

Composing Transformations to Optimize Linear Code

Thomas Noll Stefan Rieger

MOVES: Software Modeling and Verification
RWTH Aachen University, Germany

28.9.2007



ICTAC 2007, Macao

Overview

- 1 Introduction
- 2 Classical Algorithms
- 3 DAG-Optimization
- 4 Composition of the Transformations
- 5 Conclusion/Future Work

Straight-Line-Code

SLC–programs: **intermediate code** contained in **basic blocks**

- No Branching
- No Loops
- Simple arithmetic expressions (at most one operation symbol)
- **Loop bodies** consist of SLC \Rightarrow optimization important

Straight-Line-Code

SLC–programs: **intermediate code** contained in **basic blocks**

- No Branching
- No Loops
- Simple arithmetic expressions (at most one operation symbol)
- **Loop bodies** consist of SLC \Rightarrow optimization important

Applications

- Compilers
- Model Checking

Straight-Line-Code

SLC–programs: **intermediate code** contained in **basic blocks**

- No Branching
- No Loops
- Simple arithmetic expressions (at most one operation symbol)
- **Loop bodies** consist of SLC \Rightarrow optimization important

Applications

- Compilers
- Model Checking

Model

Machine with

- unlimited amount of registers
- arbitrary operations (interpretation)

Example Program

$\vec{v}_{in} : (x, y)$

$\beta : u \leftarrow 3;$

$v \leftarrow x - y;$

$w \leftarrow u + 1;$

$x \leftarrow x - y;$

$v \leftarrow w - 1;$

$u \leftarrow x - y;$

$z \leftarrow u * w;$

$u \leftarrow 2 * u;$

$\vec{v}_{out} : (u, v)$

Signature $\Sigma = (F, C)$

- finite set of **function symbols** F
- (not necessarily finite) set of **constant symbols** C

Signature $\Sigma = (F, C)$

- finite set of **function symbols** F
- (not necessarily finite) set of **constant symbols** C

Interpretation

$\mathfrak{A} := (A, \varphi)$ with **domain** A and **interpretation function**

$$\varphi : F \cup C \cup A \rightarrow \bigcup_{i=0}^{\infty} \{\delta \mid \delta : A^i \rightarrow A\}$$

Signature $\Sigma = (F, C)$

- finite set of **function symbols** F
- (not necessarily finite) set of **constant symbols** C

Interpretation

$\mathfrak{A} := (A, \varphi)$ with **domain** A and **interpretation function**

$$\varphi : F \cup C \cup A \rightarrow \bigcup_{i=0}^{\infty} \{\delta \mid \delta : A^i \rightarrow A\}$$

Example (Arithmetic on \mathbb{Z})

$$F = \{+^{(2)}, -^{(2)}, *^{(2)}\}, \quad C = \mathbb{Z}, \quad A = \mathbb{Z}$$

$$\varphi(+) = +_{\mathbb{Z}} \quad \varphi(-) = -_{\mathbb{Z}}$$

$$\varphi(*) = *_{\mathbb{Z}} \quad \varphi(z) = z$$

Assumption: Completeness

We assume that every variable is defined before use.

Assumption: Completeness

We assume that every variable is defined before use.

Types of Equivalence

- **Equivalence wrt. interpretation** \mathfrak{A} : $\pi_1 \sim_{\mathfrak{A}} \pi_2$
undecidable (arithmetic over \mathbb{Z} without division: decidable)
- **Strong equivalence**: $\pi_1 \sim \pi_2$ ($\pi_1 \sim_{\mathfrak{A}} \pi_2$ for all \mathfrak{A})
decidable using the term representation

Assumption: Completeness

We assume that every variable is defined before use.

Types of Equivalence

- **Equivalence wrt. interpretation** \mathfrak{A} : $\pi_1 \sim_{\mathfrak{A}} \pi_2$
undecidable (arithmetic over \mathbb{Z} without division: decidable)
- **Strong equivalence**: $\pi_1 \sim \pi_2$ ($\pi_1 \sim_{\mathfrak{A}} \pi_2$ for all \mathfrak{A})
decidable using the term representation

Term Representation

By successive backward variable substitution a term representation of a program π wrt. an output variable can be computed.

Overview

- 1 Introduction
- 2 Classical Algorithms
- 3 DAG-Optimization
- 4 Composition of the Transformations
- 5 Conclusion/Future Work

Program Transformation

$T : SLC \rightarrow SLC$ is called (\mathcal{A} -)program transformation if:

- 1 **Correctness:** $T(\pi) \sim_{\mathcal{A}} \pi$ for an interpretation \mathcal{A}
- 2 **Idempotency:** $T(T(\pi)) = T(\pi)$

Program Transformation

$T : SLC \rightarrow SLC$ is called (\mathcal{A} -)program transformation if:

- 1 **Correctness:** $T(\pi) \sim_{\mathcal{A}} \pi$ for an interpretation \mathcal{A}
- 2 **Idempotency:** $T(T(\pi)) = T(\pi)$

Equivalence of Program Transformations

“Mutual application” without effect:

$$T_1 \sim T_2 \Leftrightarrow T_1(T_2(\pi)) = T_2(\pi) \wedge T_2(T_1(\pi)) = T_1(\pi)$$

Dead-Code Elimination T_{DC}

Backward analysis, syntax based

i	α_i	$NV_i \subseteq V_\pi$
1	$u \leftarrow 3;$	$NV_2 = \{u, x, y\}$
2	$v \leftarrow x - y;$	$NV_3 \setminus \{w\} \cup \{u\} = \{u, x, y\}$
3	$w \leftarrow u + 1;$	$NV_4 \setminus \{x\} \cup \{x, y\} = \{w, x, y\}$
4	$x \leftarrow x - y;$	$NV_5 \setminus \{v\} \cup \{w\} = \{w, x, y\}$
5	$v \leftarrow w - 1;$	$NV_6 \setminus \{u\} \cup \{x, y\} = \{v, x, y\}$
6	$u \leftarrow x - y;$	$NV_7 = \{u, v\}$
7	$z \leftarrow u * w;$	$NV_8 \setminus \{u\} \cup \{u\} = \{u, v\}$
8	$u \leftarrow 2 * u;$	$V_{out} = \{u, v\}$

Dead-Code Elimination T_{DC}

Backward analysis, syntax based

i	α_i	$NV_i \subseteq V_\pi$	Dead-Code?
1	$u \leftarrow 3;$	$NV_2 = \{u, x, y\}$	No
2	$v \leftarrow x - y;$	$NV_3 \setminus \{w\} \cup \{u\} = \{u, x, y\}$	Yes
3	$w \leftarrow u + 1;$	$NV_4 \setminus \{x\} \cup \{x, y\} = \{w, x, y\}$	No
4	$x \leftarrow x - y;$	$NV_5 \setminus \{v\} \cup \{w\} = \{w, x, y\}$	No
5	$v \leftarrow w - 1;$	$NV_6 \setminus \{u\} \cup \{x, y\} = \{v, x, y\}$	No
6	$u \leftarrow x - y;$	$NV_7 = \{u, v\}$	No
7	$z \leftarrow u * w;$	$NV_8 \setminus \{u\} \cup \{u\} = \{u, v\}$	Yes
8	$u \leftarrow 2 * u;$	$V_{out} = \{u, v\}$	No

Common Subexpression Elimination T_{CS}

Forward analysis, syntax based.

Prevents duplicate expression evaluation where possible.

i	α_i		
1	$u \leftarrow 3;$		
2	$v \leftarrow x - y;$		
3	$w \leftarrow u + 1;$		
4	$x \leftarrow x - y;$		
5	$v \leftarrow w - 1;$		
6	$u \leftarrow x - y;$		
7	$z \leftarrow u * w;$		
8	$u \leftarrow 2 * u;$		

Common Subexpression Elimination T_{CS}

Forward analysis, syntax based.

Prevents duplicate expression evaluation where possible.

i	α_i	$AE_i \subseteq \{1, \dots, n\}$	New Instr.
1	$u \leftarrow 3;$	\emptyset	
2	$v \leftarrow x - y;$	\emptyset	$t_2 \leftarrow x - y;$ $v \leftarrow t_2;$
3	$w \leftarrow u + 1;$	$AE_2 \cup \{2\} = \{2\}$	
4	$x \leftarrow x - y;$	$AE_3 \cup \{3\} = \{2, 3\}$	$x \leftarrow t_2;$
5	$v \leftarrow w - 1;$	$(AE_4 \cup \{4\}) \setminus \{2, 4\} = \{3\}$	
6	$u \leftarrow x - y;$	$(AE_5 \cup \{5\}) = \{3, 5\}$	
7	$z \leftarrow u * w;$	$(AE_6 \cup \{6\}) \setminus \{3\} = \{5, 6\}$	
8	$u \leftarrow 2 * u;$	$(AE_7 \cup \{7\}) = \{5, 6, 7\}$	

Common Subexpression Elimination T_{CS}

Forward analysis, syntax based.

Prevents duplicate expression evaluation where possible.

i	α_i	$AE_i \subseteq \{1, \dots, n\}$	New Instr.
1	$u \leftarrow 3;$	\emptyset	
2	$v \leftarrow x - y;$	\emptyset	$t_2 \leftarrow x - y;$ $v \leftarrow t_2;$
3	$w \leftarrow u + 1;$	$AE_2 \cup \{2\} = \{2\}$	
4	$x \leftarrow x - y;$	$AE_3 \cup \{3\} = \{2, 3\}$	$x \leftarrow t_2;$
5	$v \leftarrow w - 1;$	$(AE_4 \cup \{4\}) \setminus \{2, 4\} = \{3\}$	
6	$u \leftarrow x - y;$	$(AE_5 \cup \{5\}) = \{3, 5\}$	
7	$z \leftarrow u * w;$	$(AE_6 \cup \{6\}) \setminus \{3\} = \{5, 6\}$	
8	$u \leftarrow 2 * u;$	$(AE_7 \cup \{7\}) = \{5, 6, 7\}$	

Observation: insertion of copy instructions

Constant Folding T_{CF}

Only weak equivalence is retained (partial evaluation)

i	α_i
1	$u^3 \leftarrow 3;$
2	$v^? \leftarrow x^? - y^?;$
3	$w^4 \leftarrow u^3 + 1;$
4	$x^? \leftarrow x^? - y^?;$
5	$v^3 \leftarrow w^4 - 1;$
6	$u^? \leftarrow x^? - y^?;$
7	$z^? \leftarrow u^? * w^4;$
8	$u^? \leftarrow 2 * u^?;$

Constant Folding T_{CF}

Only weak equivalence is retained (partial evaluation)

i	α_i	New Instr. α'_i
1	$u^3 \leftarrow 3;$	$u \leftarrow 3;$
2	$v^? \leftarrow x^? - y^?;$	$v \leftarrow x - y;$
3	$w^4 \leftarrow u^3 + 1;$	$w \leftarrow 4;$
4	$x^? \leftarrow x^? - y^?;$	$x \leftarrow x - y;$
5	$v^3 \leftarrow w^4 - 1;$	$v \leftarrow 3;$
6	$u^? \leftarrow x^? - y^?;$	$u \leftarrow x - y;$
7	$z^? \leftarrow u^? * w^4;$	$x \leftarrow u * 4;$
8	$u^? \leftarrow 2 * u^?;$	$u \leftarrow 2 * u;$

Constant Folding T_{CF}

Only weak equivalence is retained (partial evaluation)

i	α_i	New Instr. α'_i
1	$u^3 \leftarrow 3;$	$u \leftarrow 3;$
2	$v^? \leftarrow x^? - y^?;$	$v \leftarrow x - y;$
3	$w^4 \leftarrow u^3 + 1;$	$w \leftarrow 4;$
4	$x^? \leftarrow x^? - y^?;$	$x \leftarrow x - y;$
5	$v^3 \leftarrow w^4 - 1;$	$v \leftarrow 3;$
6	$u^? \leftarrow x^? - y^?;$	$u \leftarrow x - y;$
7	$z^? \leftarrow u^? * w^4;$	$x \leftarrow u * 4;$
8	$u^? \leftarrow 2 * u^?;$	$u \leftarrow 2 * u;$

Observation: constant folding “produces” dead-code.

Overview

- 1 Introduction
- 2 Classical Algorithms
- 3 DAG–Optimization**
- 4 Composition of the Transformations
- 5 Conclusion/Future Work

DAG

- Construction of a **directed acyclic graph**
- Proposed by Aho, Sethi and Ullman (1970) for code generation and optimization

DAG

- Construction of a **directed acyclic graph**
- Proposed by Aho, Sethi and Ullman (1970) for code generation and optimization

Correspondence

- ① DAG-Construction
 - Common Subexpression Elimination
 - Constant Folding
- ② Code Generation
 - Dead-Code Elimination

DAG-Construction

i	α_i	u	v	w	x	y	z
0					x	y	
	$u \leftarrow 3$						
	$v \leftarrow x - y$						
	$w \leftarrow u + 1$						
	$x \leftarrow x - y$						
	$v \leftarrow w - 1$						
	$u \leftarrow x - y$						
	$z \leftarrow u * w$						
	$u \leftarrow 2 * u$						

1

2

3

 x y

DAG-Construction

i	α_i	u	v	w	x	y	z
0					x	y	
1	$u \leftarrow 3$ $v \leftarrow x - y$ $w \leftarrow u + 1$ $x \leftarrow x - y$ $v \leftarrow w - 1$ $u \leftarrow x - y$ $z \leftarrow u * w$ $u \leftarrow 2 * u$	3					

1

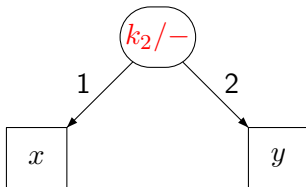
2

3

 x y

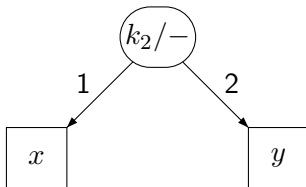
DAG-Construction

i	α_i	u	v	w	x	y	z
0					x	y	
1	$u \leftarrow 3$	3					
2	$v \leftarrow x - y$		k_2				
	$w \leftarrow u + 1$						
	$x \leftarrow x - y$						
	$v \leftarrow w - 1$						
	$u \leftarrow x - y$						
	$z \leftarrow u * w$						
	$u \leftarrow 2 * u$						



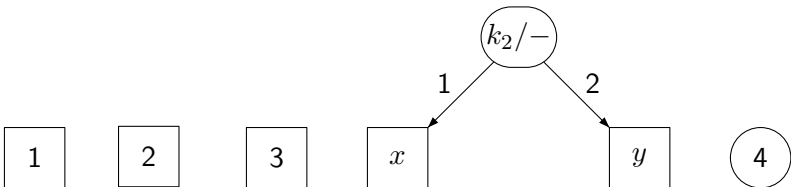
DAG-Construction

i	α_i	u	v	w	x	y	z
0					x	y	
1	$u \leftarrow 3$	3					
2	$v \leftarrow x - y$		k_2				
3	$w \leftarrow u + 1$			4			
	$x \leftarrow x - y$						
	$v \leftarrow w - 1$						
	$u \leftarrow x - y$						
	$z \leftarrow u * w$						
	$u \leftarrow 2 * u$						



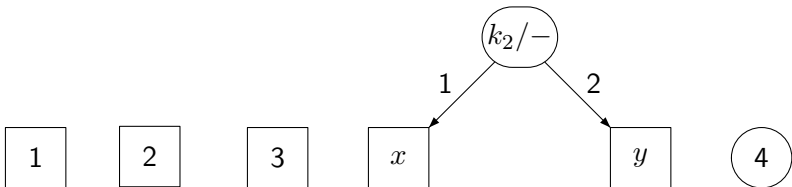
DAG-Construction

i	α_i	u	v	w	x	y	z
0					x	y	
1	$u \leftarrow 3$	3					
2	$v \leftarrow x - y$		k_2				
3	$w \leftarrow u + 1$			4			
4	$x \leftarrow x - y$				k_2		
	$v \leftarrow w - 1$						
	$u \leftarrow x - y$						
	$z \leftarrow u * w$						
	$u \leftarrow 2 * u$						



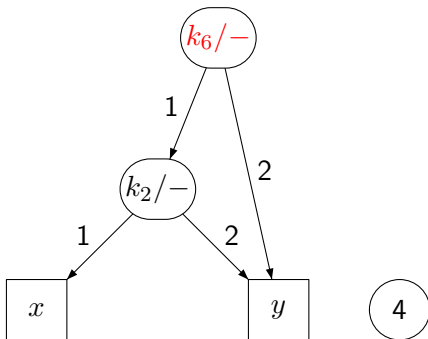
DAG-Construction

i	α_i	u	v	w	x	y	z
0					x	y	
1	$u \leftarrow 3$	3					
2	$v \leftarrow x - y$		k_2				
3	$w \leftarrow u + 1$			4			
4	$x \leftarrow x - y$				k_2		
5	$v \leftarrow w - 1$		3				
	$u \leftarrow x - y$						
	$z \leftarrow u * w$						
	$u \leftarrow 2 * u$						



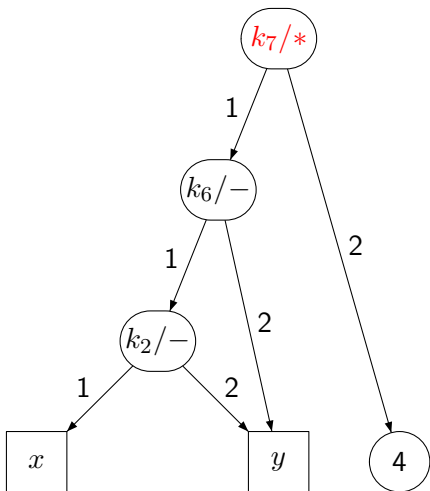
DAG-Construction

i	α_i	u	v	w	x	y	z
0					x	y	
1	$u \leftarrow 3$	3					
2	$v \leftarrow x - y$		k_2				
3	$w \leftarrow u + 1$			4			
4	$x \leftarrow x - y$				k_2		
5	$v \leftarrow w - 1$		3				
6	$u \leftarrow x - y$	k_6					
	$z \leftarrow u * w$						
	$u \leftarrow 2 * u$						



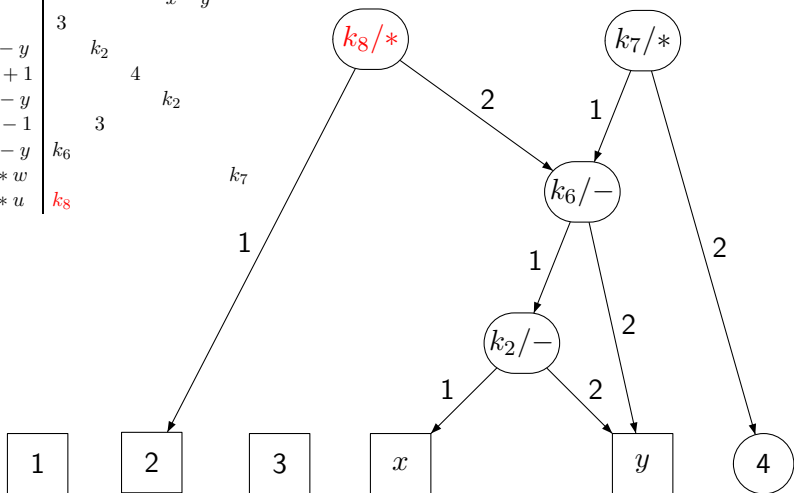
DAG-Construction

i	α_i	u	v	w	x	y	z
0					x	y	
1	$u \leftarrow 3$	3					
2	$v \leftarrow x - y$		k_2				
3	$w \leftarrow u + 1$			4			
4	$x \leftarrow x - y$				k_2		
5	$v \leftarrow w - 1$		3				
6	$u \leftarrow x - y$	k_6					
7	$z \leftarrow u * w$						k_7
	$u \leftarrow 2 * u$						



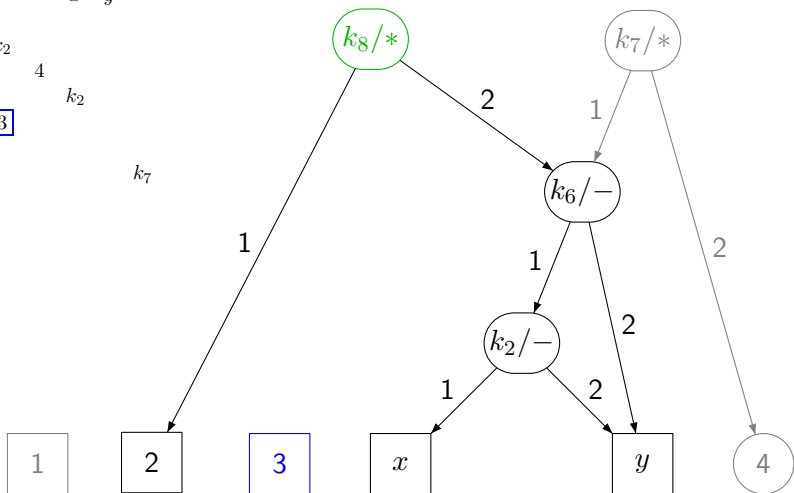
DAG-Construction

i	α_i	u	v	w	x	y	z
0					x	y	
1	$u \leftarrow 3$	3					
2	$v \leftarrow x - y$		k_2				
3	$w \leftarrow u + 1$			4			
4	$x \leftarrow x - y$				k_2		
5	$v \leftarrow w - 1$		3				
6	$u \leftarrow x - y$	k_6					
7	$z \leftarrow u * w$						
8	$u \leftarrow 2 * u$	k_8					



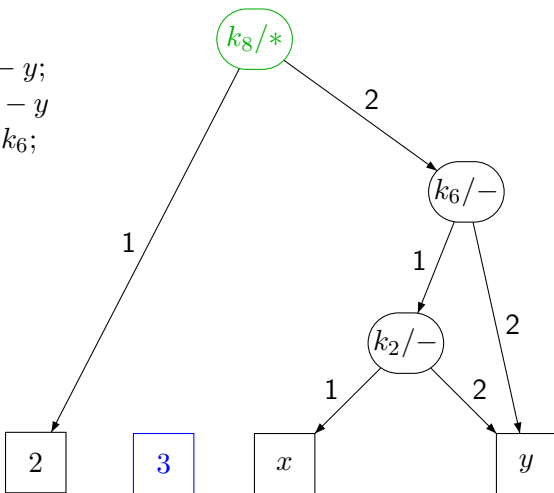
Determining Output-Relevant Nodes

i	u	v	w	x	y	z
0				x	y	
1	3					
2		k_2				
3			4			
4				k_2		
5						
6	k_6					
7					k_7	
8	k_8					



Code Generation from the DAG

$\vec{v}_{in} : (x, y)$
 $\beta' : k_2 \leftarrow x - y;$
 $k_6 \leftarrow k_2 - y$
 $u \leftarrow 2 * k_6;$
 $v \leftarrow 3;$
 $\vec{v}_{out} : (u, v)$



Optimality

T_{DAG} is optimal wrt. all classical transformations

Optimality

T_{DAG} is optimal wrt. all classical transformations

Backward direction valid for composition of classical algorithms?

$$\vec{v}_{in} : (x, y)$$

$$\beta : \boxed{u} \leftarrow x;$$

$$y \leftarrow \boxed{u} + y;$$

$$\boxed{v} \leftarrow u;$$

$$u \leftarrow \boxed{v} * y;$$

$$v \leftarrow u + \boxed{v};$$

$$\vec{v}_{out} : (u, v)$$

Optimality

T_{DAG} is optimal wrt. all classical transformations

Backward direction valid for composition of classical algorithms?

$$\begin{array}{l}
 \vec{v}_{in} : (x, y) \\
 \beta : \boxed{u} \leftarrow x; \\
 \quad y \leftarrow \boxed{u} + y; \\
 \quad \boxed{v} \leftarrow u; \\
 \quad u \leftarrow \boxed{v} * y; \\
 \quad v \leftarrow u + \boxed{v}; \\
 \vec{v}_{out} : (u, v)
 \end{array}
 \xrightarrow{T_{DAG}}
 \begin{array}{l}
 k_2 \leftarrow x + y; \\
 u \leftarrow x * k_2; \\
 v \leftarrow u + x;
 \end{array}$$

Optimality

T_{DAG} is optimal wrt. all classical transformations

Backward direction valid for composition of classical algorithms?

$\vec{v}_{in} : (x, y)$

$\beta : \boxed{u} \leftarrow x;$

$y \leftarrow \boxed{u} + y;$

$\boxed{v} \leftarrow u;$

$u \leftarrow \boxed{v} * y;$

$v \leftarrow u + \boxed{v};$

$\vec{v}_{out} : (u, v)$

T_{DAG}
→

$k_2 \leftarrow x + y;$

$u \leftarrow x * k_2;$

$v \leftarrow u + x;$

⇒ **No!**

⇒ Introduction of “preprocessing step”

Copy Propagation

i	α_i	$CP_i \subseteq V_\pi \times V_\pi \times \mathbb{N}$
1	$u \leftarrow x;$	\emptyset
2	$y \leftarrow u + y;$	$\{(u, x, 1)\}$
3	$v \leftarrow u;$	$\{(u, x, 1)\}$
4	$\boxed{u} \leftarrow v * y;$	$\{(\boxed{u}, x, 1), (v, \boxed{u}, 1), (v, x, 2)\}$
5	$v \leftarrow u + v$	$\{(v, x, 2)\}$

Copy Propagation

i	α_i	$CP_i \subseteq V_\pi \times V_\pi \times \mathbb{N}$	New Instr. α'_i
1	$u \leftarrow x;$	\emptyset	$u \leftarrow x;$
2	$y \leftarrow u + y;$	$\{(u, x, 1)\}$	$y \leftarrow x + y;$
3	$v \leftarrow u;$	$\{(u, x, 1)\}$	$v \leftarrow x;$
4	$\boxed{u} \leftarrow v * y;$	$\{(\boxed{u}, x, 1), (v, \boxed{u}, 1), (v, x, 2)\}$	$u \leftarrow x * y;$
5	$v \leftarrow u + v$	$\{(v, x, 2)\}$	$v \leftarrow u + x;$

Copy Propagation

i	α_i	$CP_i \subseteq V_\pi \times V_\pi \times \mathbb{N}$	New Instr. α'_i
1	$u \leftarrow x;$	\emptyset	$u \leftarrow x;$
2	$y \leftarrow u + y;$	$\{(u, x, 1)\}$	$y \leftarrow x + y;$
3	$v \leftarrow u;$	$\{(u, x, 1)\}$	$v \leftarrow x;$
4	$\boxed{u} \leftarrow v * y;$	$\{(\boxed{u}, x, 1), (v, \boxed{u}, 1), (v, x, 2)\}$	$u \leftarrow x * y;$
5	$v \leftarrow u + v$	$\{(v, x, 2)\}$	$v \leftarrow u + x;$

- To achieve determinism the **transitive depth** is important at instruction 4.

Copy Propagation

i	α_i	$CP_i \subseteq V_\pi \times V_\pi \times \mathbb{N}$	New Instr. α'_i
1	$u \leftarrow x;$	\emptyset	$u \leftarrow x;$
2	$y \leftarrow u + y;$	$\{(u, x, 1)\}$	$y \leftarrow x + y;$
3	$v \leftarrow u;$	$\{(u, x, 1)\}$	$v \leftarrow x;$
4	$\boxed{u} \leftarrow v * y;$	$\{(\boxed{u}, x, 1), (v, \boxed{u}, 1), (v, x, 2)\}$	$u \leftarrow x * y;$
5	$v \leftarrow u + v$	$\{(v, x, 2)\}$	$v \leftarrow u + x;$

- To achieve determinism the **transitive depth** is important at instruction 4.
- Final Dead-Code Elimination removes instructions 1 and 3.

Overview

- 1 Introduction
- 2 Classical Algorithms
- 3 DAG-Optimization
- 4 Composition of the Transformations**
- 5 Conclusion/Future Work

Enabling Relation

Definition

$$T_1 \rightarrow T_2 \Leftrightarrow \exists \pi \in \mathcal{SLC} \text{ with } T_1(\pi) = \pi \wedge T_1(T_2(\pi)) \neq T_2(\pi)$$

Enabling Relation

Definition

$$T_1 \rightarrow T_2 \Leftrightarrow \exists \pi \in \mathcal{SLC} \text{ with } T_1(\pi) = \pi \wedge T_1(T_2(\pi)) \neq T_2(\pi)$$

	T_{DC}	T_{CS}	T_{CF}	T_{CP}
T_{DC}	-	-	-	-
T_{CS}	-		-	→
T_{CF}	→	→		-
T_{CP}	→	→	-	

 $\vec{v}_{in} : (x)$
 $\beta : y \leftarrow f(x, x);$
 $z \leftarrow f(x, x);$
 $z \leftarrow f(y, z);$
 $\vec{v}_{out} : (y, z)$
 $\vec{v}_{in} : (x)$
 $\beta : t \leftarrow a;$
 $y \leftarrow f(x, a);$
 $z \leftarrow f(x, t);$
 $\vec{v}_{out} : (y, z)$
 $\vec{v}_{in} : (x)$
 $\beta : t \leftarrow x;$
 $y \leftarrow g(x);$
 $z \leftarrow g(t);$
 $\vec{v}_{out} : (y, z)$

Mutual Dependency of T_{CS} and T_{CP}

Is there a fixed constant $c \in \mathbb{N}$, such that c applications of $T_{CP} \circ T_{CS}$ suffice?

Mutual Dependency of T_{CS} and T_{CP}

Is there a fixed constant $c \in \mathbb{N}$, such that c applications of $T_{CP} \circ T_{CS}$ suffice?

n -fold application of T_{CP} and T_{CS} necessary

$\pi_n := ((\{f^{(2)}\}, \emptyset), (x), (y, z), \beta_n)$ with β_n as follows:

$$\beta_1 := y \leftarrow f(x, x);$$

$$z \leftarrow f(x, x);$$

$$\beta_{n+1} := \beta_n;$$

$$y \leftarrow f(y, x);$$

$$z \leftarrow f(z, x);$$

Example Computation

$y \leftarrow f(x, x);$

$z \leftarrow f(x, x);$

$y \leftarrow f(y, x);$

$z \leftarrow f(z, x);$

$y \leftarrow f(y, x);$

$z \leftarrow f(z, x);$

\vdots

Example Computation

$y \leftarrow f(x, x);$	$t_1 \leftarrow f(x, x);$
$z \leftarrow f(x, x);$	$y \leftarrow t_1;$
$y \leftarrow f(y, x);$	$z \leftarrow t_1;$
$z \leftarrow f(z, x);$	$y \leftarrow f(y, x);$
$y \leftarrow f(y, x);$	$z \leftarrow f(z, x);$
$z \leftarrow f(z, x);$	$y \leftarrow f(y, x);$
\vdots	$z \leftarrow f(z, x);$
	\vdots

Example Computation

$y \leftarrow f(x, x);$	$t_1 \leftarrow f(x, x);$	$t_1 \leftarrow f(x, x);$
$z \leftarrow f(x, x);$	$y \leftarrow t_1;$	$y \leftarrow t_1;$
$y \leftarrow f(y, x);$	$z \leftarrow t_1;$	$z \leftarrow t_1;$
$z \leftarrow f(z, x);$	$y \leftarrow f(y, x);$	$y \leftarrow f(t_1, x);$
$y \leftarrow f(y, x);$	$z \leftarrow f(z, x);$	$z \leftarrow f(t_1, x);$
$z \leftarrow f(z, x);$	$y \leftarrow f(y, x);$	$y \leftarrow f(y, x);$
\vdots	$z \leftarrow f(z, x);$	$z \leftarrow f(z, x);$
	\vdots	\vdots

Example Computation

```

y ← f(x, x);
z ← f(x, x);
y ← f(y, x);
z ← f(z, x);
y ← f(y, x);
z ← f(z, x);
⋮

```

```

t1 ← f(x, x);
y ← t1;
z ← t1;
y ← f(y, x);
z ← f(z, x);
y ← f(y, x);
z ← f(z, x);
⋮

```

```

t1 ← f(x, x);
y ← t1;
z ← t1;
y ← f(t1, x);
z ← f(t1, x);
y ← f(y, x);
z ← f(z, x);
⋮

```

```

t1 ← f(x, x);
y ← t1;
z ← t1;
t4 ← f(t1, x);
y ← t4;
z ← t4;
y ← f(y, x);
z ← f(z, x);
⋮

```

Fixed Point Iteration

For $T := T_{CP} \circ T_{CS}$, $T_{CPCS} : \mathcal{SLC} \rightarrow \mathcal{SLC}$ is given by:

$$T_{CPCS}(\pi) := \begin{cases} \pi & \text{if } T(\pi) = \pi \\ T_{CPCS}(T(\pi)) & \text{otherwise} \end{cases}$$

At most as many iterations as operation expressions to reach the fixed point.

Fixed Point Iteration

For $T := T_{CP} \circ T_{CS}$, $T_{CPCS} : \mathcal{SLC} \rightarrow \mathcal{SLC}$ is given by:

$$T_{CPCS}(\pi) := \begin{cases} \pi & \text{if } T(\pi) = \pi \\ T_{CPCS}(T(\pi)) & \text{otherwise} \end{cases}$$

At most as many iterations as operation expressions to reach the fixed point.

Compositional Transformation

$$T_{COPT} := T_{DC} \circ T_{CPCS} \circ T_{CF} = T_{DC} \circ (T_{CP} \circ T_{CS})^* \circ T_{CF}$$

Example T_{COPT}

 $\vec{v}_{in} : (x, y)$ $\beta : u \leftarrow 3;$ $v \leftarrow x - y;$ $w \leftarrow u + 1;$ $x \leftarrow x - y;$ $v \leftarrow w - 1;$ $u \leftarrow x - y;$ $z \leftarrow u * w;$ $u \leftarrow 2 * u;$ $\vec{v}_{out} : (u, v)$

Example T_{COPT} $\vec{v}_{in} : (x, y)$ $\beta : u \leftarrow 3;$ $v \leftarrow x - y;$ $w \leftarrow u + 1;$ $x \leftarrow x - y;$ $v \leftarrow w - 1;$ $u \leftarrow x - y;$ $z \leftarrow u * w;$ $u \leftarrow 2 * u;$ $\vec{v}_{out} : (u, v)$ T_{COPT}
→ $t_2 \leftarrow x - y;$ $v \leftarrow 3;$ $u \leftarrow t_2 - y;$ $u \leftarrow 2 * u;$

Example T_{COPT}

 $\vec{v}_{in} : (x, y)$
 $\beta : u \leftarrow 3;$
 $v \leftarrow x - y;$
 $w \leftarrow u + 1;$
 $x \leftarrow x - y;$
 $v \leftarrow w - 1;$
 $u \leftarrow x - y;$
 $z \leftarrow u * w;$
 $u \leftarrow 2 * u;$
 $\vec{v}_{out} : (u, v)$
 T_{COPT}
 \longrightarrow
 $t_2 \leftarrow x - y;$
 $v \leftarrow 3;$
 $u \leftarrow t_2 - y;$
 $u \leftarrow 2 * u;$
 T_{DAG}
 \longrightarrow
 $k_2 \leftarrow x - y;$
 $k_6 \leftarrow k_2 - y;$
 $u \leftarrow 2 * k_6;$
 $v \leftarrow 3;$

Example T_{COPT}

$$\vec{v}_{in} : (x, y)$$

$$\beta : u \leftarrow 3;$$

$$v \leftarrow x - y;$$

$$w \leftarrow u + 1;$$

$$x \leftarrow x - y;$$

$$v \leftarrow w - 1;$$

$$u \leftarrow x - y;$$

$$z \leftarrow u * w;$$

$$u \leftarrow 2 * u;$$

$$\vec{v}_{out} : (u, v)$$

$$\xrightarrow{T_{COPT}}$$

$$t_2 \leftarrow x - y;$$

$$v \leftarrow 3;$$

$$u \leftarrow t_2 - y;$$

$$u \leftarrow 2 * u;$$

$$\xrightarrow{T_{DAG}}$$

$$k_2 \leftarrow x - y;$$

$$k_6 \leftarrow k_2 - y;$$

$$u \leftarrow 2 * k_6;$$

$$v \leftarrow 3;$$

Minor Problems

- Order of operations not identical
- Variable names not consistent

Result

 T_{XOPT}

$$T_{XOPT} = T_{SSA} \circ T_{RC} \circ T_{DC} \circ (T_{CP} \circ T_{CS})^* \circ T_{CF} \circ T_{SSA}$$

Result

 T_{XOPT}

$$T_{XOPT} = T_{SSA} \circ T_{RC} \circ T_{DC} \circ (T_{CP} \circ T_{CS})^* \circ T_{CF} \circ T_{SSA}$$

Optimality

T_{XOPT} is optimal wrt. T_{DAG} .

(The backward direction holds since T_{DAG} is optimal wrt. the single transformations.)

→ Proof in the corresponding technical report

Overview

- 1 Introduction
- 2 Classical Algorithms
- 3 DAG-Optimization
- 4 Composition of the Transformations
- 5 Conclusion/Future Work

Conclusion

- Classical algorithms do not suffice to “simulate” T_{DAG}
- Essential component: $T_{DC} \circ (T_{CP} \circ T_{CS})^* \circ T_{CF}$
- Higher computation overhead, but:
 - DAG-algorithm not applicable to iterative code
 - Many implementations of classical algorithms exist

Conclusion

- Classical algorithms do not suffice to “simulate” T_{DAG}
- Essential component: $T_{DC} \circ (T_{CP} \circ T_{CS})^* \circ T_{CF}$
- Higher computation overhead, but:
 - DAG-algorithm not applicable to iterative code
 - Many implementations of classical algorithms exist

Technical Report

Th. Noll, S. Rieger: Optimization of Straight-Line Code Revisited
Aachener Informatik-Bericht 2005–21, Nov. 2005

aib.informatik.rwth-aachen.de/2005/2005-21.pdf

Conclusion

- Classical algorithms do not suffice to “simulate” T_{DAG}
- Essential component: $T_{DC} \circ (T_{CP} \circ T_{CS})^* \circ T_{CF}$
- Higher computation overhead, but:
 - DAG-algorithm not applicable to iterative code
 - Many implementations of classical algorithms exist

Technical Report

Th. Noll, S. Rieger: Optimization of Straight-Line Code Revisited
Aachener Informatik-Bericht 2005-21, Nov. 2005

aib.informatik.rwth-aachen.de/2005/2005-21.pdf

Webinterface

Implementation for testing purposes: slc.srieger.com

Future Work

- Combination of T_{CS} and T_{CP} possible to avoid iterative application?
- Application of framework on iterative code
- Evaluation of optimization quality on iterative code
- ...

Composing Transformations to Optimize Linear Code

Thomas Noll Stefan Rieger

MOVES: Software Modeling and Verification
RWTH Aachen University, Germany

28.9.2007



Some copy instructions cannot be eliminated

π	$T_{COPT}(\pi)$	$T_{DAG}(T_{COPT}(\pi))$
$y \leftarrow f(x, x);$ $z \leftarrow f(x, x);$	$t_1 \leftarrow f(x, x);$ $y \leftarrow t_1;$ $z \leftarrow t_1;$	$y \leftarrow f(x, x);$ $z \leftarrow y;$

Some copy instructions cannot be eliminated

π	$T_{COPT}(\pi)$	$T_{DAG}(T_{COPT}(\pi))$
$y \leftarrow f(x, x);$ $z \leftarrow f(x, x);$	$t_1 \leftarrow f(x, x);$ $y \leftarrow t_1;$ $z \leftarrow t_1;$	$y \leftarrow f(x, x);$ $z \leftarrow y;$

⇒ specialized Algorithm [see techn. report]

Some copy instructions cannot be eliminated

π	$T_{COPT}(\pi)$	$T_{DAG}(T_{COPT}(\pi))$
	$t_1 \leftarrow f(x, x);$	
$y \leftarrow f(x, x);$	$y \leftarrow t_1;$	$y \leftarrow f(x, x);$
$z \leftarrow f(x, x);$	$z \leftarrow t_1;$	$z \leftarrow y;$

⇒ specialized Algorithm [see techn. report]

Another problem

π	CP_i
$t \leftarrow x;$	\emptyset
$x \leftarrow f(x);$	$\{(t, x, 1)\}$
$y \leftarrow t;$	\emptyset
$z \leftarrow g(x);$	$\{(y, t, 1)\}$

Some copy instructions cannot be eliminated

π	$T_{COPT}(\pi)$	$T_{DAG}(T_{COPT}(\pi))$
	$t_1 \leftarrow f(x, x);$	
$y \leftarrow f(x, x);$	$y \leftarrow t_1;$	$y \leftarrow f(x, x);$
$z \leftarrow f(x, x);$	$z \leftarrow t_1;$	$z \leftarrow y;$

⇒ specialized Algorithm [see techn. report]

Another problem

π	CP_i
$t \leftarrow x;$	\emptyset
$x \leftarrow f(x);$	$\{(t, x, 1)\}$
$y \leftarrow t;$	\emptyset
$z \leftarrow g(x);$	$\{(y, t, 1)\}$

⇒ Transformation into Static Single Assignment-Form