4A-8

関数毎のプラグマ指定による
コンパイラの最適化オプションセット探索

大城 亮 † 石浦 菜岐佐 †
† 関西学院大学 理工学部 情報科学科

1 はじめに

多くのコンパイラでは，実装されている最適化処理をオプションで制御できる．最適化の効果はソースコードの性質に依存するため，与えられたプログラムに対して最適なオプションセット (オプションの組み合わせ) を探索する研究が種々行われている．反復改善手法 [1, 2, 3] や機械学習 [4] により，プログラムの計算時間やメモリ使用量，消費電力等を最小化する試みが種々行われているが，いずれの研究においてもオプションセットはプログラム単位で指定されている．

これに対し本稿では，オプションセットを関数単位で指定する探索手法を提案する．コンパイラに実装されているプラグマ記述を利用することにより，関数単位でのオプションセット指定を実現する．シミュレーテッドアニーリングを利用した反復改善手法プログラムを実装した結果，対象プログラムについてメモリ使用量を平均約 1.72% 削減できた．

2 最適化オプションセットの探索

コンパイラには多くの最適化処理が実装されているが，その効果は対象となるプログラムやアーキテクチャの特徴に依存する．そのためコンパイラによっては個々の最適化処理をオプションで制御できるようにしている．例えば GCC バージョン 4.7 には 181 個の最適化オプションがあり，個々の最適化処理の適用/非適用をコマンドラインオプションで制御できる．

この最適化オプションセットをプログラムやアーキテクチャに合わせて調整することにより，計算時間やメモリ使用量，消費電力等の削減が図れる．オプションが多いと人手による調整は困難であるため，反復改善手法や機械学習を用いたオプションセット探索が研究されている [1, 2, 3, 4].

これらの研究では，プログラム単位でオプションセットを指定することが前提となっている．しかし，プログラム中に性質の異なる関数が混在している場合には，十分な効果が得られない可能性がある．

```
 1  #include <stdio.h>
 2  void func1();
 3  void func2();
 4  int main(void){
 5      func();
 6      return 0;
 7  }
 8  #pragma GCC optimize("align-jumps","gcse")
 9  void func1(){
10      ··· ;
11  }
12  #pragma GCC optimize("no-inline-functions")
13  void func2(){
14      ··· ;
15  }
```

図 1: プラグマによる最適化オプションの指定例

3 関数毎の最適化オプションセット指定

本稿では，プログラム単位ではなく関数単位でオプションセットを指定する手法を提案する．関数毎の最適化オプションセット指定は，GCC 等に実装されているプラグマ記述により行う．

図 1 は GCC におけるプラグマ指定の例である．8 行目と 12 行目のプラグマは，それぞれ直下にある関数 `func1` と `func2` に作用する.「`optimize`」は最適化オプションの指定を意味し，関数 `func1` に `-falign-jumps` と `-fgcse`，関数 `func2` に `-fno-inline-functions` が適用される．コマンドラインの指定とプラグマ指定が競合した時には，プラグマ指定が優先される．図 1 のファイルに対して `-finline-functions` をコマンドラインから指定した場合，関数 `func2` にはプラグマで該当処理の非適用 (`no-inline-functions`) が指定してあるため，インライン展開の処理は行われない．

このように各関数の冒頭にオプションセットを指定するプラグマを挿入することにより，関数単位で最適のオプションセットを設定することができる．

4 メモリ使用量最小化を目的とした探索

本稿では，シミュレーテッドアニーリングを利用した反復改善手法によりメモリ使用量を最小化するオプションセットを求める．オプションの個数を m 個，関数の個数を k 個とした場合，$(s_1, \cdots, s_m, (t_{1,1}, \cdots, t_{1,m}), \cdots, (t_{k,1}, \cdots, t_{k,m}))$ を

Automatic Selection of Compiler Optimization Options for Individual Functions
† Ryo Ohki
† Nagisa Ishiura
School of Schiense and Technolgy. Kwansei Gakuin University (†)
2-1 Gakuen, Sanda, Hyogo 669-1337, Japan

シミュレーテッドアニーリングの一状態とする．ただし，$s_i = 1(0)$ はコマンドラインに指定する i 番目オプションの適用 (非適用) を表し，$t_{j,i}$ は j 番目の関数に対する i 番目のオプションの適用 (非適用) を表す．また $s_i, t_{j,k}$ いずれかの値の $0 \rightarrow 1$，$0 \rightarrow 1$ 変化を状態遷移とする．

本稿では，[5] と同様に探索目的をメモリ使用量の最小化とする．メモリ使用量は ROM と RAM の使用量の合計値とする．ROM の使用量は text セクション，data セクションおよび rodata セクションのサイズの和であり，RAM の使用量は data セクション，bss セクションおよびスタックのサイズの和である．各セクションのサイズはオブジェクトコードのヘッダから取得できる．スタックサイズはアセンブリ命令列から算出する．

5　実装と実験

以上の手法に基づき，与えられたプログラムに対する最適なオプションセットを探索するプログラムを Perl 5.10 で実装した．探索アルゴリズムには適応的温度調節機能を持つ温度並列シミュレーテッドアニーリング [6] を利用した．コンパイラには arm 用 GCC のバージョン 4.7.0 を使用した．このコンパイラの最適化オプションは全部で 181 個あるが，プラグマを使用した際にエラーが発生する 13 個を除外した 168 個で探索を行った．

実験結果を表 1 に示す．列「Prog」は対象プログラム名である．「 sha 」,「 rijndael 」,「 blowfish 」 は MiBench のセキュリティ部門に，vfprintf は「 newlib 」に含まれるプログラムである．列「Func」が対象プログラム中に含まれる関数の個数である．列「OP」のうち「-Os」が GCC の `-Os` オプションの結果であり「従来」が同じ探索手法でプログラム単位に指定する [5] の結果，そして「本手法」が提案手法による数値である．列「Total」がメモリ使用量で，本稿における最小化目標である．列「Ratio」は `-Os` の「Total」を 100 としたときの比であり，列「CPU」は探索に要した CPU 時間である．提案手法では，従来手法と比較して 3 倍から 100 倍の探索時間が必要となるが，対象プログラムに対して平均約 1.72% メモリ使用量を削減できた．

6　むすび

本稿では，関数毎プラグマ指定によるコンパイラの最適化オプション探索手法を提案した．4 つのプログラムについて探索した結果，メモリ使用量を平均約 1.72% 削減できた．

今後の課題としては，探索収束までの時間の短縮や，実行時間の短縮への適用などが挙げられる．

参考文献

[1] Z. Pan and R. Eigenmann: “Fast and effective orchestration of compiler optimizations for automatic performance tuning,” in *Proc. CGO*, pp. 319–332 (Mar. 2006).

[2] R. P. J. Pinkers, P. M. W. Knijnenburg, M. Haneda, and H. A. G. Wijshoff: “Statistical selection of compiler options,” in *Proc. MASCOTS*, pp. 494–501 (Oct. 2004).

[3] M. Haneda, P. M. W. Knijnenburg, and H. A. G. Wijshoff: “Code size reduction by compiler tuning,” in *Proc. SAMOS*, pp. 186–195 (July 2006).

[4] E. Park, S. Kulkarni, and Johne Cavazos: “An evaluation of different modeling techniques for iterative compilation,” in *Proc. CASES*, pp. 65–74 (Oct. 2011).

[5] 大城亮，石浦 菜岐佐: “メモリ使用量最小化を目的としたコンパイラの最適化オプションセットの探索,” 信学ソ大 A-3-14 (Sept. 2011).

[6] 三木光範, 廣安知之, 輪湖純也, 吉田武史: “適応的温度調節機能を持つ温度並列シミュレーテッドアニーリング,” 65 回情報処理学会全国大会, 5Y–6 (Mar. 2003).

表 1: 実験結果

			Size[B]								
Prog	Func	OP	text	data	bss	stack	ROM	RAM	Total	Ratio	CPU[s]
sha	2	-Os	1240	0	0	52	1240	52	1292	100.0	
		従来	1100	0	0	52	1100	52	1152	89.1	3544.9
		本手法	1076	0	0	44	1076	44	1120	86.6	11187.4
rijndael AES	4	-Os	33310	8	12	60	33318	80	33398	100.0	
		従来	32794	8	12	68	32802	88	32890	98.4	19441.9
		本手法	32682	8	12	68	32690	88	32778	98.1	871415.9
blowfish	9	-Os	5192	4172	0	172	9364	4344	13708	100.0	0.3
		従来	9160	4	0	180	9164	184	9348	71.0	20860.2
		本手法	8988	4	0	168	8992	172	9164	66.8	201368.6
vfprintf	2	-Os	6616	0	0	36	6616	36	6652	100.0	-
		従来	5976	0	0	36	5976	36	6012	90.3	1385.7
		本手法	5854	0	0	36	5854	36	5890	88.6	142237.3