Cコンパイラの算術最適化のランダムテストにおける浮動小数点演算の導入

Incorporating Floating Point Operations into Random Testing of Arithmetic Optimization of C Compilers

橋本淳史 Atsushi Hashimoto
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

コンパイラには非常に高い信頼性が要求されるが, 徹底的なテストを経ても, なお不具合が潜在することがある. ランダムテストは, ランダムに生成したプログラムによるテストを繰り返し実行することにより, このような不具合を検出する手法である. Csmith [1] は GCC や LLVM の不具合を 325 件以上報告している. また, 算術最適化を対象とした永井らの手法 [2] も最新版の GCC の不具合を検出している. しかし, いずれの手法も対象とする演算は整数演算に限られていた. 本稿では, 算術最適化のランダムテストへの浮動小数点演算の導入を提案する.

2 ランダムテストにおける浮動小数点演算の課題

C 言語の仕様 [3] では, 浮動小数点演算の結果の値, 算術型変換の結果の値, および浮動小数点定数の値は, その型よりも高い精度で計算してもよいとされている. すなわち, float 型の演算が, 状況によって double 型や long double 型の精度で計算されることがあり, プログラムの指定どおりに期待値を計算しても, 実行値との間に誤差が生じる. 特に, 浮動小数点型から整数型への変換を含むプログラムの場合には, エラーの判定に誤りが頻発することが問題となる.

3 浮動小数点演算の導入手法

本稿では, 下記 (1)〜(4) により前節の問題を解決する.

(1) 浮動小数点型の値は, その型の仮数部のビット数を M として, -2^M〜2^M の範囲の整数値に限定する.

(2) 演算結果が (1) の範囲を超える場合, 図 1 のように加算を挿入して, 値を (1) の範囲に収める.

(3) 割り切れない除算の場合, 非除数に (2) と同様の処理を行って, 割り切れるように修正する.

(4) 整数型から浮動小数点型への型変換により値が (1) の範囲を超える場合も, (2) と同様の処理を行って, 値を (1) の範囲に収める.

```
float x = 12723.0F;

float y = x * 717.0F;
if (y!=9122391.0F) error();
```

⇒

```
float x = 12723.0F;
float z = -9802.0F;
float y = (x + z) * 717.0F;
if (y!=2094357.0F) error();
```

図 1 加算演算子の挿入

4 実装結果

本手法を [2] のランダムテストに組み込み, 実験を行った結果を表 1 に示す. 実行時間は 12 時間である. “Integer” は整数演算のみの従来法, “Integer & Floating” は, 本手法の浮動小数点演算を含むテストの結果である. “test” はテスト総数, “error” は検出したエラー数, () 内は, 浮動小数点型を含むエラー数を示す. 従来法に比べてエラーの検出数は減少したが, 従来法で検出できなかった不具合を検出することができた. 図 2 は, 本手法で LLVM-3.3+Clang の内部エラーを検出したプログラムであり, 11 行目に浮動小数点演算を含む.

表 1 実験結果

Compiler (Target)	Integer		Integer & Floating	
	test	error	test	error
gcc-4.5.4 (i686-pc-linux)	21,342	87 (0)	18,031	43 (30)
gcc-4.4.4 (i686-pc-linux)	20,945	54 (0)	18,608	44 (34)
gcc-4.4.1 (arm-elf)	18,118	40 (0)	16,450	19 (3)

12 hours (Ubuntu 12.04, Core i5-2540M 2.60GHz, RAM 8GB)

```
 1: #include <stdio.h>
 2: #define OK() printf("@OK@\n")
 3: #define NG(fmt,val) printf("@NG@ (t = "fmt")\n",val)
 4: double x14 = 3511269280748732.0;
 5: int k132 = 1199736362;
 6: int main (void)
 7: {
 8:  volatile unsigned int x68 = 1U;
 9:  volatile int x106 = 1;
10:  unsigned int t4 = 1U;
11:  t4 = ((((unsigned)1U-((unsigned)1U>((unsigned)1U*
     ((unsigned)2U/(((-1)+k132)<<(0<<(((int)(x14+
     (double)-3511267518266169.0))+(-1762482553)))))))
     /x106)>>((-1%(x14!=x68))/1));
12:  if (t4 == 0U) { OK(); } else { NG("%u", t4); }
13:  return 0;
14: }
```

図 2 LLVM-3.3+Clang の内部エラーを検出したプログラム

5 むすび

本稿では, Cコンパイラの算術最適化のランダムテストへの浮動小数点演算の適用手法を提案した. 本手法により, 従来では検出できなかったエラーを検出できた. 構造体や関数呼び出しの導入が, 今後の課題である.

参考文献

[1] X. Yang, et al.: “Finding and Understanding Bugs in C Compilers,” in Proc. PLDI, pp. 283–294 (June 2011).

[2] 永井, 橋本, 石浦: “C コンパイラの算術最適化のランダムテストにおける式生成の強化,” 信学技報, VLD2012-117 (Jan. 2013).

[3] 日本規格協会: JIS X 3010:2003（ISO/IEC 9899:1999）プログラム言語 C, (Dec. 2003).