社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

プログラム併合によるコンパイラのリグレッションテストの高速化

森本　和志† 　石浦菜岐佐† 　内山　裕貴†† 　引地　信之†††

† 関西学院大学 理工学部 〒669–1337 兵庫県三田市学園 2-1
†† 株式会社ケイ・オプティコム 〒530–6116　大阪市北区中之島 3 丁目 3 番 23 号　中之島ダイビル
††† 株式会社 SRA 〒171–8513 東京都豊島区南池袋 2-32-8

あらまし　本稿では, テストスイート中のテストプログラムの併合により, コンパイラのリグレッションテストを高速化する手法を提案する. コンパイラのテストスイートは膨大な数のテストプログラムから構成されるため, これらのコンパイルと実行には長大な計算時間を要する. 特に, コンパイラの開発段階では, コンパイラの修正とリグレッションテストを繰り返し行うため, テストの高速化は非常に重要な課題となる. これに対し本稿では, 複数のテストプログラムを併合することによって, できる限り元のプログラムの意図を維持したままテストの実行時間を短縮する手法を提案する. プログラムの併合に際しては, グローバル変数, 関数, および typedef の識別子の衝突の他, ヘッダファイルの処理, 分割コンパイルへの対応等が問題となるが, これらを解決する. 本手法を testgen テストスイートに適用し, 約 9,000 本のテストプログラムを 117 本に併合した結果, テスト実行時間を 2.53GHz の Core i5 (メモリ 2GB) 上の Windows Cygwin で約 1/44.2, Linux (Ubuntu) で約 1/7.7 に削減することができた.

キーワード　C 言語, コンパイラ, テストスイート, gcc, プログラム併合, testgen

Acceleration of Regression Test of Compilers by Program Merging

Kazushi MORIMOTO†, Nagisa ISHIURA†, Yuki UCHIYAMA††, and Nobuyuki HIKICHI†††

† Kwansei Gakuin University, Gakuen 2-1, Sanda, Hyogo, 669–1337 Japan
†† K-Opticom Corporation, Nakanoshima Dai Bldg. 3-3-23, Kita-ku, Osaka, 530–6116 Japan
††† Software Research Associates, Inc., Minami Ikebukuro 2-32-8, Toshima-ku, Tokyo, 171–8513 Japan

Abstract　This article presents a method of accelerating regression test of compilers by merging programs in test suites. Testing of compilers needs a large amount of computation time, since test suites of compilers usually consist of a huge number of test programs. Especially, in the early stages of compiler development, acceleration of test is an important issue, for bug fixes and regression test are alternately repeated for many times. The proposed method attempts to merge test programs in the test suites into longer programs so as to reduce the time for file open/close. During the merging, conflicts among the names of global variables, functions, and "typedef" declarations, as well as header file inclusion and separate compilation, are carefully handled so that the semantics of the original programs are maintained. In an experiment where about 9,000 test programs in *testgen* test suite were merged into 117 programs, computation time was reduced into 1/44.2 on Windows Cygwin and into 1/7.7 on Linux (Ubuntu) on 2.53GHz Core i5 with 2GB memory.

Key words　C language, compiler, test suite, gcc, program merging, testgen

1. はじめに

ソフトウェアを開発するための基盤ツールとして, コンパイラには極めて高い信頼性が要求される. このためコンパイラの開発に際しては, テストスイート, コンパイラ自身や OS のカーネル, あるいはランダムに生成したプログラムを用いた徹底的なテストが行われる.

コンパイラのテストスイートは, コンパイラの各機能をテストするプログラムの集合であり, これらをコンパイルし実行して意図通りの結果が得られるかどうかを確認することによりテストを行う. コンパイラが対象とするプログラミング言語の多様な機能をできる限り網羅するため, テストスイートは膨大な数のテストプログラムを含む. その数は数千から数十万におよび, テストの実行には長大な時間が必要になる.

テストの実行が最終的な動作確認だけで済むのであれば, その計算時間は大きな問題にならない. しかし, 実際にエラーが検出された場合には, 不具合の修正後に再度テストを行う必要が生じる. 特に, コンパイラの開発段階では, 1つの不具合の修正が新たな不具合を引き起こさないかの確認が必要であったり, テストスイートによるエラー数の削減を指標に開発を進めるため, コンパイラの修正とテストの実行を頻繁に行うことになる. このため, テストスイートによるテストの実行時間の短縮は非常に重要な課題となる.

そこで本稿では, 複数のテストプログラムを併合することにより, ファイルのオープン/クローズの回数を減らし, テストを高速化する手法を提案する. また, プログラムを併合する際に問題となる, グローバル変数, 関数, typede の識別子の衝突, ヘッダファイルの処理, 期待値計算, 分割コンパイルの処理等の解決手法を提案する.

本手法を実装し, C コンパイラ用テストスイートである testgen テストスイートに適用した. その結果, testgen テストスイートと比較し, テスト実行時間を Windows Cygwin で平均 1/44.2, Ubuntu で平均 1/7.7 に短縮できた.

2. C コンパイラ用テストスイート

コンパイラのテストは, コンパイラが対象とする言語で書かれたプログラムをコンパイルし, 得られた目的プログラムを実行して期待通りの結果が得られるかどうかを確認することにより行える. テストに用いるプログラムをテストプログラムと呼び, テストプログラムの集合をテストスイートと呼ぶ(注1). テストプログラムに対して, 入力データと期待される出力データの対を準備することもあるが, テストプログラムの中で入力データの準備や期待値との照合まで行う場合が多い.

通常, コンパイラのテストスイートは非常に多くのテストプログラムを含む. その理由は, コンパイラが受け付けるプログラミング言語の多様な機能をできる限り網羅するためだが, さらに, テストでエラーが発見された際に, その原因を突き止め易くするためでもある. すなわち, テストプログラムでエラーが検出された場合には, 多くの機能の検査を含む長いテストプログラムよりも, 特定の項目を検査するできるだけ短いプログラムの方が, コンパイラの不具合の原因を特定するのに有用だからである. しかし, テストプログラムの総行数が同じであっても, テストプログラムの数が増えると, ファイルのオープン/クローズ処理の回数が増えるため, テストに要する時間が長くなる.

GCC (GNU Compiler Collection) 付属のテストスイートは, 約 2,500 個のファイルから構成され, テストフレームワークとして GNU ソフトウェアの 1 つである dejagnu を用いる. このテストスイートは GCC の幅広い機能をテストするテストプログラムから構成され, リリース前など, 完成度が高いコンパイラの動作確認テストに適している.

コンパイラの徹底的な品質向上を目的としたテストスイートは, 膨大な数のテストプログラムを含む. 例えば, [1] の商用のコンパイラ評価サービスで用いられるテストスイートのテストプログラム数は約 29 万本にもおよび, テストの実行時間も長大となる.

一方で, testgen [2] のように, コンパイラの開発段階での使用を想定したテストスイートも存在する. testgen テストスイートは, 基本的な構文の機能をテストする約 9,000 本のテストからなる. 図 1 に testgen テストスイートのプログラムの一例を示す. このプログラムでは, 静的変数への代入が正しく行えるかどうかをテストしており, 結果が正しい/誤っている場合には, printok()/printno() によりその旨を表わす文字列を出力する(注2).

```
 1: #ifdef SYSDEP_H
 2: #include "sysdep.h"
 3: #endif
 4: #include "stnd.h"
 5:
 6: main()
 7: {
 8:   static int Variable;
 9:
10:   itest = 0;
11:   Variable = 1;
12:   itest = Variable;
13:
14:   if (itest == 1)
15:      printok();
16:   else
17:      printno();
18:
19:   return (0);
20: }
```

図 1　テストプログラムの例

testgen テストスイートの全テストプログラムの実行に要する時間は, Core i5, CPU 2.53GHz, メモリ 2GB の環境で, Windows Cygwin で約 177 分 0.8 秒, Ubuntu で約 13 分 36.8 秒である. コンパイラの動作確認として 1 度だけテストを行うのであれば問題にならない時間だが, 開発段階で繰り返しテストを行う場合には, 無視できなくなる. 即ち, まだコンパイラの完成度が高くない段階では, コンパイラの修正をしてはテストを行うという処理が繰り返し行われる. 1 つの修正作業によって別の不具合が生じることもあるため, 過去にパスしたテストプログラムの動作も繰り返し確認する必要がある. GCC のマシン記述 (コンパイラの機械依存部の記述) を修正してビルド (コンパイルしてリンクする処理) に要する時間は Windows Cygwin で約 206 分 35.5 秒, Ubuntu で約 11 分 15.3 秒であり, ビルドとその後のテストにほぼ同程度の時間を費やすことになる. これを繰り返す開発過程では, テストの高速化が非常に重要な課題となる.

3. テストプログラムの併合によるテスト高速化

本稿では, テストスイートを用いたコンパイラのリグレッションテストの高速化手法として, テストプログラムの併合に基づ

(注1)：テストスイートには, テストの実行に必要なデータや, テストの実行, 期待値の照合処理, 結果集計を行うためのスクリプトまで含めることもある.

(注2)：printok()/printno() はユーザ定義可能なマクロであり, 標準入出力ライブラリが使用できない環境では, 他の手段を用いてテストの結果をユーザに報告することができる.

く手法を提案する．

図 2 に提案手法の概要を示す．ディレクトリ test-A 内にテストプログラム t001.c, t002.c, t003.c, t004.c があるとする．これを併合したものが test-A.c である．test-A.c は，元のテストプログラムの main 関数を各々1 つの関数に改名し，新たな main 関数からこれらを順次に呼び出す．test-A.c は機能的には元の 4 つのプログラムと等価であるが，ファイルのオープン/クローズの回数が減るため，コンパイルと実行に要する時間は短くなる．

ただし，プログラムを併合する際には，名前の衝突，ヘッダファイルの処理，期待値の計算，分割コンパイルの処理等に留意する必要がある．

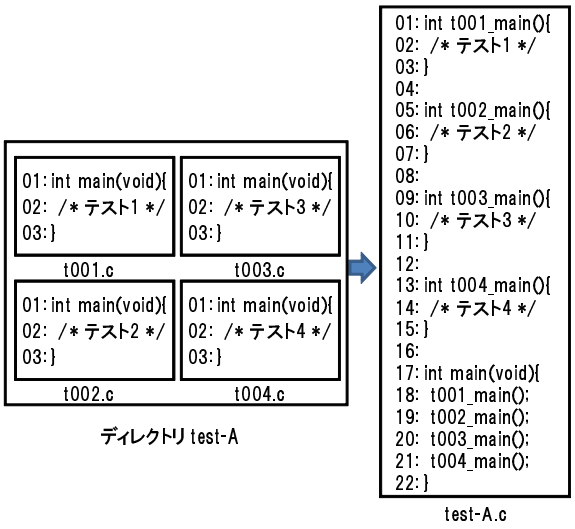


図 2　プログラム併合の概要

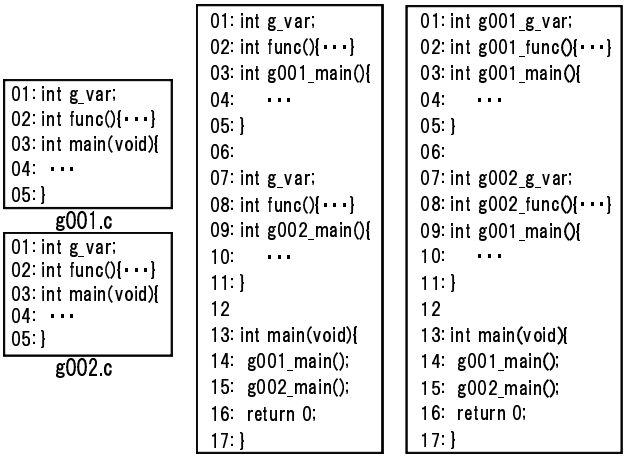
3.1　名前の衝突への対応

単純に図 2 のようなプログラムの併合を行うと，名前の衝突が起こる．問題となるのが，グローバル変数，関数，および typedef による型の定義である．図 3 はグローバル変数と関数の名前の衝突の例である．g001.c, g002.c はともにグローバル変数 g_var と関数 func を定義している．このプログラムの併合を単純に行うと (b) のようになるが，1, 7 行目でグローバル変数 g_var, 2, 8 行目で関数 func の名前が衝突する．g_var の定義を 1 つに統合した場合，g001_main や g002_main の中で g_var の初期化を行っていなければ，テストプログラムが正常な動作を行わなくなる可能性がある．また，C 言語の仕様ではグローバル変数の初期値は 0 と定めているので，これを前提としたテストが行えなくなってしまう．func 関数は g001.c と g002.c で定義されている．内容が同一であれば統合可能だが，一般にその保証はない．

そこで本手法では，変数名，関数名，typedef の定義にプレフィックスとして元テストプログラムの名前を付加することにより，この問題を解決する．図 3(c) にプレフィックス付加の例を示す．これにより，名前の衝突を回避し，テストプログラムの意図を維持することができる．

3.2　ヘッダファイルの処理

ヘッダファイルには，標準ヘッダファイルとユーザが定義し



(a) 併合前　　(b) 単純な併合　　(c) 名前の修正

図 3　プログラム併合の流れ

たヘッダファイルがある．ユーザ定義のヘッダファイルには，グローバル変数や関数などが記述されているため，これを展開して 3.1 節で述べた名前の変更処理を行う必要がある．一方，標準ヘッダファイルでは名前の衝突を考える必要がない．

そこで，ユーザ定義のヘッダファイルのみ展開して名前の変更処理を行い，標準ヘッダファイルについては併合後のプログラムに必要なすべてのインクルードを記述することにする．図 4 に例を示す．test-B 内にテストプログラム t001.c, t002.c があり，t001.c では標準ヘッダファイル stdio.h と math.h, t002.c では stdio.h の標準ヘッダファイルをそれぞれインクルードしている．また，2 つのプログラムはともにユーザ定義のヘッダファイル incfile.h をインクルードしている．このディレクトリ内のプログラムの併合を行った結果が test-B.c である．test-B.c の 4〜6 行はユーザヘッダファイルを展開し，名前にプレフィックス t001_を付加している．同様に 9〜11 行は t002_を付加している．標準ヘッダファイルは t001.c と t002.c でインクルードされている stdio.h と math.h を test-B.c でインクルードしている．

ユーザファイルの展開は，名前変更処理の前にプリプロセッサにより行う．ユーザヘッダファイルのみの展開は，例えば gcc の -E -nostdinc オプション等により行える．標準ヘッダファイルは，これらの処理の間にリストを作成し，併合後のファイルにそれらのインクルードを書きこめばよい．

3.3　期待値の処理

2 章で述べたように，コンパイラのテストには入力データや期待値データを持つものと持たないものが存在する．入力データや期待値データがある場合には，プログラムを併合したものと同じ順序でこれらのデータを結合したものを新たな入力データ/期待値とすればよい．また，testgen のように，特定の文字列の出力数をカウントしている場合には，その合計値を期待値とすればよい．

3.4　分割コンパイルの処理

テストスイートの中には分割コンパイルのテストを行うものがある．この場合には，1 つのプログラムファイルが複数の異な

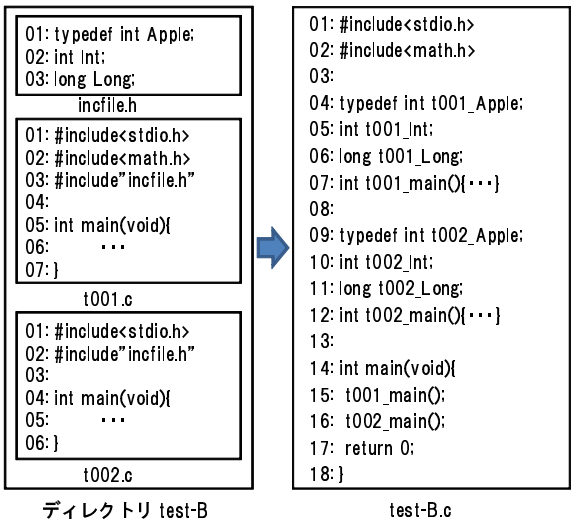


図 4　ヘッダファイル処理

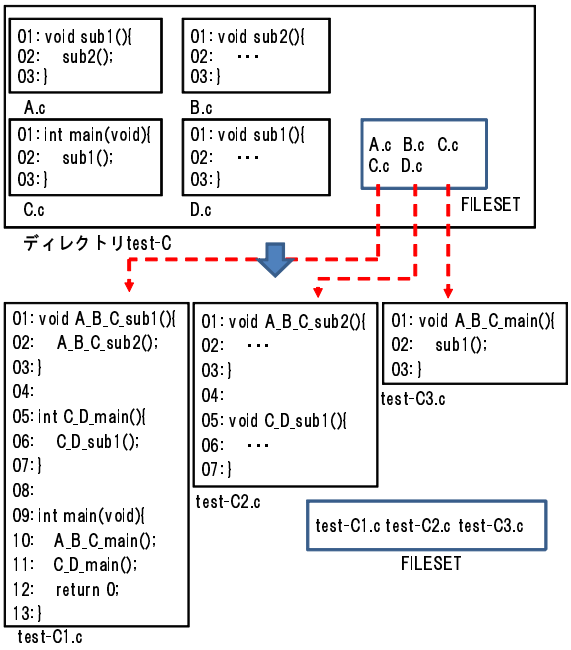


図 5　分割コンパイルテストの併合

るプログラムファイルの組み合わせでリンクされることがあるため, 3.1 節で述べた単純な名前の変更では対応できない. そこで, 分割コンパイルのテストが含まれる場合には, 結合されるプログラムの名前を結合したものを, 変数や関数のプレフィックスとして付加する.

図 5 に例を示す. ディレクトリ test-C 内に 4 つのテストプログラム (このうち C.c が main 関数を含んでいる) があるとする. どの組み合わせでプログラムをリンクしてテストするかは FILESET というファイルに指示されているものとする. この場合, 併合結果は test-C1.c, test-C2.c, test-C3.c という 3 つのプログラムファイル, および新たな FILESET ファイルとなる. test-C1.c, test-C2.c, test-C3.c は, それぞれ元の FILESET で指定された組み合わせに現れる, 1 番目, 2 番目, 3 番目のプログラムを併合したものである. 全体の main 関数はいずれのファイルに生成してもよい (この例では test-C1.c に生成している). この際, 呼び出される A.c, B.c, C.c の組み合わせには対応する変数や関数のすべてに A_B_C_というプレフィックスを付加する. 同様に C.c, D.c の組み合わせには C_D_というプレフィックスを付加する. これにより, 分割コンパイルのテストを含むテストスイートを併合できる.

4. 実　　装

提案手法に基づくテストプログラム併合スクリプトを Perl 5.10.1, gcc-4.3.4 で実装した. 併合スクリプトはテストスイートのディレクトリを指定するとその直下にあるディレクトリごとにディレクトリ内のテストプログラムを併合する. 実装および実験は,

- Windows7 Cygwin
- Ubuntu Linux

の下で行った (計算機は Core i5, CPU 2.53GHz, メモリ 2GB).

併合スクリプトを testgen テストスイートに適用し, 85 ディレクトリ, 約 9000 ファイルのプログラム群を 117 ファイルに併合した. ディレクトリ数に比べてファイル数の方が多いのは, 分割コンパイルのテストプログラムの併合を行っているためである. 併合に要した時間は Cygwin で約 37 分 53.7 秒, Ubuntu で約 1 分 45.0 秒である. テストの実行を x86 gcc-4.5.1 ネイティブコンパイラ, および arm gcc-4.5.1 クロスコンパイラで行い, 時間を比較した結果を表 1 と表 2 に示す. 実行時間は併合前と比べ x86 gcc では Windows Cygwin で約 1/44.7, Ubuntu で約 1/7.7, arm gcc では Windows Cygwin で約 1/43.7, Ubuntu で約 1/7.7 となった.

表 1　x86 gcc-4.5.1 実行時間の比較

	Cygwin	Ubuntu
併合前	182m12.6s	13m51.4s
併合後	4m4.8s (2.2%)	1m47.4s (12.9%)

表 2　arm gcc-4.5.1 実行時間の比較

	Cygwin	Ubuntu
併合前	177m0.8s	13m36.8s
併合後	4m2.9s (2.3%)	1m46.7s (13.1%)

併合後のテストプログラムの機能は, 原理的には併合前のプログラムと同じはずであるが, プログラムやデータが配置される番地, またはコンパイラの不具合によっては, 両者でテスト結果が一致しないことがあり得る. 実際に, arm gcc-4.5.1 の -O0 オプションで, 併合前テストスイートのテストはすべて PASS したにもかかわらず, 併合後のテストの 1 つでエラーが検出された. 逆に, エラーが発生するプログラム [3] をテストスイートに混入させて併合を行ってみたが, 数通りの実験を行った範囲では併合後のプログラムがエラーを見のがすケースは確認されなかった.

5. むすび

本研究ではテストプログラムの併合により, コンパイラのリグレッションテストを高速化する手法を提案した. 本手法を testgen テストスイートに適用した結果, テスト実行時間を Cygwin で平均 1/44.2, Ubuntu で平均 1/7.7 に削減できた.

今回実装した併合スクリプトは, [4] とともに GPL にて公開予定 [5] である.

謝　　辞

本研究を行う上で多くのご助言, ご協力頂いた関西学院大学石浦研究室の多賀惣一朗氏, 大城亮氏に感謝します. また, 本研究に関してご協力, ご討議頂いた関西学院大学石浦研究室の諸氏に感謝します.

文　　献

[1] http://www.jnovel.co.jp/service/compiler/index.html.
[2] 内山, 引地, 石浦, 永松: “C コンパイラ用テストスイートおよびその生成ツール testgen,” 信学技報 VLD2006-95 (Jan. 2007).
[3] 武田直哉, 粟津裕亘, 石浦菜岐佐: “定数畳み込みを対象とした 6C コンパイラのランダムテスト,” 情報処理学会関西支部支部大会, A-13 (Sept. 2009).
[4] “C コンパイラ用テストスイート生成システム testgen2,” 情報処理学会関西支部大会, A-06 (Sept. 2010).
[5] http://ist.ksc.kwansei.ac.jp/∼ishiura/testgen/.