Smart Coding using New Code Optimization Techniques in Java to Reduce Runtime Overhead of Java Compiler

Prajakta Gotarane
M. Tech Scholar
UMIT, SNDT Womens University, Santacruz, Mumbai

Sumedh Pundkar
Asst. Prof. In Comp.Sci. &Tech
UMIT, SNDT Womens University, Santacruz, Mumbai

ABSTRACT
Java is a popular object oriented programming language suitable for writing Java programs. Sometimes programmers spend most of the time to increase the execution time of the program, but simultaneously its effect on code size. Therefor the code become more complex and unreliable, so this leads to reduce the efficiency of code. Today so many compilers are exist like c, javac,c++,cobol,etc.we studied the code optimization techniques for java compiler separately and that time we come across some new code optimization strategies which is the smart way to do the coding in java. In this paper we applied some new java code optimization techniques on existing code. We verify the code optimization, performance using our executor. These code optimization strategies indirectly help to reduce the work of garbage collection, data structure and also work on loop optimization. So the results which we found after doing experimentations are quite satisfactory as compare to original results. so these techniques are help to improve the code quality.

Keywords
Code optimization, code efficiency, execution time, code quality, readability. Garbage Collection, Loop optimization, Data Structure.

1. INTRODUCTION
In compiler design, Optimization is the process of .transforming a piece of code (un-optimized code) to make more efficient without changing its output. The optimized programmer is simply defined as a program is smaller in size ,which consume less memory also which required less execution time. Most of the time while writing a java program the programmer can easily make simple mistakes that are harmless for small application. but as the application grows the performance of java application become slowler.so to improve the performance of an java application code optimization is the important factor. On using different optimization techniques, the code can be optimized without affecting the original (actual) algorithm and final output with the intent of high performance. When performance is to be considered, then there is need to choose an algorithm which runs quickly and the available computing resources are being used.

Basically, Code optimization involves the employment rules and algorithms to the program segment with the goal, such that the code becomes efficient ,requires less memory and execute faster and so on. Optimization is classified as high level optimization and low level optimization. High level optimization are usually performed by those programmers who handles abstract entities and also keeps in mind the general framework of the task to optimize design of a system. On the other hand, low level optimization is performed at the stage when source code is compiled into a set of machine instructions.

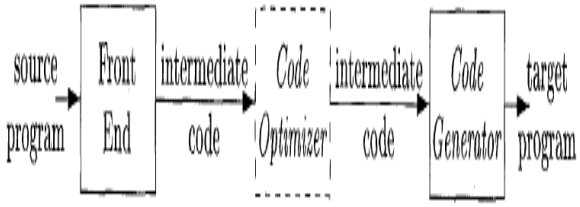

Fig 1: Code optimization process

In the above fig shows how the optimization works in any compiler. The source code which is written in any programming language .

The major part of code optimization includes the output does not changes after applying the techniques. The semantic of the code optimization should be preserved.

The structure of the paper is as follows: section 2 contains literature survey of the existing code optimization techniques. section 3 contains proposed techniques which are working on java language. Section 4 contains results and analysis also this paper ended with conclusion and feature scope.

2. LITERATURE REIVIEW

Table 1: Existing Code Optimization Techniques

S.N	Code Optimization Techniques	Extract of the paper	How code optimization works

1	Constant Folding	Optimization is done by replacing all expressions with constant result computed at compile time	public void CP() {
			int c= 4+1;//the value of c is always 5 so compiler replace the value of c by 5 directly
			System.out.println(c);
			}

2	Constant Propagation	Compute as many possible values at compile time for optimization of code	int b = 6;
			System.out.println(b);//b will always be 6,
			So the statement b=6 has no meaning in the program

3	Useless Expression Elimination	Done by eliminating unnecessary expressions	Sometimes we assign a value to the variable like s=2; bt this variable is not used in the program so there is no need to assign the extra variables

4	Copy Propagation	Done by replacing one variable by another when they are equal	public void CP1(int x) {
			int y = x;
			System.out.println(y); // y=x therefore optimization replace x by y
			}

5	Common SubExpression Elimination	This optimization uses the concept of temporary variables. temp variable help to store the intermediate result.so the common subexpressions are eliminated	public void CSE(int x,int y) {
			int p= x*y;
			int q = x*y;
			System.out.println(p); //so p==q
			System.out.println(q);
			}

6	Reduce Mathematical Strength	This optimization is done by changing the mathematical expression, which requires a longer time for computation are replaced by the expression which require less time for computing the same .	If the expression is p=f*2 then for optimization you can replace above expression by p=f+f. For optimization replaces multiplication by addition, And exponentional by multiplication.

7	Global Constant Propagation	Optimization is done throughout in program by replacing the constant expression by constant values at compile time.	public void GCP(boolean P) {
			int X= 5;
			int R;
			if(P) {
			R = X; }
			else {
			R= X; }
			int d = R;
			System.out.println(d);
			}

8	Global Common Subexpression Elimination	AS we know common subexpressions use the concept of temporary variables. So in this optimization the temporary values are computed at once during runtime	Here, if x=p*q;
			Y=p*q; // so x=y;
			G=p*q; // so here once the compiler computes value of p*q, so no need to compute the value of p* q again and again at this stage we can use the concept of temporary variable to store the result.

9	Dead Code Elimination	The optimization is done by removing the byte code which is generated at output that should not be executed.	Some conditions or statements which never be true ,should be eliminated for optimization
10	Code Hoisting	The optimization is done by computing busy expressions as early as possible. So this will help to reduce the size of code	Code hoisting is performed in java to reduce the size of code .

3. PROPOSED TECHNIQUE:

In this section we proposed new code optimization techniques which help to reduce runtime overhead on java compiler. So by using the below techniques the programmer can speed up the execution time as well as code quality will improve too. the implementation part is done on following:

Section 1. the Controller control the input file and generation of output file call the patterns.

Section 2.both input and output files goes through the executor which compile both file and calculate the execution time of both the files. The executor creates. .class file and .jar file fot both the inputs while compilation

Section 3 Report generator will display the runtime and difference between both the files.

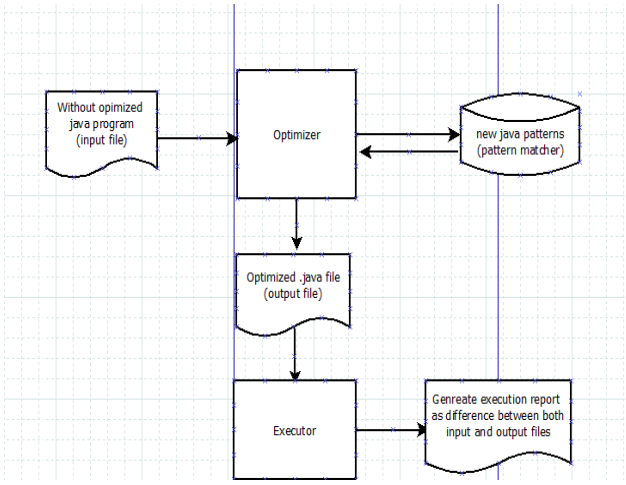

Fig 2: Java Code Optimization framework

A) String append pattern:

This technique helps to reduce the work of garbage collection. Garbage collection is automatic in java programs. When the object is no longer more useful than it will available for GC.so disposing a java object is called as garbage collection. The two techniques can apply which can help reduce the work of GC.

In the first technique, applications can use the existing object so there will be no need to create and destroy the object again and again, but in this case the programmer has to do the extra work .Because the value of the object is needed to be reinitialized.

The 2nd technique which can reduce the work of GC is use the appropriate object only which can meet the exact requirement.

We all know the fact that concatenation of Two stings is expensive because of the immutable property of the string.immutable means the value of the string can not be change. So whenever we performed the string concatenation the intermediate result is created which is string object so each of the string object is needed to be GC.

1. Sample code(un optimized and optimized code)

```
* Source Code Optimization Projects
package com.codeoptimzation.input;
public class StringAppendPattern
{
  public StringAppendPattern()
  {
  String variable = "GSMoze";
   for( int i = 0 ; i < 1000 ; i++)
   {
  variable = "India" + variable;
   }
    system.out.println(variable);
}

public static void main(String[] args)
{
 new StringAppendPattern();
}

}
```

```
package com.codeoptimzation.output;

public class StringAppendPattern{

  public StringAppendPattern() {

  StringBuffer variable = new
StringBuffer("GSMoze");

for (int i = 0; i < 1000; i++) {

    variable.append("India");

    variable.append(variable);

    System.out.println(variable);

}

public static void main(String[] args) {

    new StringAppendPattern();

  }}
```

Above code shows the un-optimized and optimized code. in this experimentation we gave un optimized file as input to our controller which check all the pattern related to + operator and then produced a new optimized code which new function in place of + operator is .append().

2. Executor

Process both unoptimized and optimized files on executor so it will process both files. this executor will generate .class and .jar file for both the input and output file.

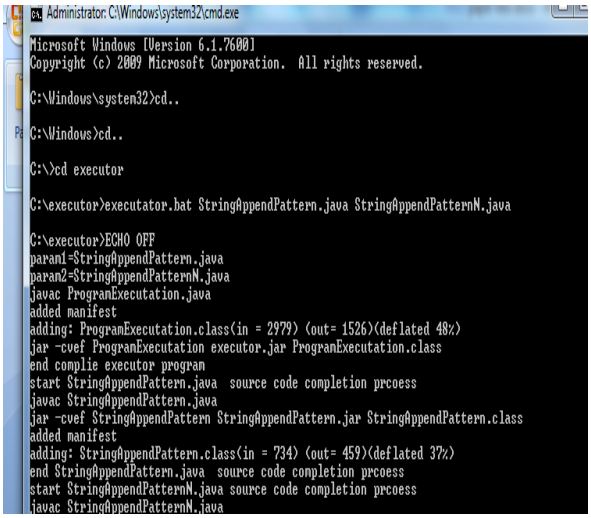

Fig 2: Executor working

3. Result Formation

After executing both the files on executor. the executor compilers both the files and find out the how much time is required for both files for execution and the difference between them .also this result will displayed in html page.

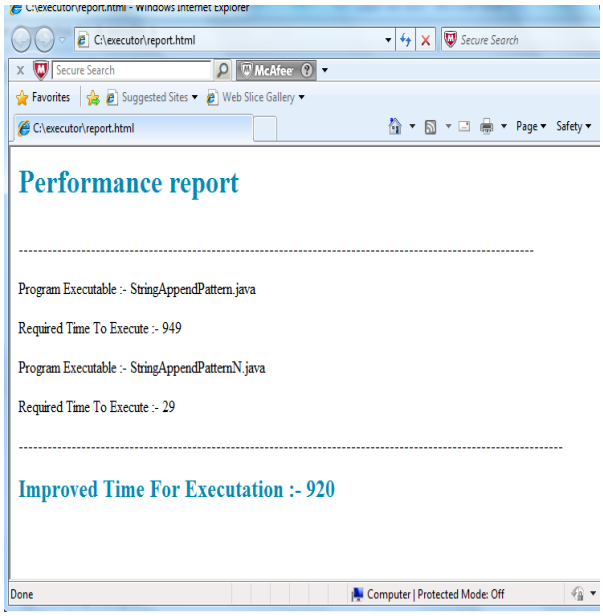

Fig 3: Final Report Generation

Other optimization Techniques are as follows:

Table 2: Proposed code optimization techniques

S.N	OPTIMIZATION TECHNIQUES	CODE BEFORE OPTIMIZATION	EXECUTION TIME IN MILE SECONDS	CODE AFTER OPTIMIZATION	EXECUTION TIME IN MILE SECONDS
1	Avoid new with string to reduce stack pool size	String str = new String("string"); String str1 = new String(); String str2 = new String(str1);	995	String str = "string"; String str1 = ""; String str2 = str1;	20

2	Use stringbuffer append pattern.	String variable=”abc”;
		Variable=”india”+variable;
			949	public StringAppendPattern() {
				StringBuffer variable = new StringBuffer(“abc");
				for (int i = 0; i < 1000; i++) {
				variable.append("India");
				variable.append(variable);
					29

3.	Avoid creating thread without run method	public class TR { public void method() throws Exception{
		new Thread().start();
			456	public class TR{
				public void method(Runnable r) throws Exception {
				new Thread(r).start();
					43

4	Thread sleep pattern	public void ThreadSleep() throws Exception{
		this.wait(1000);
		System.out.println("nnnnnnnnnnnn");
		Thread.sleep(10000);
		System.out.println("hiiiiiiiiiiiii");
		}
			1926	public void ThreadSleep() throws Exception {
				this.wait(1000);
				System.out.println("nnnnnnnnnnnn");
				this.wait(10000);
				System.out.println("hiiiiiiiiiiiii"); }
					55

5	Avoid using java lang Class forName.	Class.forname(java.lang.Integer).getName());	968	System.out.println(java.lang.Integer.class.getName()); // CORRECTION }	62
6.	Use arraylist instead of Linked Lists	LinkedList<Object> list = **new** LinkedList<Object>();LinkedList<Object> list1 = **new** LinkedList<Object>(); LinkedList<Object> list2 = **new** LinkedList<Object>();	15	ArrayList<Object> list = **new** ArrayList<Object>(); ArrayList<Object> list1 = **new** ArrayList<Object>(); ArrayList<Object> list2 = **new** ArrayList<Object>();	13

7	Do not use empty static initialize.	public class SI{
		static // VIOLATION
		{
		// empty
		} }
			33	public class SI{
				// ...
				} }
					27

8	Use short circuit Boolean operator instead Of BinaryOperator.	String sValue = "binary";
		if(sValue.equals("true") | sValue.equals("false")) // unoptimize code
		{System.out.println("valid boolean");}
			931	String sValue = "binary"; **if** (sValue.equals("true") || sValue.equals("false"))
				// optimize
				{
				System.*out*.println("valid boolean");
				}
					23

9	For optimization Avoid empty if	if (n<0)
		// without optimization
		{
		}
		n =0;
		} }
			867	if (n<0) // optimization
				{
				}
				*/
				N=0;;
				} }
					24

10	String equal pattern	String str = **new** String();
		if(str.equals(""))
			939	**public void** stringEqualPattern() {
				String str = "";
				if (str.length() == 0) {}
				}
					23

11	Optimize array size	public static void main(String[] args) {
		ArrayList<String> list = new ArrayList<String>();
		for(int index = 0 ; index < 200 ; index++){
		list.add(index +"")}
		for(int index = 0 ; index < list.size() ; index++){
		ArrayList<Integer> integerlist = new ArrayList<Integer>();
		integerlist.add(index);
			28	ArrayList<String> list = new ArrayList<String>();
				for(int index = 0 ; index < 200 ; index++){
				list.add(index +"");}
				int size = 10;
				ArrayList<Integer> integerlist = new ArrayList<Integer>();
				for(int index = 0 ; index < list.size() ; index++){
				integerlist.add(index);
				integerlist.clear();
					16

12	Avoid input output	try{	1906	try {	1021

	operations in loop	FileWriter fileWriter = new FileWriter(new File("temp.txt"));
		
		for(int i = 0 ; i < 500000000 ; i++){
		
		fileWriter.write(i +"");
		
		fileWriter.flush();
		
		}}
				FileWriter fileWriter = new FileWriter(new File("temp.txt"))\
				
				StringBuffer var = new StringBuffer();
				
				for (int i = 0; i < 500000000; i++) {var.append(i + ""
				fileWriter.flush(); }
				
				fileWriter.write(var.toString());
				fileWriter.flush(); }	

13	Avoid use of integer to tostring pattern in program	String str="goods" System.*out*.println(str.toString()); String s ="abc "+str.substring(0); System.*out*.println(s.toString()); System.*out*.println(s.substring(0)); AvoidTostring a = new AvoidTostring(); a.xyz (s)	946	String str = "goods" System.*out*.println(str); String s = "abc " + str System.*out*.println(s); Systzem.*out*.println(s); AvoidTostring a = new AvoidTostring(); a.xyz(s);	72

4. CONCLUSION AND FUTURE SCOPE

In this paper the experimentation shows our approach for code optimization using above techniques. These techniques are use to speed up the program execution without affecting the final output. So this tool provides a way to optimize the unoptimized code and reduce complexity of code. One more benefit of this tool is this will increase the quality of the code.

In future work we will incorporate the other techniques which will suggest the programmer how to do the smart coding. This will also helpful for the beginner of the programmer.

5. REFERENCES

[1] Michael Dorf, (2012), "5 Easy Java Optimization Tips", http://www.learncomputer.com/java-optimization-tips .

[2] IBM, (2009) , "Optimizing C code atoptimization level 2 ", Copyright International Business Machines Corporation 2009.

[3] Maggie Johnson,(2008) ,"Code Optimization",Handout 20.

[4] Kevin Williams1,Albert Noll2,Andreas Gal3 and David Gregg1 ,(2008) , "Optimization Strategies for a Java Virtual Machine Interpreter on the Cell Broadband Engine"1Trinity College Dublin, Dublin, Ireland,2ETH Zurich, Zurich, Switzerland. 3University of California, Irvine, CA, USA.

[5] Huib van den Brink, (2008), "The current and future optimizations performed by the Java HotSpotCompiler" , Institute of information and Computing Sciences, Utrecht University P.O. Box 80.089, 3508 TB Utrecht, The Netherlands.

[6] Pawan Nagar1,Nitasha Soni2, (2012)," Optimizing Program-States using Exception-Handling Constructerin Java ",1M.Tech.Scholar, CSE Department, Lingaya's University ,Haryana, India ,2Lecturer, CSE Department, Lingaya's University, Haryana, India, International Journal of Engineering Science &Advanced Technology.

[7] Hiroshi Inoue and Toshio Nakatani ,(2012) ,"Identifying the Sources of Cache Misses in Java Programs Without Relying on Hardware Counters ",© ACM, 2012. This is the author's version of the work.

[8] Peter Sestoft ,(2010) ,"Numeric performance in C, C# and Java",IT University of CopenhagenDenmark,Version 0.9.1 of 2010-02- 19.

[9] http://www.onjava.com/pub/a/onjava/2002/03/20/optimization.html?page=4http://www.javaperformancetuning.com/tips/rawtips. shtml.

[10] http://www.appperfect.com/support/java-coding-rules/optimization.html.

[11] Tony Sintes , (2002) ,"The String class's strange behavior explained",http://www.javaworld.com/article/2077355/core-java/don-tbe- strung-along.html.

[12] Ont Community ,(2012),"About .class operator ",Oracle.

[13] Technology Specialist, (2012), "One 4 All", http://www.javaperformancetuning.com/tips/rawtips.shtml, ,http://www.glenmccl.com/jperf/.

[14] Felix Hernandez-Campos ,(2002) ,"COMP 144 Programming Language Concepts" ,The University of North Carolina at Chapel Hill.Ben Van Vliet ,(2008),"C++FA 3.1 OPTIMIZING C++", http://www.benvanvliet.net/Downloads/CFA3.1_Optimizing%20CPP.

[15] Guihot, H. (2012). Optimizing Java Code. Pro Android Apps Performance Optimization, Springer**:** 1-31.