# コンパイラのアーキテクチャ依存部の最適化性能のランダムテスト

Random Testing of Optimization Performance of Architecture-Dependent Part of Compilers

牟田 亘希 [1] 石浦 菜岐佐 [2]
Koki Muta Nagisa Ishiura

関西学院大学 大学院理工学研究科 [1]
Graduate School of Science and Technology, Kwansei Gakuin Univ.

関西学院大学 工学部 [2]
School of Engineering, Kwansei Gakuin Univ.

## 1 はじめに

新しい CPU に対するコンパイラ開発では, アーキテクチャ依存部 (バックエンド) のみを新規実装することが多いが, この際にはバックエンドが正しいコードを生成しているかだけでなく, 適切な最適化を行っているかも重要なテスト項目となる. コンパイラの最適化不足を検出する手法としては, 文献 [1] のランダムテスト手法等が提案されているが, これらの手法ではミドルエンドとバックエンドの最適化不足が混在して検出されてしまう. 本稿では, コンパイラのバックエンドの最適化不足のみを検出する手法を提案する.

## 2 コンパイラの最適化性能のランダムテスト

文献 [1] の処理の流れを図 1 の点線部分に示す. ランダムに生成したプログラムを複数のコンパイラでコンパイルし, 生成したアセンブリコードを比較することによって一方の最適化不足を検出する. しかし, GCC や LLVM ではミドルエンドの最適化不足全てが直ちに修正されるとは限らず, ランダムテストが検出する多数の最適化不足のケースの中にバックエンドに起因するものが埋没してしまうという課題がある.

## 3 バックエンドの最適化性能のランダムテスト

本稿では, バックエンドのみが異なる複数のコンパイラで同一のプログラムをコンパイルし, 1 つのコンパイラでのみ最適化不足が生じるようなケースを検出することによって, バックエンドの最適化不足を検出する手法を提案する.

x86 用の GCC-8 を対象とする場合の処理の流れを図 1 に示す. 従来法により x86 用 GCC-8 の最適化不足のテストを行うとともに, ARM 用と MIPS 用の GCC-8 に対しても同じテストを行う. x86 用の GCC-8 でのみ最適化不足が検出された場合, x86 用のバックエンドに起因する可能性が高いと考えられる.

当該事象を検出したプログラム (エラープログラム) は最小化 (当該事象が起こる限りプログラムを縮約) した後, 実際にバックエンドで最適化不足が生じているかを分析する.

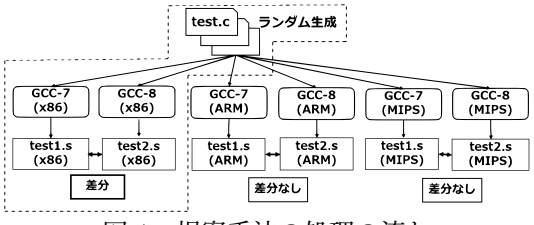


図 1 提案手法の処理の流れ

## 4 実験結果

提案手法を C コンパイラのランダムテストシステム Orange4 [2] を用いて実装し, x86, ARM, MIPS の GCC-8 を対象に実験を行った. 結果を表 1 に示す. “#test” は総テスト数, “#diff” は 1 つのコンパイラでのみ最適化不足を検出したプログラム数, “#diff (最小化)” は自動最小化後でも同じ結果が得られたプログラム数, “#error” は最終的にバックエンドの最適化不足と判定したプログラム数である. x86 用と ARM 用でそれぞれ 1 件バックエンドの最適化不足を検出できた.

図 2 に ARM 用 GCC-8 のエラープログラムを示す. 乗算命令が ARM 用の方にのみ生成されている.

表 1 実験結果 (GCC-8)

| target | #test | #diff | #diff (最小化) | #error |
|---|---|---|---|---|
| x86 | 3,000 | 24 | 10 | 1 |
| ARM | 3,000 | 24 | 8 | 1 |
| MIPS | 3,000 | 12 | 4 | 0 |

(Intel Core i5-6200U CPU@2.30GHz, Ubuntu 18.04 LTS)

test529.c

```
1:  int main()
2:  {
3:    volatile signed short x1 = 1;
4:    volatile unsigned char x2 = 1;
5:
6:    long t1 = (12358<(1/(5%(x1/5))))*x2;
7:
8:    return 0;
9:  }
```

529-ARM-1.s (GCC-7)

```
1:  ldrh r3, [sp, #6]
2:  ldrb r3, [sp, #5]
3:
4:
5:
6:
7:
8:
9:
```

529-ARM-2.s (GCC-8)

```
1:  movw r3, #41121
2:  ldrh r1, [sp, #6]
3:  movt r3, 41120
4:  ldrb r4, [sp, #5]
5:  sxth r1, r1
6:  uxtb r4, r4
7:  smull r2, r3, r3, r1
8:  asrs r2, r1, #31
9:  add r3, r3, r1
```

529-x86-1.s (GCC-7)

```
1:  movl $102, %eax
2:  movw %ax, -2(%rsp)
3:  movb $3, -3(%rsp)
4:  movzwl -2(%rsp), %eax
5:  movzbl -3(%rsp), %eax
6:  xorl %eax, %eax
```

529-x86-2.s (GCC-8)

```
1:  movl $102, %eax
2:  movw %ax, -10(%rsp)
3:  movb $3, -11(%rsp)
4:  movzwl -10(%rsp), %eax
5:  movzbl -11(%rsp), %eax
6:  xorl %eax, %eax
```

529-MIPS-1.s (GCC-7)

```
1:  daddiu $sp,$sp,-16
2:  li $2,102 # 0x66
3:  sh $2,14($sp)
4:  li $2,3 # 0x3
5:  sb $2,13($sp)
6:  lhu $2,14($sp)
```

529-MIPS-2.s (GCC-8)

```
1:  daddiu $sp,$sp,-32
2:  sd $31,24($sp)
3:  sd $28,16($sp)
4:  li $2,102 # 0x66
5:  ld $31,24($sp)
6:  sh $2,14($sp)
```

図 2 ARM 用バックエンドの最適化不足

## 5 むすび

本稿では, コンパイラのバックエンドの最適化不足のみを検出する手法を提案した. 1 つのコンパイラでのみ最適化不足が生じる状態を維持したまま最小化を行うことが今後の課題である.

### 参考文献

[1] K. Kitaura and N. Ishiura: “Random testing of compilers' performance based on mixed static and dynamic code comparison,” in *Proc. A-TEST 2018*, pp. 38–44 (Nov. 2018).

[2] K. Nakamura and N. Ishiura: “Random testing of C compilers based on test program generation by equivalence transformation,” in *Proc. APCCAS 2016*, pp. 676–679 (Oct. 2016).