4J-4

プロファイル情報を用いた最適化におけるコストと効果についての評価

高橋 恭平　VONGTHONGKHAM Jinlatda　増保 智久　大津 金光　横田 隆史　馬場 敬信†
宇都宮大学工学部情報工学科‡

1　はじめに

従来から，より効果的な最適化を実現する方法として，プログラム実行時の動的情報(プロファイル情報)を用いた最適化手法が研究されてきた．本研究では，再コンパイル時にプロファイル情報を用いて，より効果的な実行コードを得るフィードバック最適化において，最適化後のコードサイズの増分や速度向上率をみることで，各最適化のコスト(適用の際にかかるコード数)と効果を定量的に評価する．評価の対象とするアプリケーションは，メディア処理の重要性を考慮し，MediaBench[1]のJPEGを選択した．本稿では，JPEGのエンコーダプログラムであるcjpegとデコーダプログラムであるdjpegの，実行時間の多くを占めるホットループと実行頻度の高い関数(ホットコール関数)を対象に，(1)ループアンローリング，(2)関数のインライン展開，(3)関数の特殊化の3つの最適化について，前述の評価を行った結果を示す．

2　JPEGの解析

2.1　ホットループ検出

cjpegとdjpegに最適化オプション-O2を適用し，スレッドパイプライニングモデルシミュレータSIMCA[2]用にコンパイルする．次に，SIMCA上でプログラムカウンタを2000サイクル間隔でサンプリングしながら実行する．そして，ループに含まれるアドレスがサンプリングされた回数の多い順に上位5位までをホットループとして検出する．**表1**と**表2**がその結果である．表中のコードサイズは命令数を表し，%はプログラムの全実行時間に対する各ホットループの占める割合を表す．

2.2　ホットコール関数検出

cjpegとdjpegをGNUのプロファイラgprof[3]用にコンパイルし，生成したオブジェクトファイルとプロファイルデータファイルから各関数の呼び出し回数を検出する．呼び出し回数の多い順に上位5位までをホットコール関数とする．**表3**と**表4**がその結果である．

表1. cjpegのホットループ情報

	ループを含む関数名	コードサイズ	(%)
#1Loop	encode_mcu_AC_refine	17	17.968
#2Loop	encode_mcu_AC_refine	65	14.401
#3Loop	encode_mcu_AC_first	49	10.855
#4Loop	forward_DCT	42	8.223
#5Loop	jpeg_fdct_islow	155	6.026

表2. djpegのホットループ情報

	ループを含む関数名	コードサイズ	(%)
#1Loop	jpeg_idct_islow	218	27.128
#2Loop	ycc_rgb_convert	35	20.401
#3Loop	jpeg_idct_islow	228	14.830
#4Loop	h2v2_fancy_upsample	22	9.404
#5Loop	decode_mcu	80	8.175

表3. cjpegのホットコール関数情報

	関数名	コードサイズ	呼び出し回数
#1Callee	emit_bits	72	28745
#2Callee	emit_symbol	25	17946
#3Callee	emit_buffered_bits	24	12692
#4Callee	emit_eobrun	33	10243
#5Callee	encode_mcu_AC_first	129	2804

表4. djpegのホットコール関数情報

	関数名	コードサイズ	呼び出し回数
#1Callee	jpeg_fill_bit_buffer	92	1682
#2Callee	jpeg_idct_islow	483	851
#3Callee	decode_mcu	314	150
#4Callee	h2v2_fancy_upsample	92	150
#5Callee	jzero_far	8	150

3　最適化手法と評価手法

(1) ループアンローリング

表1と**表2**に示したホットループを対象とする．ループ1回分の処理量を増やし繰り返し回数を減らすことで，ループ制御のオーバーヘッドを削減する．ループ制御変数の開始値や終了値が変数で与えられていて，繰り返し回数が静的に不明な場合は，変数の値を出力する命令を加えて実行することでループの繰り返し回数を調べ，これを評価の際のヒントとする．繰り返し回数がアンロール回数で割りきれない場合，端数の処理が必要となるが，これはループで回す．アンロール回数(ループの中に展開する処理数，最適化前は1回とする)は繰り返し回数の半分程度まで増やして評価を行う．

(2) 関数のインライン展開

表3と**表4**に示したホットコール関数を対象とする．実行頻度の高い関数の呼び出しもと(caller)にその関数を展開することで，関数呼び出しのオーバーヘッドを削減する．各ホットコール関数に対し，複数のcallerがある中で，呼び出し回数の多い順上位5位までのcallerを対象とする．それぞれのcallerのみにインライン展開した場合と，呼び出し回数上位5位すべてのcallerにインライン展開した場合の評価を行う．

(3) 関数の特殊化

表3と**表4**に示したホットコール関数を対象とする．関数のインライン展開と同様に，各ホットコール関数のそれぞれのcallerの中で呼び出し回数が多い順上位5位までを対象として関数の特殊化を行う．まず，対象のcallerにおいて，引数の値を出力する命令を加えて実行することで引数値パターンの偏りを調べる．引数が複数ある場合は，セットで1つのパターンとする．1番頻度が高い引数値パターンによって特殊化した関数を用意し，callerの引数値と一致した場合にこれを呼び出し，その他の場合はもとの関数を呼び出すようにする．特殊化した関数内では，引数である変数を定数に置き換え，定数伝播やそれによって無用となる命令を削除することで最適化する．なお，引数がポインタ変数であるものは，最適化前と後でアドレス値が変化する可能性があるため，対象としない．それぞれのcallerでのみ特殊化した場合と，呼び出し回数上位5位すべてのcallerで特殊化した場合の評価を行う．

Evaluation of Cost and Effect of Optimization Using Profile Information

† Kyohei Takahashi, Jinlatda Vongthongkham, Tomohisa Masuho, Kanemitsu Ootsu, Takashi Yokota and Takanobu Baba
‡ Department of Information Science, Faculty of Engineering, Utsunomiya University

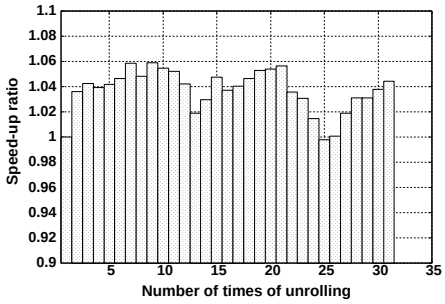

図 1. cjpegの＃2Loopを対象としたループアンローリングの速度向上率

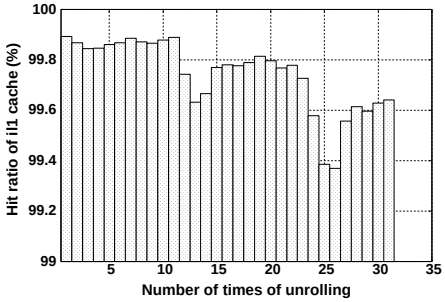

図 2. cjpegの＃2Loopを対象としたループアンローリングの命令キャッシュヒット率

4　評価

評価にはSIMCAを用いた．命令キャッシュ，データキャッシュ，2次キャッシュの容量(KB)はそれぞれ16，16，256，連想度は1，4，4，レイテンシ(cycle)は1，1，6，置換法はすべてLRUである．コンパイラ最適化オプションは-O2を適用する．各最適化の効果の指標となる速度向上率は，最適化前の全実行サイクル数を最適化適用後の全実行サイクル数で割ったものである．

(1) ループアンローリング

図1はcjpegの#2Loopを対象とした場合の速度向上率であり，**図2**はそのときの命令キャッシュのヒット率である．どちらも横軸がアンロール回数である．アンロール9回のとき1.059倍で1番の速度向上を得た．これは各ホットループの中で最高である．アンロール13回と25回のとき速度向上率の低下が大きい．このときの命令キャッシュのヒット率をみると，ともに低下しており，このことが大きな原因と考えられる．#2Loopの繰り返し回数は63回であり，アンロール13回，25回付近では端数処理が10回以上と多く，逆に端数処理が0となる7，9，21，31回のときは比較的高い速度向上がみられる．ループアンローリングではほとんどの場合に速度向上率と命令キャッシュのヒット率に相関がみられ，これが速度向上に大きく影響していると考えられる．cjpegで速度向上がみられたのは#1〜#3Loopである．これらのループのコードサイズはそれぞれ17，65，49であり，総イテレーション数はいずれも10万を超える．速度向上しなかった#4，#5Loopのコードサイズは42，155であり，総イテレーション数は851，6808である．ループのコードサイズが大きいとキャッシュヒット率低下の危険が増し，総イテレーション数が少ないとオーバーヘッド削減による高速化要素が少なく速度向上が期待できないと考えられる．djpegの#1，#3Loopはソースにすでにループアンローリングが適用されており，これ以上適用するとコードサイズが増し，キャッシュヒット率が低下してしまい速度向上は得られなかった．

(2) 関数のインライン展開

cjpegで1番の速度向上を得られたのは，#3Calleeを呼び出し回数上位3位すべてのcallerに展開したときの1.0098倍である．#3Calleeのコードサイズは24，callerは全部で3つあり，そのすべてにインライン展開しても，命令キャッシュのヒット率は低下しなかった．cjpegの#1Calleeはコードサイズが72，callerは全部で8つある．この場合，1番の速度向上は呼び出し回数1位のcallerのみに展開したときの1.0073倍であった．呼び出し回数上位5位すべてに展開した場合の速度向上率は1.0067であり，速度向上するが，呼び出し回数1位のcallerのみに展開したときよりも低い値である．これは命令キャッシュのヒット率低下の影響であると考えられる．このように，関数サイズが小さい場合は，キャッシュのヒット率が低下しない範囲で複数のcallerに展開することで速度向上を得られる．また，呼び出し回数が少ないと，関数呼び出しのオーバーヘッド削減による高速化要素が少なく，キャッシュミスによる低速化の影響が大きくなることが確認できた．一方，djpegはマクロ定義の中にcallerがある場合や，関数ポインタ変数が頻繁に使用されるなど，最適化適用が困難であったり，速度向上が得られない場合がほとんどであった．

(3) 関数の特殊化

cjpegの#4，#5CalleeとdjpegのCalleeは，引数がすべてポインタ変数であるため適用不可能であった．1番の速度向上を得られたのは，cjpegの#3Calleeのすべてのcallerを対象に特殊化したときの1.0085倍である．この関数は頻度1位の引数値では何も処理をしない関数であり，この引数値では関数を呼び出さないよう最適化したため，特殊化によって削除できる命令数は最大であるといえる．また，引数値パターンの偏りもすべてのcallerで60％を超える．一方，cjpegの#3Callee以外では速度向上を得られなかった．これは，特殊化によって削除できる命令数が少なく，引数値の判定のための命令の増加による影響が大きいためと考えられる．このように，特殊化によって削除できる命令数が多く，引数値パターンの偏りが十分に大きい場合でないと速度向上が期待できないことが確認できた．

5　おわりに

本稿では，JPEGのcjpegとdjpegを対象にプロファイル情報を用いた各最適化のコストと効果についての定量的な評価を行った．今後の課題として，評価内容を深めるためにより多くのアプリケーションを対象にすることや，各最適化を組み合わせて適用した場合にどれだけの速度向上を得られるか評価することなどが挙げられる．

謝辞　本研究は，一部日本学術振興会科学研究費補助金(基盤研究(B)14380135，同(C)16500023，若手研究(B)17700047)の援助による．

参考文献

[1] Chunho Lee, et al., “MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems,” http://cares.icsl.ucla.edu/MediaBench/.

[2] J. Huang, “The SImulator for Multithreaded Computer Architecture (Release 1.2),” http://www.cs.umn.edu/Research/Agassiz/Tools/SIMCA/simca.html.

[3] Jay Fenlason and Richard Stallman, “GNU gprof,” http://www.gnu.org/software/binutils/manual/gprof-2.9.1/html_mono/gprof.html