

# Optimizing String manipulation performance

Markus Wondrak

Institute of Computer Science  
Johann Wolfgang von Goethe Universität, Frankfurt am Main

July 22, 2014

# Table of Contents

- 1 Motivation
- 2 Bytecode
- 3 WALA
- 4 Analysis
- 5 Transformation
- 6 Benchmarks
- 7 Conclusion

## Section 1

# Motivation

# Example

The "normal" way

```
String result = "";  
  
for (int i = 0; i < line.length(); i += 2) {  
    result += line.substring(i, i + 1);  
}  
  
return result;
```

# Example

## The optimized way

```
SubstringString lineOpt = new SubstringString(line);

StringListBuilder builder = new StringListBuilder();

for (int i = 0; i < line.length(); i+=2) {
    builder.append(lineOpt.substring(i, i + 1));
}

return builder.toString();
```

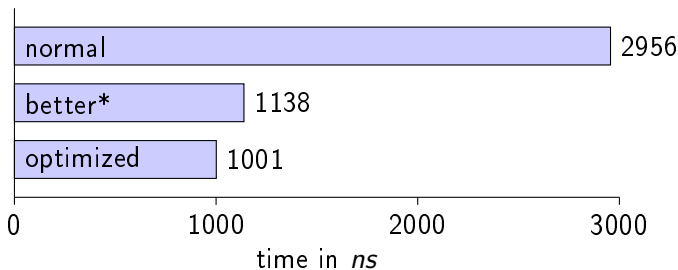
# What is the difference?

- Java strings are immutable → manipulation causes `char[]` copy
- '+' operator is compiled to a `StringBuilder`, **but** the loop is not recognized
- `StringBuilder` also array-based
- optimized types avoid this behavior

# What is the difference?

- optimized types avoid this behavior
- `SubstringString` returns only a new object pointing to the new boundaries
- `StringBuilder` is a linked list

# Measurement



(\*) with the use of `StringBuilder` around the loop



Wouldn't it be nice to have the performance of the optimized one with the readability of the normal one?

# Requirements

Given a method optimization definition, the system should ...

- ...be applicable to already compiled programs
- ...identify method calls in the Java bytecode
- ...replace these method calls by the optimized ones

## Section 2

# Bytecode

# Bytecode

- What the JVM actual executes (platform independence)
- Assembly language like
- Stack-based and imperative

# Bytecode

Java:

```
String x = "Hallo Welt";  
String y = x.substring(5);
```

Bytecode:

```
LDC "Hallo World!"  
ASTORE 1  
ALOAD 1  
ICONST 5  
INVOKEVIRTUAL java/lang/String.substring(I)Ljava/lang/  
    String;  
ASTORE 2
```

# WALA

- T.J. Watson Library of Analysis (IBM)
- static analysis for Java bytecode and Javascript
- open sourced at <http://github.com/wala/WALA> since 2006

# Features

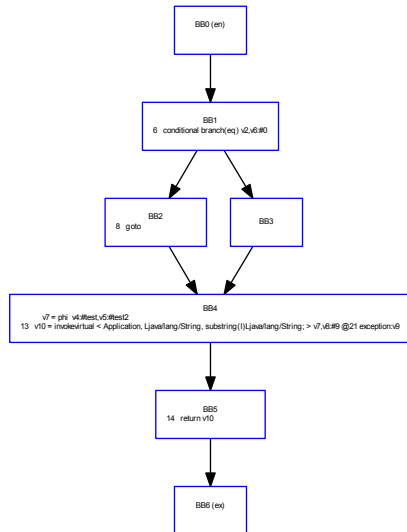
- Java type system and class hierarchy analysis
- supports frontends for Java and JavaScript
- SSA-based Intermediate Representation
- bytecode manipulation

# Intermediate Representation

- central data structure that represents the analyzed method
- abstracts the actual bytecode
- is in static single assignment form
- consists of a control-flow graph
- $\phi$ -nodes represent a merge of variables



```
String a = "test";  
String b = "test2";  
String c = ((is) ? a:b);  
  
return c.substring(9);
```



## Section 4

# Analysis

# Naming

value number

a variable in the IR

local

a local variable in the bytecode

label

a definition how certain method calls are identified and can be replaced

# Basic idea

- Create a dataflow graph of the value numbers in the IR
- determine a bubble in that graph by
  - label all affected method call instructions
  - inherit the labels to all connected value numbers and instructions, if possible

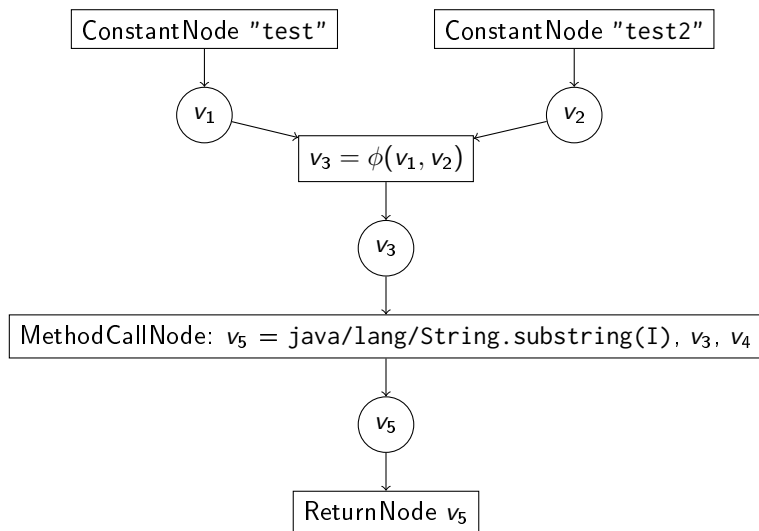
# How does the dataflow graph look like?

- directed graph based on the IR
- is composed of 2 kinds of nodes
  - Reference merely the value number ( $R$ )
  - InstructionNode can be seen as a instruction ( $I$ )
- for  $r \in R, i \in I$ ,
  - $(i, r)$  is called the definition of  $r$
  - $(r, i)$  is called a use of  $r$
- $\forall r \in R, in(r) = 1$ , so every  $r$  has exactly 1 definition, but  $n$  uses (SSA)
- $\forall i \in I, out(i) \leq 1$ , so every  $i$  can at most define one  $r$

# InstructionNodes

- ConstantNode a constant definition (e.g. "Hallo World")
- ParameterNode a parameter of the method
- MethodCallNode a method call (e.g.  $x.f(y)$ )
- ReturnNode a return instruction of the method
- PhiNode a  $\phi$  node in the IR

## example graph



# How to define a label?

From the interface `TypeLabel`:

```
boolean canBeUsedAsParamFor(MethodReference, int)  
boolean canBeUsedAsReceiverFor(MethodReference)  
boolean canBeDefinedAsResultOf(MethodReference)  
boolean canReturnedValueBeLabeled(MethodReference)  
boolean compatibleWith(TypeLabel)  
ReceiverInfo getReceiverUseInfo(MethodReference)
```



# How to deal with phis?

- $\phi$ -nodes just represent the merge of value numbers
- any label could be compatible with any  $\phi$  instruction
- so they were labeled after the analysis has taken place
- the decision is made by the count of labeled references connected to the particular phi

## Section 5

# Transformation

# What to do?

- Create conversation at the "bubbles" barriers
- replace the original method calls with the optimized ones
- to not overwrite the original values, create appropriate locals for the optimized ones

# local matrix

maxlocals are 6 and there are 2 labels ( $l_1, l_2$ ):

| original | $l_1$ | $l_2$ |
|----------|-------|-------|
| 1        | 7     | 10    |
| 2        | 8     | 11    |
| 5        | 9     | 12    |

# How to get the locals for a value number?

- IR is an abstraction of the actual bytecode
- simple stack simulation tries to find the position at which the object is pushed onto / popped of the stack
- additionally save the position of the relevant (if any) store / load instruction
- not possible for branches

# Conversations

2 different scenarios:

- ① *The value is stored to a local*: Double the value and store it to the optimized local
- ② *The value is kept on the stack*: Convert the value on the stack

# Method call replacement

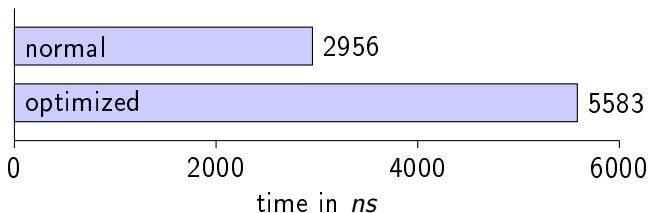
- replace the load instruction to load the optimized type
- replace the method call itself to match the expected optimized type
- replace the store instruction to store the result to the optimized type

## Section 6

# Benchmarks



# What are the results?



# What did go wrong?

## Section 7

# Conclusion

# Conclusion

- the system needs just the definition how to replace a certain method
- algorithm to determine the "bubble" is type independent, so not limited to String
- transformation on bytecode level makes the system applicable to already compiled programs (libraries in the classpath)

# Future Work

- loop sensitive `StringBuilder` optimization
- inter procedural optimization would boost performance
- a more offensive bubble growing strategy would cause a bigger bubble
- sources and slides at [github.com/wondee/faststring](https://github.com/wondee/faststring)
- because of Shrikes loose bytecode creation optimized classes need to be run with `-noverify`