

**JVM COMPILER BACKEND FOR BALLERINA
INTERMEDIATE REPRESENTATION**

Thangarajah Kishanthan

179329D

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

May 2019

**JVM COMPILER BACKEND FOR BALLERINA
INTERMEDIATE REPRESENTATION**

Thangarajah Kishanthan
179329D

Thesis submitted in partial fulfillment of the requirements for the degree Master of
Science

Department of Computer Science and Engineering
University of Moratuwa
Sri Lanka

May 2019

DECLARATION

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis/dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature:

Date:

(T. Kishanthan)

The above candidate has carried out research for the Masters thesis under my supervision.

Signature:

Date:

(Dr. Indika Perera)

ABSTRACT

Ballerina is an open source, strongly typed language for writing microservices and network applications with main focus on solving enterprise integration requirements. The ballerina compiler converts the ballerina source to set of ballerina byte code which is then executed by the ballerina virtual machine (BVM). The BVM does not perform well for most of the CPU bound operations due to its current design. This project focus on compiling ballerina source to JVM byte code and will be executed by the JVM directly, which will solve the performance bottleneck at BVM. This project also proposes a new compiler architecture, in which, the ballerina source code is transformed to an intermediate representation which is a low level representation of the ballerina program and it is used for generating the target JVM byte code. The performance of JVM based compiler backend implementation against the current BVM was compared for certain algorithms and programs. From the evaluation of the test results, it is found that the JVM target outperforms the ballerina runtime by factor of 100 in certain scenarios. With this promising results, the proposed new compiler architecture based on ballerina intermediate representation and the JVM compiler backend can potentially be used as the replacement for current ballerina compiler and runtime.

ACKNOWLEDGEMENT

I would like to express my profound gratitude to my supervisor Dr. Indika Perera, for his invaluable support throughout by advising and guiding me through the correct directions. His expertise and continuous guidance are one of the key reasons for the success of this research.

I should also thank Dr. Sanjiva Weerawarana (Founder and Chairman of WSO2) and Mr. Sameera Jayasoma (Senior Director at WSO2), who encouraged me to complete this research project. I am mostly thankful for my wife, my parents for their immense support and patience at all the time. At last but not least, I'm thankful to all my colleagues at WSO2, who helped me a lot in various ways throughout this research.

TABLE OF CONTENTS

DECLARATION	i
ABSTRACT	ii
ACKNOWLEDGEMENT	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	vii
LIST OF TABLES	viii
LIST OF ABBREVIATIONS	ix
1. INTRODUCTION	1
1.1 Ballerina	1
1.2 Ballerina Compiler	3
1.3 Problem	4
1.4 Objectives	5
2. LITERATURE REVIEW	6
2.1 Ballerina Compiler	6
2.1.1 Frontend Phase	8
2.1.2 Optimizer Phase	10
2.1.3 Backend Phase	10
2.2 Intermediate Representation	11
2.3 Ballerina Intermediate Representation	14
2.3.1 BIR Instructions	18
2.4 JVM Class File	20
2.5 JVM Runtime Execution Model	21
2.6 JVM Instruction Set	24
3. METHODOLOGY	26
3.1 JVM Compiler Backend For Ballerina	26
3.2 AST to BIR Generation	27
3.3 BIR to JVM Target Generation	28
4. IMPLEMENTATION	29
4.1 Modeling Ballerina Types & Values	29

4.1.1 Simple Basic Types	29
4.1.2 Structured & Behavioral Types	30
4.2 Modeling Ballerina Project	31
4.2.1 Modeling package	32
4.2.2 Modeling the class with the description	33
4.2.3 Generation of method(s) description and signatures	34
4.2.4 Generation of method body	35
4.2.5 Processing of method return value	36
4.2.6 Processing of method arguments	37
4.2.7 Processing of method basic blocks	38
4.2.8 Processing of basic block instructions	40
4.2.9 Constant Load Instruction	41
4.2.10 Move Instruction	42
4.2.11 Binary Operation Instructions	43
4.2.12 Add Instruction	43
4.2.13 Subtract Instruction	45
4.2.14 Multiply & Divide Instructions	46
4.2.15 Equal Instruction	46
4.2.16 Condition Based Instructions	47
4.2.17 AND Instruction	49
4.2.18 OR Instruction	50
4.2.19 Array Load Instruction	51
4.2.20 Array Store Instruction	51
4.2.21 Loops	52
4.2.22 Length Instruction	53
4.2.23 Processing of basic block termination instructions	54
4.2.24 Generation of class file content into binary a file (.class)	57
4.3 Updating Ballerina Build Command	57
5. RESULTS AND EVALUATION	59
5.1 Fibonacci Series	60
5.2 Merge Sort	61
5.3 Quick Sort	62
5.4 Matrix Multiplication	63

5.5 String Regular Expression Match	64
5.6 Evaluation of the results	65
6. CONCLUSION	68
6.1 Limitations	68
6.2 Future Work	69
6.2.1 Reference types support	69
6.2.2 Debugging support	69
6.2.3 Error and stack trace modeling	69
6.2.4 Concurrency modeling	70
6.2.5 Update BIR model with all language constructs	70
7. REFERENCES	71

LIST OF FIGURES

Figure 1.1 Textual view of a ballerina program	2
Figure 1.2 Graphical view of a ballerina program	2
Figure 2.1 Ballerina Compiler Architecture	7
Figure 2.2 Ballerina Compiler Frontend	8
Figure 2.3 Ballerina Compiler Backend	11
Figure 2.4 Example BIR in CFG	16
Figure 2.5 Java Class Structure	21
Figure 2.6 JVM Stack Frame	22
Figure 3.1 Proposed JVM Compiler Backend For Ballerina	26
Figure 4.1 Directory Structure of ballerina/http Module	32
Figure 4.2 Java Package Structure of ballerina/http Module	32
Figure 4.3 Java Method Signature Mapping	35
Figure 4.4 Java Method Signature Example	35
Figure 4.5 Example BIR	36
Figure 4.6 BIR Variable To JVM Index Mapping	37
Figure 4.7 Java Bytecode Generated	40
Figure 5.1 Fibonacci Test	60
Figure 5.2 Mergesort Test	61
Figure 5.3 Quicksort Test	62
Figure 5.4 Matrix multiplication Test	63
Figure 5.5 String Regular Expression Match Test	64
Figure 5.6 JProfiler based Profiled view of BVM	66

LIST OF TABLES

Table 2.1 Example LLVM CFG	14
Table 2.2 Example BIR	15
Table 2.3 BIR Constructs	16
Table 2.4 BIR Instructions	18
Table 4.1 Ballerina Basic Value Type Mapping	29
Table 4.2 Ballerina Structured & Behavioral Type Matching	30
Table 4.3 Example Ballerina Source To Java Class Mapping	33
Table 4.4 Basic Block Generation	38
Table 4.5 Constant Load Instruction Generation	41
Table 4.6 Constant Load Instruction Mapping	42
Table 4.7 Move Instruction Mapping	43
Table 4.8 Types of Move Instruction Mappings	43
Table 4.9 Add Instruction Generation	44
Table 4.10 Add Instruction Mapping	44
Table 4.11 String Concatenation Mapping	45
Table 4.12 Subtract Instruction Mapping	45
Table 4.13 Multiply & Divide Load Instruction Mapping	46
Table 4.14 Equal Instruction Generation	46
Table 4.15 Equal Instruction Mapping	47
Table 4.16 Condition Based Instruction Mapping	47
Table 4.17 Binary AND Instruction Generation	49
Table 4.18 Binary AND Instruction Mapping	50
Table 4.19 Binary OR Instruction Mapping	50
Table 4.20 Array Load Instruction Mapping	51
Table 4.21 Array Store Instruction Mapping	51
Table 4.22 Mapping of Loops	52
Table 4.23 Length Instruction Generation	53
Table 4.24 Length Instruction Mapping	54
Table 4.25 Basicblock Termination Types	54
Table 4.26 Call Instruction Generation	55
Table 4.27 Call Instruction Mapping	57
Table 5.1 Test System Configuration	59

LIST OF ABBREVIATIONS

Abbreviation	Description
BVM	Ballerina Virtual Machine
AST	Abstract Syntax Tree
IR	Intermediate Representation
BIR	Ballerina Intermediate Representation
JVM	Java Virtual Machine
LLVM	Low Level Virtual Machine
CFG	Control Flow Graph