An Comparative Study on various Java Compiler Optimization Techniques

R Sathish kumar[1] ,D.Ranjani [2]

Assistant Professor ,Department of Electrical and Electronics Engineering ,Sri Krishna College of Technology

Assistant Professor ,Department of Information Technology,Sri Krishna College of Technology

Abstract: Compilers can be designed to provide code optimization. Users should only focus on optimizations not provided by the compiler such as choosing a faster and/or less memory intensive algorithm. A Code optimizer sits between the front end and the code generator. Works with intermediate code Can do control flow analysis.Can do data flow analysis.Does transformations to improve the intermediate code. Optimizations provided by a compiler includes Inlining small functions ,Code hoisting ,Dead store elimination,Eliminating common sub-expressions ,Loop unrolling,Loop optimizations: Code motion, Induction variable elimination, and Reduction in strength.This paper deals with the Comparative Study on various Compiler Optimization Techniques.

1. INTRODUCTION

Inlining small functions Repeatedly inserting the function code instead of calling it, saves the calling overhead and enable further optimizations. Inlining large functions will make the executable too large.Code hoisting Moving computations outside loops Saves computing time.Code hoistingIn the following example (2.0 * PI) is an invariant expression there is no reason to recompute it 100 times.

```
DO I = 1, 100
ARRAY(I) = 2.0 * PI * I
ENDDO
```

By introducing a temporary variable 't' it can be transformed to:

```
t = 2.0 * PI
DO I = 1, 100
ARRAY(I) = t * I
END DO
```

Dead store elimination -If the compiler detects variables that are never used, it may safely ignore many of the operations that compute their values. Eliminating common sub-expressions.Optimization compilers are able to perform quite well:

```
X = A * LOG(Y) + (LOG(Y) ** 2)
```

Introduce an explicit temporary variable t:

```
t = LOG(Y)
X = A * t + (t ** 2)
```

Saves one 'heavy' function call, by an elimination of the common sub-expression LOG(Y), the exponentiation now is:

```
X = (A + t) * t
```

Loop unrolling-The loop exit checks cost CPU time.Loop unrolling tries to get rid of the checks completely or to reduce the number of checks.If you know a loop is only performed a certain number of times, or if you know the number of times it will be repeated is a multiple of a constant you can unroll this loop.

Loop unrolling

Example:

```
// old loop
for(int i=0; i<3; i++) {
        color_map[n+i] = i;
}
// unrolled version
```

```
int i = 0;

colormap[n+i] = i;

i++;

colormap[n+i] = i;

i++;

colormap[n+i] = i;
```

Code Motion -Any code inside a loop that always computes the same value can be moved before the loop.

Example:

```
while (i <= limit-2)

do {loop code}
```

where the loop code doesn't change the limit variable. The subtraction, limit-2, will be inside the loop. Code motion would substitute:

```
t = limit-2;

while (i <= t)

do {loop code}
```

Compilers can provide some code optimization. Programmers do have to worry about such optimizations.Program definition must be preserved.

2. LITERATURE SURVEY

2.1 Java Performance Optimization

PMD

PMD scans Java source code and looks for potential problems like Possible bugs – empty try/catch/finally/switch statements,Dead code – unused local variables, parameters and private methods.Suboptimal code – wasteful String/StringBuffer usage. Overcomplicated expressions – unnecessary if statements, for loops that could be while loops.Duplicate code – copied/pasted code means copied/pasted bugs

PMD is integrated with JDeveloper, Eclipse, JEdit, JBuilder, BlueJ, CodeGuide, NetBeans/Sun Java Studio Enterprise/Creator, IntelliJ IDEA, TextPad, Maven, Ant, Gel, JCreator, and Emacs.

FindBug

FindBugs, a program which uses static analysis to look for bugs in Java code.

Clover

Measures statement, method, and branch coverage and has XML, HTML, and GUI reporting. and comprehensive plug-ins for major IDEs.Improve Test Quality Increase Testing Productivity.Keep Team on TrackFully integrated plugins for NetBeans, Eclipse , IntelliJ IDEA, JBuilder and JDeveloper. These plugins allow you to measure and inspect coverage results without leaving the IDE.
Seamless Integration with projects using Apache Ant and Maven. * Easy integration into legacy build systems with command line interface and API.Fast, accurate, configurable, detailed coverage reporting of Method, Statement, and Branch coverage.
Rich reporting in HTML, PDF, XML or a Swing GUI
Precise control over the coverage gathering with source-levelfiltering.Historical charting of code coverage and other metrics.Fully compatible with JUnit 3.x & 4.x, TestNG, JTiger and other testing frameworks. Can also be used with manual, functional or integration testing.

Macker

Macker is a build-time architectural rule checking utility for Java developers. It's meant to model the architectural ideals programmers always dream up for their projects, and then break — it helps keep code clean and consistent. You can tailor a rules file to suit a specific project's structure, or write some general "good practice" rules for your code. Macker doesn't try to shove anybody else's rules down your throat; it's flexible, and writing a rules file is part of the development process for each unique project.

EMMA

Reports on class, method, basic block, and line coverage (text, HTML, and XML).EMMA can

instrument classes for coverage either offline (before they are loaded) or on the fly (using an instrumenting application classloader).Supported coverage types: class, method, line, basic block. EMMA can detect when a single source code line is covered only partially.Coverage stats are aggregated at method, class, package, and "all classes" levels.Output report types: plain text, HTML, XML. All report types support drill-down, to a user-controlled detail depth. The HTML report supports source code linking.Output reports can highlight items with coverage levels below user-provided thresholds.Coverage data obtained in different instrumentation or test runs can be merged together.EMMA does not require access to the source code and degrades gracefully with decreasing amount of debug information available in the input classes.EMMA can instrument individial .class files or entire .jars (in place, if desired). Efficient coverage subset filtering is possible, too.Makefile and ANT build integration are supported on equal footing.EMMA is quite fast: the runtime overhead of added instrumentation is small (5-20%) and the bytecode instrumentor itself is very fast (mostly limited by file I/O speed). Memory overhead is a few hundred bytes per Java class.EMMA is 100% pure Java, has no external library dependencies, and works in any Java 2 JVM (even 1.2.x).

XRadar

The XRadar is an open extensible code report tool currently supporting all Java based systems. The batch-processing framework produces HTML/SVG reports of the systems current state and the development over time – all presented in sexy tables and graphs.The XRadar gives measurements on standard software metrics such as package metrics and dependencies, code size and complexity, code duplications, coding violations and code-style violations.

Hammurapi

Hammurapi is a tool for execution of automated inspection of Java program code. Following the example of 282 rules of Hammurabi's code, we are offered over 120 Java classes, the so-called inspectors, which can, at three levels (source code, packages, repository of Java files), state whether the analysed source code contains violations of commonly accepted standards of coding.

Relief

Relief is a design tool providing a new look on Java projects. Relying on our ability to deal with real objects by examining their shape, size or relative place in space it gives a "physical" view on java packages, types and fields and their relationships, making them easier to handle.

Hudson

Hudson is a continuous integration (CI) tool written in Java, which runs in a servlet container, such as Apache Tomcat or the GlassFish application server. It supports SCM tools including CVS, Subversion, Git and Clearcase and can execute Apache Ant and Apache Maven based projects, as well as arbitrary shell scripts and Windows batch commands.

Cobertura

Cobertura is a free Java tool that calculates the percentage of code accessed by tests. It can be used to identify which parts of your Java program are lacking test coverage. It is based on jcoverage.

SonarSource

Sonar is an open platform to manage code quality. As such, it covers the 7 axes of code quality:Architecture & Design, Duplications, Unit Tests, Complexity, Potential bugs, Coding rules, Comments.

3. CONCLUSION

Thus the paper depicts the comparative study on various java compiler optimization Techniques and the java tools for implementing it.

REFERENCES

[1] https://dzone.com/articles/java-tools-source-code

[2] https://opensource.com/life/16/8/5-great-tricks-java-performance-optimization

[3] https://www.tutorialspoint.com/compiler_design/compiler_design_code_optimization.htm

[4] http://programming4.us/desktop/462.aspx

[5] https://pdfs.semanticscholar.org/6ce2/e99865863ad262c610b9acb626f4ad24650e.pdf