DEX バイトコード解析による非 JIT ART 環境における Kotlin および Java プログラムの性能に関する一考察

柴田　涼一[†1]　園山　敦也[†1]　佐藤　悠祐[†1]　神山　剛[†2]
福田　晃[†2]　小口　正人[†3]　山口　実靖[†1]

キーワード：Java，Kotlin，Android Runtime，DEX バイトコード，Java バイトコード

1. はじめに

Android におけるアプリケーション開発言語には長年 Java が用いられきたが，近年は新しい言語である Kotlin による開発の事例も増えてきており，2018 年に Kotlin は Android の公式言語とされ[1], Android アプリケーション開発言語として Kotlin が注目を集めている．ただし，Kotlin は新しい言語であり，同一の処理を Java 言語と Kotlin 言語で記述したときに得られる性能が同一となるかなどの検証は十分にされておらず，性能に関する考察が重要となっている。

Kotlin は JVM(Java Virtual Machine)ベースの言語であり，Kotlin と Java の両言語は共通のライブラリを使用できるなど高い互換性を持っている．Android アプリケーションを開発する場合はどちらの言語で開発を行ったとしても，まずそれぞれの言語のソースコードが Java バイトコードに変換(コンパイル)され，次にその Java バイトコードが DEX(Dalvik Executable)バイトコードに変換(コンパイル)され，それがアプリケーションとなる．DEX ベースのアプリケーションは，ART(Android Runtime)上で実行される．実行の際，ART は DEX バイトコードを非ネイティブコードのまま実行する場合と，一部を JIT(Just-In-Time)コンパイラによりネイティブコードにコンパイルして実行する場合がある．

本稿では JIT を用いていない ART 環境における Java と Kotlin で記述した for 文処理の性能差について考察し，DEX バイトコードの改変により Java で記述したアプリケーションの性能を改善する手法を提案する．

2. 既存手法

我々は文献[2]にて図 1 の Kotlin 記述の for 文処理と Java 記述の for 文処理の性能を比較し，図 2 の Java(通常手法)と Kotlin の様に Kotlin 記述の for 文の方が性能が高いことを示した．

そして，図 3 と図 4 の様に Java 記述のソースコードより生成された DEX バイトコードにおける for 文処理と Kotlin 記述のバイトコードを解析し，goto 移行先などに差異があることを示し，Java ソースコードより生成されたバイトコードの移行先を改変することにより，Java 記述の for 文処理の性能を向上させることができることを示した．具体的には，Java 記述の for 文の DEX バイトコードは図 4 の様に定数処理を繰り返し実行する非効率的なものであることを示し，図 5 の様に移行先から定数処理を省くことにより性能を向上できることを示した．

ただし，改変後の Java 記述の for 文の性能も Kotlin 記述のそれより依然として劣っており，さらなる改善の余地が確認されている．

```
Java:
for(int j=1; j<=100000000; j++){ }

Kotlin:
for(j in 1..100000000){ }
```

図 1. 測定プログラム

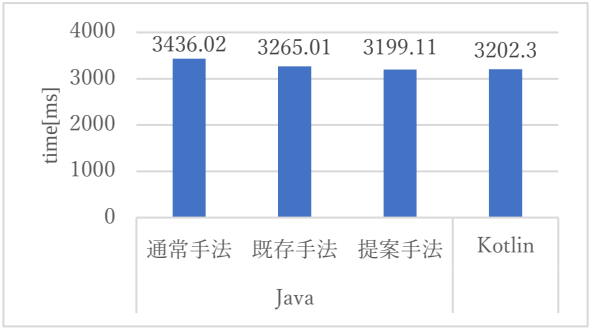

図 2 各プログラムの実行時間

```
14d8c8: 1405 00e1 f505      |0012: const v5, #05f5e100
14d8ce: 1216                |0015: const/4 v6, #1
14d8d0: 1217                |0016: const/4 v7, #1
14d8d2: 3657 0400           |0017: if-gt v7, v5, +0004
14d8d6: b067                |0019: add-int/2addr v7, v6
14d8d8: 28fd                |001a: goto -0003
```

図 3 Kotlin プログラムより生成された DEX バイトコード

3. 提案手法

本章にて，既存手法[2]より高度な改変を行い，Java 由来

†1 工学院大学
Kogakuin University
†2 九州大学
Kyushu University
†3 お茶の水女子大学
Ochanomizu University

```
1240a0: 1215                |0014: const/4 v5, #1
1240a2: 1406 00e1 f505      |0015: const v6, #05f5e100
1240a8: 3665 0500           |0018: if-gt v5, v6, +0005
1240ac: d805 0501           |001a: add-int/lit8 v5, v5, #01
1240b0: 28f9                |001c: goto -0007
```

図 4 Java プログラムより生成された DEX バイトコード

```
1240a0: 1215                |0014: const/4 v5, #1
1240a2: 1406 00e1 f505      |0015: const v6, #05f5e100
1240a8: 3665 0500           |0018: if-gt v5, v6, +0005
1240ac: d805 0501           |001a: add-int/lit8 v5, v5, #01
1240b0: 28fc                |001c: goto -0004
```

図 5 既存手法における改変後の DEX バイトコード

```
1240a0: 1405 00e1 f505      |0014: const v5, #05f5e100
1240a6: 1216                |0017: const/4 v6, #1
1240a8: 1217                |0018: const/4 v7, #1
1240aa: 3657 0400           |0019: if-gt v7, v5, +0004
1240ae: b067                |001b: add-int/2addr v7, v6
1240b0: 28fd                |001c: goto -0003
```

図 6 提案手法における改変後の DEX バイトコード

の DEX バイトコードの for 文処理性能を Kotlin 由来のバイトコードと同等の性能とする手法を提案する. 具体的には, 性能が劣っている Java 由来の DEX バイトコードにおける for 文処理の命令群を, 性能が勝っている Kotlin 由来の DEX バイトコードのものと同一(使用するレジスタの名前を除き同一)に変換し，これにより Java 由来の DEX バイトコードアプリケーションの性能を Kotlin 由来のそれと同等とする手法を提案する.

図 4 の変換結果は図 6 の様になる．goto 命令のジャンプ先を 6 バイト前に変更している.

4. 性能評価

前章の提案手法をハンドトランスレートにより実現し，提案手法の性能を評価した.

図 1 に示した Java および Kotlin で記述したソースコードより，Android アプリケーションを作成し，Android 端末上で実行しその性能を評価した．Java 由来のアプリケーションに関しては, 無変換のもの(通常手法), 既存手法により変換したもの，本稿の提案手法により変換したものを用いて評価を行った. 測定は Pixel 3a (CPU Qualcomm Snapdragon 670，メモリ 4GB)，Android 9.0 上で行った．開発およびコンパイルは, Android Studio 3.4.1, Open JDK 1.8.0_152, Kotlin 1.3.31 を用いて行った.

図 2 に Java を用いた場合の通常手法，既存手法，提案手法の実行時間と，Kotlin を用いた場合の実行時間を示す. 図より, 提案手法における実行時間は通常手法より 7.4%短く，既存手法と比較しても 2.1%短いことが分かる．また Kotlin 由来コードと，提案手法適用後の Java 由来コードの性能差はほぼなくなり(0.1%未満)，両言語の処理系に依存する性能差を Java 言語アプリケーションに活用できていることが分かる.

5. おわりに

本稿では JIT なし ART 環境下での Java と Kotlin の for 文処理における性能差を比較し，その差が DEX バイトコードの差異により生じることを示した．そして，性能の劣る Java 由来の DEX バイトコードアプリケーションを性能の勝る Kotlin 由来のバイトコードと同等に改変することにより，Java 由来のアプリケーションの性能を改善する手法を提案した．今後は DEX コンパイラの改変によるアプリケーション性能の改善手法の実装を行う予定である.

謝辞

本研究は JSPS 科研費 17K00109, 18K11277 の助成を受けたものである.

本研究は，JST，CREST JPMJCR1503 の支援を受けたものである.

参考文献

[1] Banerjee, M., Bose, S., Kundu, A., & Mukherjee, M. (2018). “A COMPARATIVE STUDY: JAVA VS KOTLIN PROGRAMMING IN ANDROID APPLICATION DEVELOPMENT”. International Journal of Advanced Research in Computer Science, 9(3), 41-45. doi:https://doi.org/10.26483/ijarcs.v9i3.5978

[2] Ryoichi Shibata, Yusuke Sato, Masato Oguchi, Akira Fukuda, Takeshi Kamiyama, Saneyasu Yamaguchi, “A Study on Compiler Dependent Performance Improvement”, HPC Asia 2020, Fukuoka Japan, Jan. 2020.