Optimizing String manipulation performance

Markus Wondrak

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

July 24, 2014

Table of Contents

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 better way

```
StringBuilder result = new StringBuilder();

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

return result.toString();
```

Example

The optimized way

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

StringListBuilder builder = new StringListBuilder();

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

return builder.toString();
```

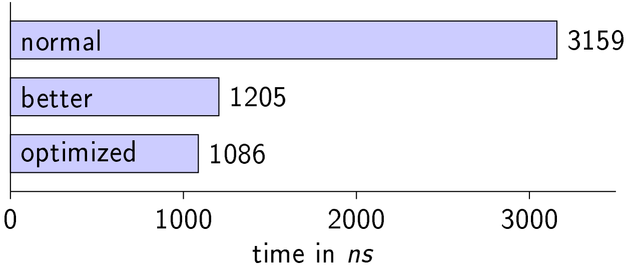
What is the difference?

- Java strings are immutable → manipulation causes `char[]` copy
- '+' operator is compiled to a `StringBuilder`, **but** the loop is not recognized
- `StringBuilder` also array-based, when filled up the array gets copied

What is the difference?

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

Measurement



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

What needs to be done?

Given a method optimization definition (e.g. SubstringString), the system should . . .

- . . . be applicable to a already compiled program
- . . . 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)
- Java, Groovy, Scala, JavaScript (and many more)
- Assembly language like
- Stack-based and imperative

Bytecode Example

Java:

```
String x = "Hallo World";
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
```

Section 3

WALA

WALA

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

Features

- Java type system and class hierarchy analysis
- supports frontends for Java Bytecode from class files
- SSA-based Intermediate Representation
- bytecode manipulation via the shrike project

Intermediate Representation

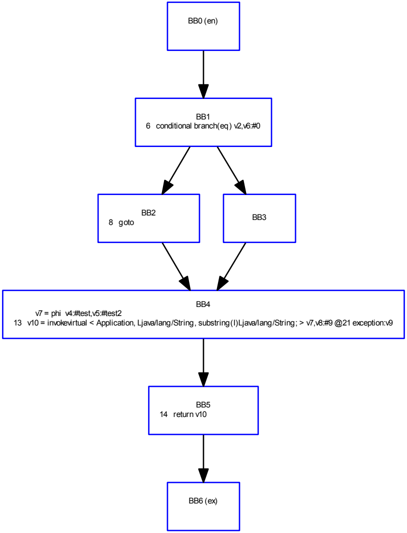
- central data structure that represents the analyzed method
- abstracts the actual bytecode in static single assignment form
- consists of a control-flow graph (with ϕs)

Phis

- Created when two variables are merged in the control flow graph
- simple meaning of $v_3 = \phi(v_1, v_2)$: v_3 can be v_1 or v_2
- when a basic block has more than one incoming edge

```
String a = "test";
String b = "test2";
String c = ((is) ? a:b);

return c.substring(9);
```


BB0 (en)
BB1
6 conditional branch(eq) v2,v6:#0
BB2
8 goto
BB3
BB4
v7 = phi v4:#test,v5:#test2
13 v10 = invokevirtual < Application, Ljava/lang/String, substring(I)Ljava/lang/String; > v7,v8:#9 @21 exception:v9
BB5
14 return v10
BB6 (ex)

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
- this "bubble" determines the value numbers for that optimization can applied

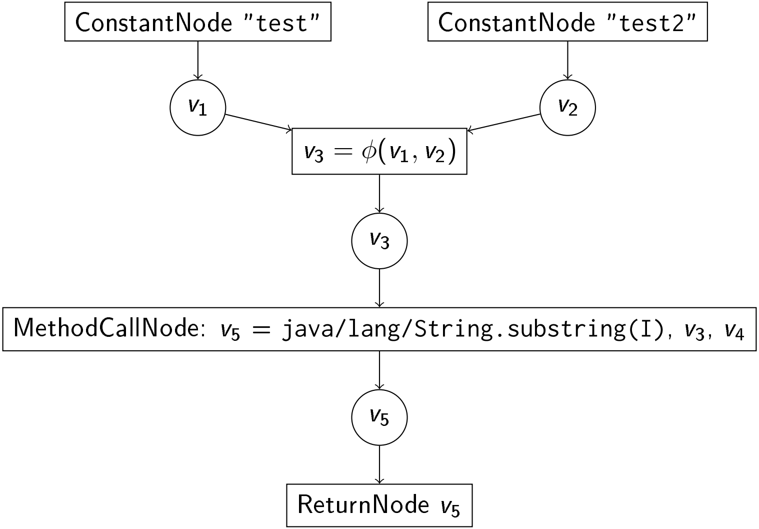
How does the dataflow graph look like?

- directed graph based on the IR
- is composed of 2 kinds of nodes
  - `Reference` merely the value number (R)
  - `InstructionNode` instruction that uses (or defines) value numbers (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
- NewNode a new instruction of an object
- PhiNode a ϕ node in the IR

example graph



How to define a label?

- for every optimized type a type implementing TypeLabel must be provided
- From the interface TypeLabel:

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

How to deal with phis?

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

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

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
- save the local and the position of the relevant (if any) store / load instruction within the `InstructionNode`

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 is optimized?

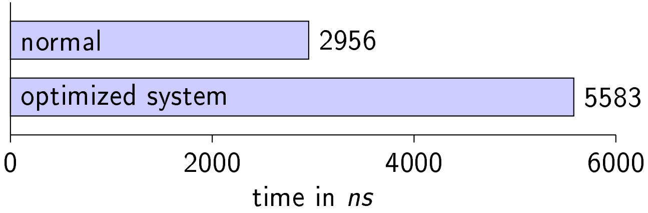
- the normal version with the '+' operator
- given the label for the optimized SubstringString

```
String result = "";

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

return result;
```

What are the results?



What went wrong?

- to append the new substring to the native `StringBuilder`, conversation is needed
- `char[]` copying takes place within the loop for every iteration

What is optimized?

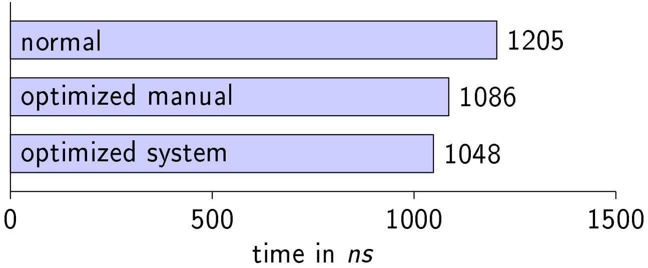
- the version with the StringBuilder used around the loop
- given labels for the optimized SubstringString and StringListBuilder

```java
StringBuilder result = new StringBuilder();

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

return result.toString();
```

What are the results?



Result of JMH

```
Benchmark                       Score   Score error
complexBuilderNormal       608304.032      4526.851
complexBuilderOpt          704025.416      6396.926
complexNormal              442065.793      5267.021
complexOpt                 647222.194      4874.660
```

Section 7

Conclusion

Lessons learned

- the growth of the bubble is very limited
- many conversations limit the performance (or make it even worse)
- provide an additional label that works with the other ones can boost the performance
- because of Shrikes loose bytecode creation optimized classes need to be run with `-noverify`
- Automated Tests are great!

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 improve performance
- a more offensive bubble growing strategy would cause a bigger bubble → less conversations
- sources and slides at `github.com/wondee/faststring`