コンパイラの最適化性能テストにおける畳み込み可能な部分式のトップダウン生成

Top-down Generation of Foldable Subexpressions for Performance Testing of Compilers

樋口 瑛子 Yoko Higuchi
石浦 菜岐佐 Nagisa Ishiura

関西学院大学 理工学部
School of Science and Technology, Kwansei Gakuin University

1 はじめに

コンパイラには高い最適化の性能が求められるため, コンパイラのテストにおいては, 意図通りの最適化が行われているかどうかも重要なテスト項目になる. 文献 [1] は, 2 つの等価なテストプログラムをランダムに生成し, コンパイルの結果得られるアセンブリコードを比較することによって, コンパイラの最適化不足を検出する手法を提案している. しかし, この手法では最適化が適応可能な部分式をボトムアップに生成するため, 部分式の大きさや形状が制御できない. これに対し, 本稿では, 最適化対象となる部分式をトップダウンに生成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] の手法では, ランダムに生成した解析木から図 1(a) のようなプログラムを生成する一方で, 解析木に対して定数畳み込みの最適化処理をボトムアップに行った別のプログラム (図 1(b)) を生成し, コンパイルの結果得られる 2 つのアセンブリコードを比較することによって, コンパイラの最適化不足を検出する. volatile 修飾されていない変数を葉とする部分木のみが畳み込みの対象となる. [1] では, 変数の volatile 修飾は, 式の生成と独立してランダムに決定しているため, 対象となる部分式の大きさや形状が制御できていなかった.

```
01: #include <stdio.h>
02: #define OK()  printf("@OK@\n")
03: #define NG(fmt,val)  printf("@NG@ (test = "fmt")\n",val)
04:
05: unsigned int x10 = 2U;
06: static volatile signed int x11 = 3;
07: volatile signed char t0 = 15;
08: int main (void)
09: {
10:   signed short t1 = 190;
11:   unsigned long long x9 = 2LLU;
12:   static unsigned long long x15 = 1LLU;
13:   static const unsigned long long x16 = 3LLU;
14:   static volatile signed char x18 = 1;
15:   const signed short x20 = 2;
16:   t0 = ((signed char)((x11)*((signed int)(x9*x10))));
17:   t1 = ((signed short)(((signed char)(x15*x16))<<x18));
18:   if (t0 == 12) { OK(); } else { NG("t0", "%d", t0); }
19:   if (t1 == 8) { OK(); } else { NG("t1", "%hhd", t1); }
20:   return 0;
21: }
```

(a) 生成するテストプログラム

```
        ...
16:   t0 = ((signed char)(((x11)*(signed int)3)));
17:   t1 = ((signed short)(((signed char)3<<x18));
        ...
```

(b) 定数畳み込みを行ったプログラム

図 1 最適化の性能テストのための等価プログラム

3 最適化可能な部分式のトップダウン生成

本稿では, 最適化可能な部分式をトップダウンに生成する手法を提案する. 図 1 の 16–17 行目の代入文の右辺の算術式をトップダウンに生成する際, 各節点を v または n でラベル付けする. 根の節点は v とし (図 2(a)), v の節点の子は少なくとも一方は v であり, n の節点の子は全ての子が n となるようにラベルをランダムに決定する (図 2(b)). 葉の節点は変数に対応するが, ラベルが v であれば volatile 変数, n であれば非 volatile 変数とする. (b) から 1 つのプログラムを生成し, さらに n の節点に定数畳み込みを適用したもの (図 2(c)) からもう 1 つのプログラムを生成する.

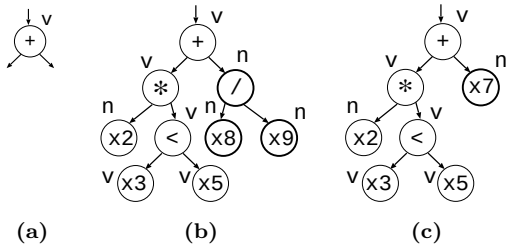

図 2 トップダウンによる volatile 変数の生成

4 実装結果

提案手法を Perl5 で実装し, GCC-8.0.0 の -O3 オプションのテストを 1 時間行った. アセンブリコードの比較方法はいずれも文献 [2] の手法で行い, 不一致判定のしきい値は 70%に設定した. “vv” は v 節点の子を両方とも v 節点にする確率であり, “rate” はプログラム中の volatile 変数の比率, “#test” はテスト総数, “#error” は検出したエラー数を示す. 従来法ではエラーを検出できなかったが, 提案手法では volatile 変数の比率が従来法より高くてもエラーを検出できた. 現在, このエラーの原因を分析中である.

表 1 テスト結果 (GCC-8.0.0, -O3 オプション)

	vv	rate	#test	#error
ボトムアップ (従来法)		0.426	17,669	0
トップダウン (本手法)	0.750	0.453	7,196	328
	0.875	0.594	6,492	89

実行時間: 1h (Intel Core i7-5500U@2.40GHz×4, 7.7GiB)

5 むすび

本稿では, コンパイラの最適化性能テストのためのプログラム生成の一手法を提案した. 検出したエラーの原因分析が今後の課題である.

参考文献

[1] 橋本, 石浦: “ランダムテストによる C コンパイラの算術最適化機会の検出,” 信学技報, VLD2014-139 (Jan. 2015).
[2] 北浦, 岩辻, 石浦: “C コンパイラの最適化のリグレッションテストのためのアセンブリコード比較法,” 信学ソ大, A-6-12 (Sept. 2016).

Copyright © 2017 IEICE