vorher zurückgesteilte Kopie mit src als Ziel ### da oben der Wert von src in dest kopiert wurde, ### kann src selbst nun überschrieben werden: geschickte Ablaufplanung ### Worklist \leftarrow Worklist \cup \{(s,d)\} ** Copyset \leftarrow Copyset - \{(s,d)\} end for
```

OptComp

A. Koch

oet Copy

Vertauschun

Platzioruna

3a. Pass: Erzeuge Kopien für einfache Fälle aus Worklist, Teil 2



- Sonderfälle vorher erledigt
  - Variable war Live-Out aus Block
  - Variable in Sprungbedingung am Blockende
- Jetzt Kopieranweisungen für eigentliche Phi-Funktionen erzeugen

```
// Auflösung dieses Teils der Phi-Funktion erzeuge Kopieranweisung "dest := \operatorname{Map}[src]" vor Block-Ende // merken, wo Wert von src jetzt verfügbar ist \operatorname{Map}[src] \leftarrow dest // wurde so ein Konflikt aus Copyset aufgelöst? for alle Kopien (s,d) \in Copyset mit d=src do // es gibt also eine vorher zurückgesteilte Kopie mit src als Ziel // da oben der Wert von src in dest kopiert wurde, // kann src selbst nun überschrieben werden: geschickte Ablaufplanung Worklist \leftarrow Worklist \cup \{(s,d)\} Copyset \leftarrow Copyset - \{(s,d)\} end for
```

OptComp

A. Koch

Loet Conv

Vertauschun

Ablautplanun

Dulama

3a. Pass: Erzeuge Kopien für einfache Fälle aus Worklist, Teil 2



- Sonderfälle vorher erledigt
  - Variable war Live-Out aus Block
  - Variable in Sprungbedingung am Blockende
- Jetzt Kopieranweisungen für eigentliche Phi-Funktionen erzeugen

```
// Auflösung dieses Teils der Phi-Funktion erzeuge Kopieranweisung "dest := \operatorname{Map}[src]" vor Block-Ende // merken, wo Wert von src jetzt verfügbar ist \operatorname{Map}[src] \leftarrow dest // wurde so ein Konflikt aus Copyset aufgelöst? for alle Kopien (s,d) \in Copyset mit d=src do // es gibt also eine vorher zurückgestellte Kopie mit src als Ziel // da oben der Wert von src in dest kopiert wurde, // kann src selbst nun überschrieben werden: geschickte Ablaufplanung Worklist \leftarrow Worklist \cup \{(s,d)\} Copyset \leftarrow Copyset - \{(s,d)\} end for
```

OptComp

A. Koch

Grundlagen Lost Copy

Vertauschung

. latzierung

3a. Pass: Erzeuge Kopien für einfache Fälle aus Worklist, Teil 2



- Sonderfälle vorher erledigt
  - Variable war Live-Out aus Block
  - Variable in Sprungbedingung am Blockende
- Jetzt Kopieranweisungen für eigentliche Phi-Funktionen erzeugen

```
# Auflösung dieses Teils der Phi-Funktion erzeuge Kopieranweisung "dest := \operatorname{Map}[src]" vor Block-Ende # merken, wo Wert von src jetzt verfügbar ist \operatorname{Map}[src] \leftarrow dest # wurde so ein Konflikt aus Copyset aufgelöst? for alle Kopien (s,d) \in Copyset mit d = src do # es gibt also eine vorher zurückgestellte Kopie mit src als Ziel # da oben der Wert von src in dest kopiert wurde, # kann src selbst nun überschrieben werden: geschickte Ablaufplanung # Worklist \leftarrow Worklist \cup \{(s,d)\} Copyset \leftarrow Copyset - \{(s,d)\} end for
```

A. Koch

arundiagen

Vertauschung

Ablaufplanung

..

3a. Pass: Erzeuge Kopien für einfache Fälle aus Worklist, Teil 2



- Sonderfälle vorher erledigt
  - Variable war Live-Out aus Block
  - Variable in Sprungbedingung am Blockende
- Jetzt Kopieranweisungen für eigentliche Phi-Funktionen erzeugen

```
// Auflösung dieses Teils der Phi-Funktion erzeuge Kopieranweisung "dest := Map[src]" vor Block-Ende // merken, wo Wert von src jetzt verfügbar ist Map[src] \leftarrow dest // wurde so ein Konflikt aus Copyset aufgelöst? for alle Kopien (s,d) \in Copyset mit d=src do // es gibt also eine vorner zurückgesteilte Kopie mit src als Ziel // da oben der Wert von src in dest kopiert wurde, // kann src selbst nun überschrieben werden: geschickte Ablaufplanung Worklist \leftarrow Worklist \cup \{(s,d)\} Copyset \leftarrow Copyset - \{(s,d)\} end for
```

OptComp

A. Koch

and Carry

Vertauschung

Ablaufplanung

riatzierung

3b. Pass: Vereinfache komplizierte Fälle aus Copyset



# Nun keine einfachen Fälle mehr übrig

- Geschickte Ablaufplanung hat nicht gereicht
- $\bullet$  Es existieren zirkuläre Abhängigkeiten ( $\to$  Vertauschungsproblem)
- Aufbrechen durch Kopieroperationen

```
// picke ein Element aus dem Zyklus
wähle ein beliebiges (src, dest) ∈ Copyset
erzeuge neue temporäre Variable "tempo"
erzeuge Kopieranweisung "tempo := dest" vor Block-Ende
// dest kann also überschrieben werden, der Zyklus ist gebrochen
Map[dest] ← "tempo"
// Nun naben wir wieder einen einfachen Fall für Worklist
Copyset ← Copyset ← ((src, dest))
```

OptComp

A. Koch

oet Copy

Vertauschung

Abiautpianun ---

Platzierung

3b. Pass: Vereinfache komplizierte Fälle aus Copyset



- Nun keine einfachen Fälle mehr übrig
  - Geschickte Ablaufplanung hat nicht gereicht
  - $\bullet$  Es existieren zirkuläre Abhängigkeiten (  $\to$  Vertauschungsproblem)
- Aufbrechen durch Kopieroperationen

```
## picke ein Element aus dem Zyklus

wähle ein beliebiges (src, dest) ∈ Copyset

erzeuge neue temporäre Variable "tempo"

erzeuge Kopieranweisung "tempo := dest" vor Block-Ende

## dest kann also überschrieben werden, der Zyklus ist gebrochen

Map[dest] ← "tempo"

## Nun haben wir wieder einen einfachen Fall für Worklist

Copyset ← Copyset − {(src, dest)}

Worklist ← Worklist∪ {(src, dest)}
```



A. Koch

arundlager

Vertauschung

Ablaufplanun

latzierung

3b. Pass: Vereinfache komplizierte Fälle aus Copyset



- Nun keine einfachen Fälle mehr übrig
  - Geschickte Ablaufplanung hat nicht gereicht
  - $\bullet$  Es existieren zirkuläre Abhängigkeiten ( $\to$  Vertauschungsproblem)
- Aufbrechen durch Kopieroperationen

```
## picke ein Element aus dem Zyklus
wähle ein beliebiges (src, dest) ∈ Copyset
erzeuge neue temporäre Variable "tempo"
erzeuge Kopieranweisung "tempo := dest" vor Block-Ende
### dest wert von dest steht jetzt in tempo
```

```
\texttt{Map}[\textit{dest}] \leftarrow \texttt{"temp0"}
```

```
Copyset \leftarrow Copyset - \{(src, dest)\}

Worklist \leftarrow Worklist \cup \{(src, dest)\}
```



A. Koch

rundiagen

Losi Copy

Ablaufolanun

Platzierung

3b. Pass: Vereinfache komplizierte Fälle aus Copyset



- Nun keine einfachen Fälle mehr übrig
  - Geschickte Ablaufplanung hat nicht gereicht
  - $\bullet$  Es existieren zirkuläre Abhängigkeiten ( $\to$  Vertauschungsproblem)
- Aufbrechen durch Kopieroperationen

```
wähle ein beliebiges (src, dest) ∈ Copyset
erzeuge neue temporäre Variable "tempo"
erzeuge Kopieranweisung "tempo" := dest" vor Block-Ende
```

```
\texttt{Map}[\textit{dest}] \leftarrow \texttt{"temp0"}
```

```
Copyset \leftarrow Copyset - \{(src, dest)\}

Worklist \leftarrow Worklist \cup \{(src, dest)\}
```



A Koch

net Conv

Vertauschung

Ablaufplanun

atzierung

3b. Pass: Vereinfache komplizierte Fälle aus Copyset



- Nun keine einfachen Fälle mehr übrig
  - Geschickte Ablaufplanung hat nicht gereicht
  - $\bullet$  Es existieren zirkuläre Abhängigkeiten ( $\to$  Vertauschungsproblem)
- Aufbrechen durch Kopieroperationen

```
wähle ein beliebiges (src, dest) ∈ Copyset
erzeuge neue temporäre Variable "tempo"
erzeuge Kopieranweisung "tempo" := dest" vor Block-Ende
```

```
\texttt{Map}[\textit{dest}] \leftarrow \texttt{"temp0"}
```

```
Copyset \leftarrow Copyset - \{(src, dest)\}

Worklist \leftarrow Worklist \cup \{(src, dest)\}
```



A Koch

net Conv

Vertauschung

Ablaufplanun

atzierung

3b. Pass: Vereinfache komplizierte Fälle aus Copyset



- Nun keine einfachen Fälle mehr übrig
  - Geschickte Ablaufplanung hat nicht gereicht
  - $\bullet$  Es existieren zirkuläre Abhängigkeiten ( $\to$  Vertauschungsproblem)
- Aufbrechen durch Kopieroperationen

```
// picke ein Element aus dem Zyklus
wähle ein beliebiges (src, dest) ∈ Copyset
erzeuge neue temporäre Variable "tempo"
erzeuge Kopieranweisung "tempo := dest" vor Block-Ende
// der Wert von dest steht jetzt in tempo
// dest kann also überschrieben werden, der Zyklus ist gebrochen
Map[dest] ← "tempo"
// Nun haben wir wieder einen einfachen Fall für Worklist
Copyset ← Copyset − {(src, dest)}
Worklist ← Worklist∪ {(src, dest)}
```

OptComp

A. Koch

rundiagen

Vertauschung

Ablaufplanun

atzierung



- Beispiele aus Folien so mit Papier und Bleistift durchrechnen
- Erläuterung des Algorithmus in Briggs auf p. 29...33
- Fehler bei Briggs
  - "Variable kommt in Sprungbedingung vor" nicht behandelt
  - Hier in Pass 3a, Teil 1 erledigt
- Implementieren erst nach dem Verstehen!
- Falls doch Vorarbeiten gemacht werden sollen
  - Datenflussanalyse f
    ür Live-Variables

OptComp

A. Koch

ost Copy

Vertauschung

latziorung





- Beispiele aus Folien so mit Papier und Bleistift durchrechnen
- Erläuterung des Algorithmus in Briggs auf p. 29...33
- Fehler bei Briggs
  - variable kommt in Sprungbedingung vor hicht behandelt
  - Hier in Pass 3a, Teil 1 erledigt
- Implementieren erst nach dem Verstehen
- Falls doch Vorarbeiten gemacht werden sollen
  - Datenflussanalyse f
    ür Live-Variables

OptComp

A. Koch

ost Copy

Vertauschung

lotziorung

Briggs-





- Beispiele aus Folien so mit Papier und Bleistift durchrechnen
- Erläuterung des Algorithmus in Briggs auf p. 29...33
- Fehler bei Briggs
  - "Variable kommt in Sprungbedingung vor" nicht behandelt
  - Hier in Pass 3a, Teil 1 erledigt
- Implementieren erst nach dem Verstehen
- Falls doch Vorarbeiten gemacht werden sollen
  - Datenflussanalyse f
    ür Live-Variables

OptComp

A. Koch

Lost Copy

Vertauschung

. . .





- Beispiele aus Folien so mit Papier und Bleistift durchrechnen
- Erläuterung des Algorithmus in Briggs auf p. 29...33
- Fehler bei Briggs
  - "Variable kommt in Sprungbedingung vor" nicht behandelt
  - Hier in Pass 3a, Teil 1 erledigt
- Implementieren erst nach dem Verstehen!
- Falls doch Vorarbeiten gemacht werden sollen
  - Datenflussanalyse f
    ür Live-Variables

OptComp

A. Koch

ost Copy

Vertauschung

latzierung





- Beispiele aus Folien so mit Papier und Bleistift durchrechnen
- Erläuterung des Algorithmus in Briggs auf p. 29...33
- Fehler bei Briggs
  - "Variable kommt in Sprungbedingung vor" nicht behandelt
  - Hier in Pass 3a, Teil 1 erledigt
- Implementieren erst nach dem Verstehen
- Falls doch Vorarbeiten gemacht werden sollen
  - Datenflussanalyse f
    ür Live-Variables

OptComp

A. Koch

ost Copy

Vertauschung

· ·latzierung





- Beispiele aus Folien so mit Papier und Bleistift durchrechnen
- Erläuterung des Algorithmus in Briggs auf p. 29...33
- Fehler bei Briggs
  - "Variable kommt in Sprungbedingung vor" nicht behandelt
  - Hier in Pass 3a, Teil 1 erledigt
- Implementieren erst nach dem Verstehen!
- Falls doch Vorarbeiten gemacht werden sollen
  - Datenflussanalyse f
    ür Live-Variables

OptComp

A. Koch

ost Copy

Vertauschung

Platzierung



- Beispiele aus Folien so mit Papier und Bleistift durchrechnen
- Erläuterung des Algorithmus in Briggs auf p. 29...33
- Fehler bei Briggs
  - "Variable kommt in Sprungbedingung vor" nicht behandelt
  - Hier in Pass 3a, Teil 1 erledigt
- Implementieren erst nach dem Verstehen!
- Falls doch Vorarbeiten gemacht werden sollen
  - Datenflussanalyse f
    ür Live-Variables

OptComp

A. Koch

arundlagen ost Copy

Vertauschung Ablaufplanung

Platzierung



- Beispiele aus Folien so mit Papier und Bleistift durchrechnen
- Erläuterung des Algorithmus in Briggs auf p. 29...33
- Fehler bei Briggs
  - "Variable kommt in Sprungbedingung vor" nicht behandelt
  - Hier in Pass 3a, Teil 1 erledigt
- Implementieren erst nach dem Verstehen!
- Falls doch Vorarbeiten gemacht werden sollen
  - Datenflussanalyse f
    ür Live-Variables

OptComp

A. Koch

irundlagen ost Copy

/ertauschung

Platzierung