コンパイラ基盤 LLVM バックエンドのランダムテスト

Randomtest of Backend of Compiler Infrastructure LLVM

田中健司 [1]　石浦菜岐佐 [1]　西村啓成 [2]　福井昭也 [2]
Kenji Tanaka　Nagisa Ishiura　Masanari Nishimura　Akiya Fukui

関西学院大学 理工学部 [1]　School of Science and Technology, Kwansei Gakuin University
ルネサスシステムデザイン株式会社 [2]　Renesas System Design Co., Ltd.

1　はじめに

コンパイラには非常に高い信頼性が求められるため, テストスイートやランダムテスト等による徹底的なテストが行われる. コンパイラ基盤 LLVM [1] は, 言語依存のフロントエンドとターゲット依存のバックエンドが分離された構造を持つため, バックエンドを開発すれば新しいプロセッサに対するコンパイラを作成できる. その際には, バックエンドを対象としたテストを重点的に行うことが重要となる. そこで本稿では, ランダムに生成した LLVM の中間表現により直接バックエンドのテストを行う手法を提案する.

2　コンパイラとそのランダムテスト

LLVM は, プログラムから中間表現 LLVM-IR を生成するフロントエンドと, LLVM-IR を最適化するミドルエンド, LLVM-IR から機械語を生成するバックエンドから成る. 新しいプロセッサのコンパイラを開発した際には, バックエンドを重点的にテストする必要がある.

コンパイラのランダムテストは, ランダムに生成したプログラムのコンパイルと実行を繰り返すことによりコンパイラをテストする手法である. Orange3 [2] は, C コンパイラの算術最適化をターゲットとするランダムテストシステムである. Orange3 が生成するプログラムは, 変数の宣言と初期化, 算術式を右辺に持つ代入文, および計算結果の照合を行う文により構成される.

3　LLVM バックエンドのランダムテスト

C 言語では, int 型が 32bit の場合, 8bit や 16bit の整数型に対する演算は, 汎整数拡張により 32bit の演算に変換されるため, バックエンドの入力に 32bit 未満の演算が出現するのは, ターゲット依存の最適化が施された場合に限られる. このため, C プログラムによるテストでは, このような演算に対するバックエンドのテストが十分に行えない可能性がある. 本稿で提案するテスト手法は, LLVM-IR のテストプログラムを直接生成することによって, この課題を解決することを狙いとしている.

本手法では Orange3 が生成するプログラムの中間表現 (抽象構文木) から LLVM-IR のテストプログラムを生成する. Orange3 では, C 言語の汎整数拡張に基づく型変換を中間表現生成時に行っているが, この部分を修正し, int 型の幅未満の演算が生成されるようにする. 本手法のテストプログラムの例を図 1 に示す. 1, 2, 7, 8 行目で変数の初期化を, 9〜12 行目で演算を行い, 13〜19 行目で結果の照合を行う.

4　実験結果

提案手法に基づくテストシステムを, Orange3 を拡張することにより実装した. 対応する LLVM のバージョンは 3.5 である. 表 1 は, x86_64-apple-macosx10.11.0 (64bit) をターゲットに生成した演算数 100 のテストプログラム 100 個中に含まれる各演算の出現回数である.

```
 1: @x0 = internal global i16 20, align 2
 2: @x3 = constant i16 30, align 2
 3: @.str = private unnamed_addr constant [6 x i8] c "@OK@\0A\00
    ", align 1
 4: @.str1 = private unnamed_addr constant [18 x i8] c "@NG@ (te
    st = %hhd)\0A\00", align 1
 5: ; Function Attrs: nounwind uwtable
 6: define i32 @main() #0{
 7:   %def_t0 = alloca i16, align 2
 8:   store i16 10, i16* %def_t0, align 2
 9:   %l_t0 = load i16* %def_t0, align 2
10:   %l_x0 = load i16* @x0, align 2
11:   %l_x3 = load i16* @x3, align 2
12:   %t0 = add i16 %l_x3, %l_x0
13:   %cp_t0 = icmp eq i32 %t0, 50
14:   br i1 %cp_t0, label %2, label %3
15: ; <label>:2
16:   %true_pr0 = call i32 (i8*, ...)* @printf(i8* getelementptr
      inbounds ([6 x i8]* @.str, i32 0, i32 0))
17:   br label %4
18: ; <label>:3
19:   %ng_pr0 = call i32 (i8*, ...)* @printf(i8* getelementptr i
      nbounds ([18 x i8]* @.str1, i32 0, i32 0), i32 %t0)
20:   br label %4
21: ; <label>:4
22:     ret i32 0
23: }
```

図 1　LLVM-IR テストプログラムの例

i8, i16 はそれぞれ 8bit, 16bit の整数型演算であり, LLVM は本手法, C (-O0), C (-O3) はそれぞれ Orange3 が生成する C プログラムを -O0, -O3 オプションでコンパイルした結果である. 本手法では, C プログラムによるテストでは出現頻度が低い演算のテストが行えている.

表 1　演算の出現数

	i8 (8bit 整数型)			i16 (16bit 型整数型)		
	LLVM	C(-O0)	C(-O3)	LLVM	C(-O0)	C(-O3)
add	791	0	4	1708	0	6
and	0	0	3	0	0	2
icmp eq	1043	0	195	4013	0	208
icmp ne	76	0	95	356	0	129
icmp sge	20	0	5	122	0	6
icmp sgt	17	0	62	115	0	86
icmp sle	23	0	4	134	0	11
icmp slt	20	0	70	118	0	105
icmp uge	62	0	5	203	0	8
icmp ugt	72	0	41	203	0	72
icmp ule	68	0	3	176	0	5
icmp ult	74	0	51	213	0	74
lshr	0	0	40	0	0	35
mul	94	0	0	322	0	0
sdiv	26	0	0	143	0	0
srem	25	0	0	128	0	0
sub	77	0	0	320	0	0
udiv	76	0	18	213	0	16
urem	70	0	20	199	0	22

5　むすび

本稿では, LLVM-IR を直接生成するランダムテスト手法を提案した. 本手法で生成されにくい演算の生成が今後の課題である. 本研究は一部科学研究費補助金 (25330073) による.

参考文献

[1] http://llvm.org/.

[2] E. Nagai, A. Hashimoto, and N. Ishiura: "Reinforcing random testing of arithmetic optimization of C compilers by scaling up size and number of expressions," *IPSJ Trans. SLDM*, vol. 7, pp. 91–100 (Aug. 2014).