Karlsruher Institut für Technologie

Procedure Cloning im Kontext SSA-basierter Zwischensprachen

Raphael von der Grün

Lehrstuhl Programmierparadigmen, IPD Snelting

Problem:
Optimiere folgendes Programm

Karlsruher Institut für Technologie

```
root = randomTree()
walk(root, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

Inlining:
Einsetzen der Prozedur an der Aufrufstelle

Karlsruher Institut für Technologie

```
root = randomTree()
walk(root, incr, None)



def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

Inlining:
Einsetzen der Prozedur an der Aufrufstelle

Karlsruher Institut für Technologie

```
root = randomTree()
walk(root, incr, None)


def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

```
root = randomTree()
incr(root)
walk(root.left,  incr, None)
walk(root.right, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

Inlining:
Einsetzen der Prozedur an der Aufrufstelle

KIT — Karlsruher Institut für Technologie

```
root = randomTree()
walk(root, incr, None)


def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

```
root = randomTree()
incr(root)
walk(root.left,  incr, None)
walk(root.right, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

Inlining:
Einsetzen der Prozedur an der Aufrufstelle

```
root = randomTree()
walk(root, incr, None)


def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

```
root = randomTree()
root.count +=1
walk(root.left,  incr, None)
walk(root.right, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

$\Rightarrow$ Programmwachstum an jeder Aufrufstelle

Procedure Cloning: Einsetzen konstanter Argumente in die Prozedur

```
root = randomTree()
walk(root, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

Procedure Cloning:
Einsetzen konstanter Argumente in die Prozedur


```
root = randomTree()
walk(root, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

```
root = randomTree()
incrTree(root)

def incrTree(n):
  if not n: return
  incr(n)
  incrTree(n.left)
  incrTree(n.right)

def incr(n):
  n.count += 1
```

Procedure Cloning:
Einsetzen konstanter Argumente in die Prozedur

```
root = randomTree()
walk(root, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

```
root = randomTree()
incrTree(root)

def incrTree(n):
  if not n: return
  incr(n)
  incrTree(n.left)
  incrTree(n.right)

def incr(n):
  n.count += 1
```

Procedure Cloning: Einsetzen konstanter Argumente in die Prozedur

Karlsruher Institut für Technologie

```
root = randomTree()
walk(root, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

```
root = randomTree()
incrTree(root)

def incrTree(n):
  if not n: return
  n.count +=1
  incrTree(n.left)
  incrTree(n.right)
```

$\Rightarrow$ Spezialisierte Prozedur kann wiederverwendet werden

Karlsruher Institut für Technologie

Procedure Cloning im Detail

Das Verfahren

Karlsruher Institut für Technologie

Problem

Procedure Cloning kann exponentielles Wachstum verursachen

Zweistufiger Ansatz[1]

1. Interessante Parameter identifizieren[2]
2. Klonen (beschränkt auf interessante Parameter)

---

[1] inspirert durch K. D. Cooper, M. W. Hall, and K. Kennedy, "A methodology for procedure cloning," Computer Languages, vol. 19, no. 2, pp. 105–117, 1993.

[2] nach M. W. Hall, Managing interprocedural optimization. PhD thesis, Rice University, 1991

Wichtige Parameter

Karlsruher Institut für Technologie

```
root = randomTree()
walk(root, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

Wichtige Parameter. . .

- beeinflussen Kontrollfluss
- sind Zieladressen von
  - Funktionsaufrufen
  - Lese-/Schreibzugriffen

Nach lokaler Analyse folgt interprozedurale Propagation

Wichtige Parameter

Karlsruher Institut für Technologie

```
root = randomTree()
walk(root, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

Wichtige Parameter. . .

- beeinflussen Kontrollfluss
- sind Zieladressen von
  - Funktionsaufrufen
  - Lese-/Schreibzugriffen

Nach lokaler Analyse folgt interprozedurale Propagation

Wichtige Parameter

Karlsruher Institut für Technologie

```
root = randomTree()
walk(root, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

Wichtige Parameter...

- beeinflussen Kontrollfluss
- sind Zieladressen von
  - Funktionsaufrufen
  - Lese-/Schreibzugriffen

Nach lokaler Analyse folgt interprozedurale Propagation

Wichtige Parameter

Karlsruher Institut für Technologie

```
root = randomTree()
walk(root, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

Wichtige Parameter…

- beeinflussen Kontrollfluss
- sind Zieladressen von
  - Funktionsaufrufen
  - Lese-/Schreibzugriffen

Nach lokaler Analyse folgt interprozedurale Propagation

Wichtige Parameter

Karlsruher Institut für Technologie

```
root = randomTree()
walk(root, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

Wichtige Parameter. . .

- beeinflussen Kontrollfluss
- sind Zieladressen von
  - Funktionsaufrufen
  - Lese-/Schreibzugriffen

Nach lokaler Analyse folgt interprozedurale Propagation

Klonen im Detail

Karlsruher Institut für Technologie

```
root = randomTree()
walk(root, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

Klonen im Detail

Karlsruher Institut für Technologie

```
root = randomTree()
walk(root, incr, None)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

Klonen im Detail

Karlsruher Institut für Technologie

```
root = randomTree()
incrTree(root)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

```
def incrTree(n):
  if not n: return
  incr(n)
  walk(n.left,  incr, None)
  walk(n.right, incr, None)
```

Klonen im Detail

Karlsruher Institut für Technologie

```
root = randomTree()
incrTree(root)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

```
def incrTree(n):
  if not n: return
  incr(n)
  walk(n.left,  incr, None)
  walk(n.right, incr, None)
```

Klonen im Detail

Karlsruher Institut für Technologie

```
root = randomTree()
incrTree(root)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

```
def incrTree(n):
  if not n: return
  incr(n)
  walk(n.left,  incr, None)
  walk(n.right, incr, None)
```

Klonen im Detail

Karlsruher Institut für Technologie

```
root = randomTree()
incrTree(root)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

```
def incrTree(n):
  if not n: return
  incr(n)
  incrTree(n.left)
  incrTree(n.right)
```

Klonen im Detail

Karlsruher Institut für Technologie

```
root = randomTree()
incrTree(root)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

```
def incrTree(n):
  if not n: return
  incr(n)
  incrTree(n.left)
  incrTree(n.right)
```

Klonen im Detail

Karlsruher Institut für Technologie

```
root = randomTree()
incrTree(root)

def walk(n, pre, post):
  if not n: return
  if pre: pre(n)
  walk(n.left,  pre, post)
  walk(n.right, pre, post)
  if post: post(n)

def incr(n):
  n.count += 1
```

```
def incrTree(n):
  if not n: return
  incr(n)
  incrTree(n.left)
  incrTree(n.right)
```

Karlsruher Institut für Technologie

Optimale Klon-Auswahl

Kostenmodell zur Klon-Bewertung

Karlsruher Institut für Technologie

Lokale Kosten

$\varrho(p)$ Ertrag, $\widetilde{\gamma}(p)$ Kosten, $\pi(p)$ Profit für Aufnehmen von Klon p

Aber: Kosten und Ertrag iA. nicht in der selben Einheit.

$\Rightarrow$ Umrechnungsfaktor $\alpha \in \mathbb{R}_{+}$, $\gamma(p)$ umgerechnete Kosten.

$$\gamma(p) := \alpha\widetilde{\gamma}(p) \qquad \pi(p) := \varrho(p) - \gamma(p).$$

Globale Kosten

Entsprechende Kennzahlen für Programm A mit $f_A : \textbf{Procs} \to \mathbb{R}_{\geq 0}$ Schätzer für erwartete Anzahl der Aufrufe einer Prozedur während eines Programmdurchlaufs von A.

$$\widehat{\varrho}(A) := \sum_{p \in A} f_A(p)\varrho(p) \qquad \widehat{\gamma}(A) := \sum_{p \in A} \gamma(p) \qquad \widehat{\pi}(A) := \widehat{\varrho}(A) - \widehat{\gamma}(A).$$

Optimale Klon-Auswahl ist NP-schwer

Karlsruher Institut für Technologie

Sei A^+ das geklonte Programm zu A. Betrachte einfache Variante mit

$$f_{A^+}(p) \equiv [p \text{ ist aus } A \text{ erreichbar}].$$

Dann lässt sich das Optimierungsproblem OPT-CS

$$\max_{A' \subseteq A^+} \quad \widehat{\pi}(A') = \sum_{p \in A'} f_{A'}(p)\varrho(p) - \gamma(p)$$

als Graphproblem auf Call-Graph von A^+ ähnlich dem *Steinerbaumproblem* auffassen, indem A zu *Wurzel* kontrahiert wird.

Wir haben per Reduktion von SET-COVER gezeigt, dass OPT-CS NP-schwer ist.

Karlsruher Institut für Technologie

Evaluation

Rahmenbedingungen der Evaluation

Karlsruher Institut für Technologie

Test mit allen C-Benchmarks der SPEC CPU2006 Suite.

Evaluierte Varianten (ohne weitere Einschränkung der Klone)

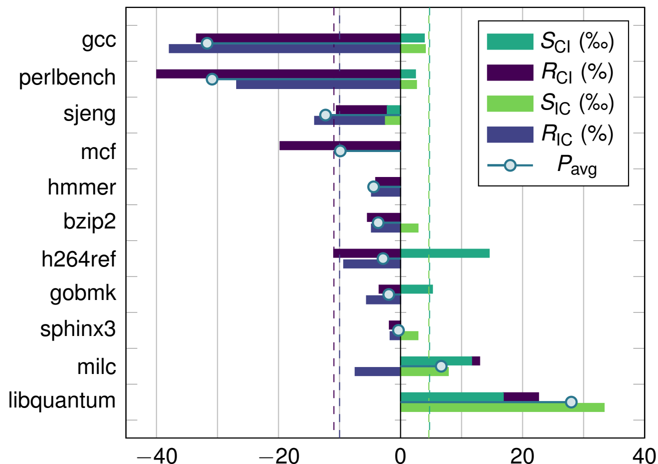
- I Kein Procedure Cloning (`cparser -m32 -O3`)
- CI Zuerst Procedure Cloning, dann Inlining
- IC Zuerst Inlining, dann Procedure Cloning

Ermittelte Größen

- T_X Mittlere Ausführungsdauer aus min. drei Durchgängen
- B_X Größe der erzeugten Binary in Bytes
- S_X Speedup von Variante X: $T_I / T_X - 1$
- R_X Größenreduktion von X: $1 - B_X / B_I$
- P_X Performance von X: $10 S_X + R_X$

CI vs IC: Kein klarer Favorit

Karlsruher Institut für Technologie



Zusammenfassung

Karlsruher Institut für Technologie

Procedure Cloning als starke Ergänzung zum Inlining

- bei Rekursion
- bei großen Prozeduren (Wiederverwendbarkeit)
- zur Verbesserung der verfügbaren Information

Programmwachstum muss kontrolliert werden

Optimale Klon-Auswahl ist NP-schwer

Karlsruher Institut für Technologie

Anhang

Die Umfassendste Abhandlung

Karlsruher Institut für Technologie

Cooper, Kennedy & Hall (1991–1993)

- Inspiration für unsere Implementierung
- Auswahl der Klone durch statische Programmanalyse
- Beinhaltet auch Merging von Klonen
- Keine Implementierung AFAICT

Rekursion kann Endlosschleifen verursachen

Karlsruher Institut für Technologie

```
root = randomTree()
incrTree(root, 0, 0)

def incrTree(n, lvl, minlvl):
  if not n: return
  if lvl >= minlvl: incr(n)
  incrTree(n.left,  lvl + 1, minlvl)
  incrTree(n.right, lvl + 1, minlvl)

def incr(n):
  n.count += 1
```

Klonen in Zyklus bei Stagnation abbrechen

Karlsruher Institut für Technologie

```
root = randomTree()
incrTree_00(root)

def incrTree_00(n):
  if not n: return
  incr(n)
  incrTree(n.left,  1, 0)
  incrTree(n.right, 1, 0)

def incrTree(n, lvl, minlvl):
  if not n: return
  if lvl >= minlvl: incr(n)
  incrTree(n.left,  lvl + 1, minlvl)
  incrTree(n.right, lvl + 1, minlvl)

def incr(n):
  n.count += 1
```

Procedure Cloning hat exponentielles Wachstum

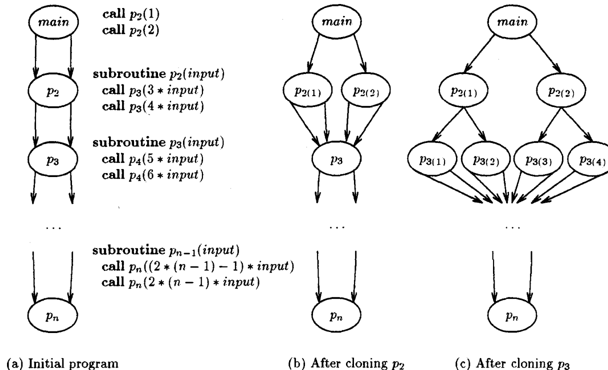


Fig. 3. Exponential code growth due to cloning. [3]

[3] Quelle: K. D. Cooper, M. W. Hall, and K. Kennedy, “A methodology for procedure cloning,” Computer Languages, vol. 19, no. 2, pp. 105–117, 1993.

Moderater Anstieg der Kompilier-Dauer – Vorteil CI

Kompilier-Dauer[4] in Minuten und Sekunden

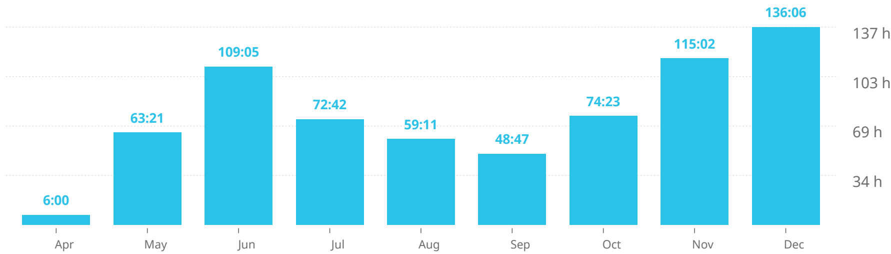
[4] Kompiliert wurden alle C-Benchmarks der SPEC CPU2006 Suite unter Verwendung der Option `makeflags = -j4`. Je Konfiguration wurde eine Messung durchgeführt.

Umfang des Projektes

Karlsruher Institut für Technologie

	LOC	
Teilbereich	Hinzugefügt	Gelöscht
Code	$+1263$	-616
Tests	$+589$	-11
Gesamt	$+1852$	-627

Arbeitsaufwand



Zeitaufwand: ca. $720\,\text{h} = 160\,\% \cdot 450\,\text{h}$

Mögliche Optimierungen

- Wachstum einschränken
  - Procedure Merging (nicht nur für Klone)
  - Gute Klon-Auswahl treffen (Kriterien?)
- Procedure Cloning mit LTO evaluieren
- Cloning und Inlining aufeinander abstimmen
- Klonbare Parametertypen erweitern
  (structs, variadische Argumente, Adressen mit Offset)
- Wichtige Parameter optimieren