Peephole Optimization

Simon Oehrl and Bao-Loc Nguyen Ngo

21. Juli 2014

Table of Contents

1 Introduction
- Context and Motivation

2 Peephole Optimization
- General
- Replacement Rules
- Pattern Matching

3 Demo

4 Replacement Rules Derivation
- Finding replacement Rules
- Automatic rules derivation
- Validation of Equivalence
- Cost Comparison

5 Conclusion
- References
- Peephole Optimization in LLVM

RWTH AACHEN UNIVERSITY

Code Optimization

- Improve code quality and efficiency
  - Increase Performance
  - Reduce code size
- Optimization on different abstraction levels
  - High-level programming language
  - Intermediate code
  - Machine code
- Optimization techniques
  - Local optimization
  - Global optimization
  - **Peephole optimization**
  - ...

Code Optimization

- Improve code quality and efficiency
  - Increase Performance
  - Reduce code size
- Optimization on different abstraction levels
  - High-level programming language
  - Intermediate code
  - Machine code
- Optimization techniques
  - Local optimization
  - Global optimization
  - **Peephole optimization**
  - ...

RWTH AACHEN UNIVERSITY

Code Optimization

- Improve code quality and efficiency
  - Increase Performance
  - Reduce code size
- Optimization on different abstraction levels
  - High-level programming language
  - Intermediate code
  - Machine code
- Optimization techniques
  - Local optimization
  - Global optimization
  - **Peephole optimization**
  - ...

Table of Contents

1. Introduction
   - Context and Motivation
2. Peephole Optimization
   - General
   - Replacement Rules
   - Pattern Matching
3. Demo
4. Replacement Rules Derivation
   - Finding replacement Rules
   - Automatic rules derivation
   - Validation of Equivalence
   - Cost Comparison
5. Conclusion
   - References
   - Peephole Optimization in LLVM

Peephole Optimization

Usage Context

- Used within modern compiler (e.g. GCC, LLVM, ACK, . . .)
- Typically applied on intermediate or machine code

Optimization precedure

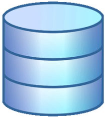
- Optimize code in a small moving window (peephole)
- Iterative Optimazation

Peephole Optimization

```
; <label>:42
  store double 0.000000e+00, double* %det, align 8
  store i32 0, i32* %j1, align 4
  br label %43

; <label>:43
  %44 = load i32* %j1, align 4
  %45 = load i32* %2, align 4
  %46 = icmp slt i32 %44, %45
  br i1 %46, label %47, label %157

; <label>:47
  %48 = load i32* %2, align 4
  %49 = sub nsw i32 %48, 1
  %50 = sext i32 %49 to i64
  %51 = mul i64 %50, 8
  %52 = call i8* @malloc(i64 %51)
  %53 = bitcast i8* %52 to double**
  store double** %53, double*** %m, align 8
  store i32 0, i32* %i, align 4
  br label %54

; <label>:54
  %55 = load i32* %i, align 4
  %56 = load i32* %2, align 4
  %57 = sub nsw i32 %56, 1
  %58 = icmp slt i32 %55, %57
  br i1 %58, label %59, label %73

; <label>:59
  %60 = load i32* %2, align 4
  %61 = sub nsw i32 %60, 1
  %62 = sext i32 %61 to i64
  %63 = mul i64 %62, 8
  %64 = call i8* @malloc(i64 %63)
```

no match

Rule Database



```
   %0 = mul i32 %1, 8
=> %0 = lsh i32 %1, 3
```

RWTH AACHEN UNIVERSITY

Peephole Optimization

```
; <label>:42
  store double 0.000000e+00, double* %det, align 8
  store i32 0, i32* %j1, align 4
  br label %43

; <label>:43
  %44 = load i32* %j1, align 4
  %45 = load i32* %2, align 4
  %46 = icmp slt i32 %44, %45
  br i1 %46, label %47, label %157

; <label>:47
  %48 = load i32* %2, align 4
  %49 = sub nsw i32 %48, 1
  %50 = sext i32 %49 to i64
  %51 = mul i64 %50, 8
  %52 = call i8* @malloc(i64 %51)
  %53 = bitcast i8* %52 to double**
  store double** %53, double*** %m, align 8
  store i32 0, i32* %i, align 4
  br label %54

; <label>:54
  %55 = load i32* %i, align 4
  %56 = load i32* %2, align 4
  %57 = sub nsw i32 %56, 1
  %58 = icmp slt i32 %55, %57
  br i1 %58, label %59, label %73

; <label>:59
  %60 = load i32* %2, align 4
  %61 = sub nsw i32 %60, 1
  %62 = sext i32 %61 to i64
  %63 = mul i64 %62, 8
  %64 = call i8* @malloc(i64 %63)
```

match

Rule Database

```
   %0 = mul i32 %1, 8
=> %0 = lsh i32 %1, 3
```

Peephole Optimization

```
; <label>:42
  store double 0.000000e+00, double* %det, align 8
  store i32 0, i32* %j1, align 4
  br label %43

; <label>:43
  %44 = load i32* %j1, align 4
  %45 = load i32* %2, align 4
  %46 = icmp slt i32 %44, %45
  br i1 %46, label %47, label %157

; <label>:47
  %48 = load i32* %2, align 4
  %49 = sub nsw i32 %48, 1
  %50 = sext i32 %49 to i64
  %51 = shl i64 %50, 3                          replace
  %52 = call i8* @malloc(i64 %51)
  %53 = bitcast i8* %52 to double**
  store double** %53, double*** %m, align 8
  store i32 0, i32* %i, align 4
  br label %54

; <label>:54
  %55 = load i32* %i, align 4
  %56 = load i32* %2, align 4
  %57 = sub nsw i32 %56, 1
  %58 = icmp slt i32 %55, %57
  br i1 %58, label %59, label %73

; <label>:59
  %60 = load i32* %2, align 4
  %61 = sub nsw i32 %60, 1
  %62 = sext i32 %61 to i64
  %63 = mul i64 %62, 8
  %64 = call i8* @malloc(i64 %63)
```

Rule Database

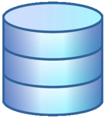
```
   %0 = mul i32 %1, 8
=> %0 = lsh i32 %1, 3
```

Peephole Optimization

```
; <label>:42
  store double 0.000000e+00, double* %det, align 8
  store i32 0, i32* %j1, align 4
  br label %43

; <label>:43
  %44 = load i32* %j1, align 4
  %45 = load i32* %2, align 4
  %46 = icmp slt i32 %44, %45
  br i1 %46, label %47, label %157

; <label>:47
  %48 = load i32* %2, align 4
  %49 = sub nsw i32 %48, 1
  %50 = sext i32 %49 to i64
  %51 = mul i64 %50, 8
  %52 = call i8* @malloc(i64 %51)
  %53 = bitcast i8* %52 to double**
  store double** %53, double*** %m, align 8
  store i32 0, i32* %i, align 4
  br label %54

; <label>:54
  %55 = load i32* %i, align 4
  %56 = load i32* %2, align 4
  %57 = sub nsw i32 %56, 1
  %58 = icmp slt i32 %55, %57
  br i1 %58, label %59, label %73

; <label>:59
  %60 = load i32* %2, align 4
  %61 = sub nsw i32 %60, 1
  %62 = sext i32 %61 to i64
  %63 = mul i64 %62, 8
  %64 = call i8* @malloc(i64 %63)
```

no match

Rule Database



```
   %0 = mul i32 %1, 8
=> %0 = lsh i32 %1, 3
```

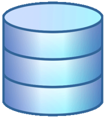
RWTH AACHEN UNIVERSITY

Peephole Optimization

```
; <label>:42
  store double 0.000000e+00, double* %det, align 8
  store i32 0, i32* %j1, align 4
  br label %43

; <label>:43
  %44 = load i32* %j1, align 4
  %45 = load i32* %2, align 4
  %46 = icmp slt i32 %44, %45
  br i1 %46, label %47, label %157

; <label>:47
  %48 = load i32* %2, align 4
  %49 = sub nsw i32 %48, 1
  %50 = sext i32 %49 to i64
  %51 = mul i64 %50, 8
  %52 = call i8* @malloc(i64 %51)
  %53 = bitcast i8* %52 to double**
  store double** %53, double*** %m, align 8
  store i32 0, i32* %i, align 4
  br label %54

; <label>:54
  %55 = load i32* %i, align 4
  %56 = load i32* %2, align 4
  %57 = sub nsw i32 %56, 1
  %58 = icmp slt i32 %55, %57
  br i1 %58, label %59, label %73

; <label>:59
  %60 = load i32* %2, align 4
  %61 = sub nsw i32 %60, 1
  %62 = sext i32 %61 to i64
  %63 = mul i64 %62, 8
  %64 = call i8* @malloc(i64 %63)
```

no match

Rule Database



```
   %0 = mul i32 %1, 8
=> %0 = lsh i32 %1, 3
```

Table of Contents

1 Introduction
   - Context and Motivation

2 Peephole Optimization
   - General
   - Replacement Rules
   - Pattern Matching

3 Demo

4 Replacement Rules Derivation
   - Finding replacement Rules
   - Automatic rules derivation
   - Validation of Equivalence
   - Cost Comparison

5 Conclusion
   - References
   - Peephole Optimization in LLVM

Replacement Rules

- Replace Sequence of Instruction with faster and/or fewer Instructions
- Many different Replacement Techniques

Common Techniques

Null Sequences

- Remove useless instructions

Example

LOV A
LOC 0
ADD

Common Techniques

Null Sequences

- Remove useless instructions

Example

LOV A		LOV A
LOC 0	$\rightarrow$	
ADD		

Common Techniques

Constant folding

- Replace constant expressions with its result

Example

LOC 8
LOC 2
ADD

Common Techniques

Constant folding

- Replace constant expressions with its result

Example

LOC 8		LOC 10
LOC 2	$\rightarrow$	
ADD		

Common Techniques

Algebraic Laws

- Use algebraic laws to simplify expressions

Example

LOV A
LOC 7
ADD
LOC 5
SUB

$(A + 7) - 5$

RWTH AACHEN UNIVERSITY

Common Techniques

Algebraic Laws

- Use algebraic laws to simplify expressions

Example

LOV A		LOV A
LOC 7		LOC 7
ADD	$\rightarrow$	LOC 5
LOC 5		SUB
SUB		ADD
$(A+7)-5$	$=$	$A+(7-5)$

Common Techniques

Strength reduction

- Replace stronger operations with weaker ones
- Often applicable to multiplication or exponentiation in loops

Example

LOV A
LOC 2
MUL

Common Techniques

Strength reduction

- Replace stronger operations with weaker ones
- Often applicable to multiplication or exponentiation in loops

Example

LOV A		LOV A
LOC 2	$\rightarrow$	DUP 2
MUL		ADD

Common Techniques

Strength reduction

- Replace stronger operations with weaker ones
- Often applicable to multiplication or exponentiation in loops

Example

LOV A		LOV A		LOV A
LOC 2	$\rightarrow$	DUP 2	$\rightarrow$	LOC 1
MUL		ADD		SHL

Common Techniques

Special instructions

- Replace Instructions with specialized Instruction
- Usually more relevant on machine code

Example

LOV A
LOC 1
ADD
STV A

Common Techniques

Special instructions

- Replace Instructions with specialized Instruction
- Usually more relevant on machine code

Example

LOV A LOC 1 ADD STV A	$\rightarrow$	INV A

Table of Contents

1. Introduction
   - Context and Motivation
2. Peephole Optimization
   - General
   - Replacement Rules
   - Pattern Matching
3. Demo
4. Replacement Rules Derivation
   - Finding replacement Rules
   - Automatic rules derivation
   - Validation of Equivalence
   - Cost Comparison
5. Conclusion
   - References
   - Peephole Optimization in LLVM

Pattern Matching

- Many instruction sequences have same semantics
- Generalize rules to avoid (semantically) identical rules
- Different kinds of matching strategies

Example

LOC 7		LOC 7
ADD	$\rightarrow$	LOC 5
LOC 5		SUB
SUB		ADD

Pattern Matching

- Many instruction sequences have same semantics
- Generalize rules to avoid (semantically) identical rules
- Different kinds of matching strategies

Example

LOC 7		LOC 7	LOC 6		LOC 6
ADD	$\rightarrow$	LOC 5	ADD	$\rightarrow$	LOC 5
LOC 5		SUB	LOC 5		SUB
SUB		ADD	SUB		ADD

Pattern Matching

- Many instruction sequences have same semantics
- Generalize rules to avoid (semantically) identical rules
- Different kinds of matching strategies

Example

LOC 7		LOC 7	LOC 6		LOC 6	
ADD	$\rightarrow$	LOC 5	ADD	$\rightarrow$	LOC 5	
LOC 5		SUB	LOC 5		SUB	
SUB		ADD	SUB		ADD	...

Pattern Matching

- Many instruction sequences have same semantics
- Generalize rules to avoid (semantically) identical rules
- Different kinds of matching strategies

Example

LOC %0		LOC %0
ADD		LOC %1
LOC %1	→	SUB
SUB		ADD

RWTH AACHEN UNIVERSITY

Demo

Demo

RWTH AACHEN UNIVERSITY

Table of Contents

1. Introduction
   - Context and Motivation
2. Peephole Optimization
   - General
   - Replacement Rules
   - Pattern Matching
3. Demo
4. Replacement Rules Derivation
   - Finding replacement Rules
   - Automatic rules derivation
   - Validation of Equivalence
   - Cost Comparison
5. Conclusion
   - References
   - Peephole Optimization in LLVM

How to find replacement rules?

Manual rule creation

- Replacement rules written by hand
- Using common techniques

Automatic rules derivation

- Deriving rules using superoptimization technique

Table of Contents

1 Introduction
   - Context and Motivation

2 Peephole Optimization
   - General
   - Replacement Rules
   - Pattern Matching

3 Demo

4 Replacement Rules Derivation
   - Finding replacement Rules
   - Automatic rules derivation
   - Validation of Equivalence
   - Cost Comparison

5 Conclusion
   - References
   - Peephole Optimization in LLVM

RWTH AACHEN UNIVERSITY

Automatic rules derivation

Superoptimization

- Given a set of instruction sequences to optimize
- Calculate every possible instruction sequence out of a set of instructions
- Compare outcome of calculated sequences to original sequences
- Check if new found equivalent sequence has lower cost

Automatic rules derivation

Superoptimization

- Given a set of instruction sequences to optimize
- Calculate every possible instruction sequence out of a set of instructions
- Compare outcome of calculated sequences to original sequences
- Check if new found equivalent sequence has lower cost

Automatic rules derivation

Superoptimization

- Given a set of instruction sequences to optimize
- Calculate every possible instruction sequence out of a set of instructions
- Compare outcome of calculated sequences to original sequences
- Check if new found equivalent sequence has lower cost

Automatic rules derivation

Superoptimization

- Given a set of instruction sequences to optimize
- Calculate every possible instruction sequence out of a set of instructions
- Compare outcome of calculated sequences to original sequences
- Check if new found equivalent sequence has lower cost

Table of Contents

1 Introduction
   - Context and Motivation

2 Peephole Optimization
   - General
   - Replacement Rules
   - Pattern Matching

3 Demo

4 Replacement Rules Derivation
   - Finding replacement Rules
   - Automatic rules derivation
   - Validation of Equivalence
   - Cost Comparison

5 Conclusion
   - References
   - Peephole Optimization in LLVM

RWTH AACHEN UNIVERSITY

Equivalence Test

Execution Test: Process

- Run both instruction sequences on machine
- Use multiple vectors as input
- Compare outcome
- Discard sequences with not matching results

Execution Test: Property

- Stochastical test
- Accuracy depends on test effort

Execution Test

Example

Sequence	
Original	Replacement
LOV X	LOV X
LOV Y	LOV Y
ADD	SUB

Input Vector (X,Y)	Stack Original	Stack Replacement
(0,0)	[0]	[0] ✓
(1,1)	[2]	[0] ✗

RWTH AACHEN UNIVERSITY

Execution Test

Example

Sequence

Original	Replacement
LOV X	LOV X
LOC 2	SHR
DIV	SHL
LOC 2	
MUL	

Input Vector (X)	Stack Original	Stack Replacement
(0)	[0]	[0] ✓
(1)	[0]	[0] ✓
(65535)	[65534]	[65534] ✓

Table of Contents

1 Introduction
- Context and Motivation

2 Peephole Optimization
- General
- Replacement Rules
- Pattern Matching

3 Demo

4 Replacement Rules Derivation
- Finding replacement Rules
- Automatic rules derivation
- Validation of Equivalence
- Cost Comparison

5 Conclusion
- References
- Peephole Optimization in LLVM

Cost Comparison

Cost Functions

- Runtime
- Code size
- Memory usage

Conclusion

- Optimization technique used in modern compilers
- Improve code quality and efficiency
- Using replacement rules for optimization
- Generalize with patterns
- Automatic generation of rules possible
  - Generated optimizer speedup of 1.7 to factor of 10 over conventional optimizer

RWTH AACHEN UNIVERSITY

References

- Tanenbaum, Andrew S. and van Staveren, Hans and Stevenson, Johan W. Using Peephole Optimization on Intermediate Code. ACM Trans. Program. Lang. Syst., pp. 21-36
- Bansal, Sorav and Aiken, Alex. Automatic Generation of Peephole Superoptimizers. IGARCH Comput. Archit. News.
- Elif Aktolga and Supervisor Dr. Des Watson. Pattern Matching Strategies for Peephole Optimisation
- Henry Massalin. Department of Computer Science Columbia University New York, NY 10027. Superoptimizer - A Look at the Smallest Program

Peephole Optimization in LLVM

Example

```
int vadd(int a)
{
        return a*2
}
```

llvm IR code

```
%a.addr = alloca i32, align 4
store i32 %a, i32* %a.addr, align 4
%0 = load i32* %a.addr, align 4
%mul = mul nsw i32 %0, 2          →    %mul = shl i32 %0, 1
ret i32 %mul
```

Execution Test

Example

Original Sequence	Replacement Candidate
ld r0, x	ld r0, x
ld r1, y	ld r1, y
add r0, r1	

vector (x,y)	Original Sequence	Replacement Candidate
(0,0)	(0,0)	(0,0) ✓
(1,1)	(2,1)	(1,1) ✗

Boolean Test

- Generate boolean formula for machine states
  - Register
  - Memory
  - Stack
- Express each instruction as formula that changes machine state
- Express equivalence relation of final machine states as constraint
- Use SAT solver to check if constraint holds

Strength Reduction in Loops

- Remove multiplication from a loop
- Occurs often during array access

Example

```
for (I = 0, A = 0; I < 10; ++I) {
    A = I * 4;
    ...
}
...
```

Strength Reduction in Loops

- Remove multiplication from a loop
- Occurs often during array access

Example

```
for (I = 0, A = 0; I < 10; ++I) {
    ...
    A += 4;
}
...
```

Strength Reduction in Loops

Example

ZRV I		ZRV I
ZRV A		ZRV A
LOV I		LOV I
LOC 10		LOC 10
BGE 7		BGE 7
LOC 4		**...**
LOV I	$\rightarrow$	**LOC 4**
MUL		**LOV A**
STV A		**ADD**
...		**STV A**
INV I		INV I
BRA -8		BRA -8
...		...