2L-1

MPEG2 エンコーダにおけるコード最適化の適用とその効果

矢野目 秀人 † 佐藤 和史 † 高橋 恭平 † 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グラム実行時の動的情報を用いて，より効果的な実行コードを得る動的最適化において，最適化後の速度向上率をみることで，各最適化の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 MPEG2 を選択した．本稿では，MPEG2 のエンコーダプログラムである mpeg2encode 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 つの最適化および (4) それらの組み合わせについて，前述の評価を行った結果を示す．

2 mpeg2encode の解析

2.1 ホットループ検出

mpeg2encode に最適化オプション-O2 を適用し，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 にホットループ情報を示す．表中の%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

mpeg2encode を GNU のプロファイラ gprof[2] 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．表 2 にホットコール関数情報を示す．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．

2.3 本稿で対象とする関数

プロファイリングの結果より関数 dist1 への偏りが見られるため，本稿では関数 dist1 を対象として評価を行う．図 1 に関数 dist1 の概略を示す．関数 dist1 の実行頻度の高い関数の呼び出し元 (caller) は 1 つしかなく，caller 内の 3 カ所で呼び出されている．

3 最適化手法と評価手法

(1) ループアンローリング

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，端数の処理はループにより行う．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) は可能な限り繰り返し回数の最後まで増やして評価を行う．

表1. mpeg2encodeのホットループ情報

	ループを含む関数名	命令数	(%)
#1Loop	dist1	105	61.789
#2Loop	dist1	20	6.766
#3Loop	fullsearch	61	4.247
#4Loop	fdct	13	4.156
#5Loop	fdct	11	3.015

表2. mpeg2encodeのホットコール関数情報

	関数名	命令数	呼び出し回数
#1Callee	dist1	193	1061538
#2Callee	putbits	110	184793
#3Callee	putAC	54	77659
#4Callee	idctcol	192	67584
#5Callee	idctrow	152	67584

```
dist1(引数) {
  :
 変数宣言
  :
 if (!hx && !hy)       { ホットループ }  // 第１ if 文
 else if (hx && !hy)   { ２重ループ }    // 第２ if 文
 else if (!hx && hy)   { ２重ループ }    // 第３ if 文
 else                  { ２重ループ }    // 第４ if 文
 return
}
```

図 1: 関数 dist1 の概略

(2) 関数のインライン展開

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．

(3) 関数の特殊化

まず，対象関数の 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．

4 評価

評価には SimpleScalar[3] 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．

Application of Code Optimizations to MPEG2 Encoder and Their Effect

†Hideto Yanome , Kazufumi Sato , Kyohei Takahashi , Tomohisa Masuho , Kanemitsu Ootsu , Takashi Yokota and Takanobu Baba
Department of Information Science, Faculty of Engineering, Utsunomiya University (†)

(1) ループアンローリング

#1Loop では，アンロール 8 回の時に 1.104 倍で 1 番の速度向上を得た．次に速度向上率がよかったのは，第 4 if 文の内側ループのアンロール 16 の時の速度向上率である．以降は第 2 if 文の内側ループ，第 3 if 文の内側ループの順に速度向上率がよかった．速度向上率はアンロール回数に比例してよくなっていた．速度向上率のよかった順は，そのままホットループの順位にとなっている．第 2~ 4 if 文の内側ループは，第 4，2，3 if 文の順に#2，#6，#7Loop である．それに対し，第 2~ 4 if 文の外側ループはアンロール回数が増えるほど速度低下を引き起こした．これらは，アンロール回数の増加による命令キャッシュミス率の増加分が内側ループに比べ大きくなるため，またサンプリングされたループの中で順位が下位であり，それほどホットではないためと考えられる．ループアンローリングは速度向上率を大きく向上させることができるが，命令キャッシュミスも大きく上昇させる．命令キャッシュミスにより，速度向上せず速度低下してしまうこともある．

(2) 関数のインライン展開

mpeg2encode で 1 番速度向上率が得られたのは，関数 dist1 を 3 カ所ともインライン展開したときの 1.017 倍である．caller 内における 2 つ目の関数 dist1 の呼び出し回数は関数 dist1 全体の 92%を占め，hx および hy が固定で第 1 if 文しか実行されず展開される命令数も少なくてすむことから，この部分だけをインライン展開しても 1.015 倍の速度向上が得られる．インライン展開を適用しても，命令キャッシュミス率が大きく増大することはないが，ほとんど速度向上率の向上を得られなかった．

(3) 関数の特殊化

ループ制御変数の最大値を決める変数で，特殊化を行った．これにより，関数を最大イテレーション数 8 と 16 の場合に分けることができる．結果，1.004 倍の速度向上率が得られた．このとき 3 つ目の関数 dist1 のみを特殊化しないようにするとさらに速度向上する．これは, 関数 dist1 の 3 つ目の呼び出し回数が関数 dist1 全体の呼び出し回数の 7 %にすぎないため，特殊化による判定のコストの方が大きかったためだと考えられる．インライン展開の時と同様に，関数の特殊化を適用しても命令キャッシュミスが大きく増大することがない代わりに，ほとんど速度向上率の向上は得られなかった．

(4) 各種最適化の組み合わせ

(1)~ (3) の最適化の評価をふまえ，それらの組み合わせによるさらなる速度向上の評価を行った．組み合わせの対象としたのは，(1)~ (3) で速度向上を得られたものに限る．得られた速度向上率のグラフを図 2 に示す．横軸は，最適化無し（A），関数 dist1 を 2 つともインライン展開（B），B で 2 つ目の関数 dist1 のループアンローリング（C），C において第 2 から第 4 if 文の内側ループをそれぞれ全展開（D~ F），C において第 2，3 if 文の内側ループを全展開（G），C において第 2，4 if 文の内側ループを全展開（H），C において第 3，4 if 文の内側ループを全展開（I），C において第 2~ 4 if 文の内側ループを全展開（J），J において caller の前 2 つの関数 dist1 に対し関数の特殊化を適用しさらにアンロール可能となった部分を全展開（K）となっている．これを見ると，速度向上率はループアンローリングの結果のよかったものの影響が強く出ている．たとえば，I よりも H の方がよい結果になっているのは，第 3 if 文の内側ループの方より第 2 if 文の内側ループの方が速度向上率が良いためである．インライン展開や関数の特殊化は，ループアンローリングに比べると速度向上率がよいとはいえない．しかし，関数の特殊化無しでは J が限界であるが，関数の特殊化によりイテレーション数 16 を全展開することが可能となり，さらなる速度向上率が得られた．インライン展開や関数の特殊化はそれ単体では高い速度向上を期待できないが，それらによってループアンローリングなどの最適化の機会を増やすことができる．

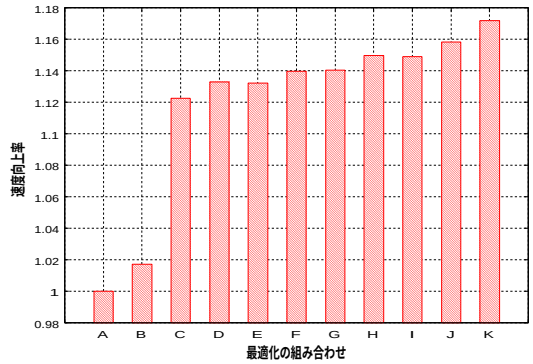

図 2: 最適化の組み合わせによる速度向上率

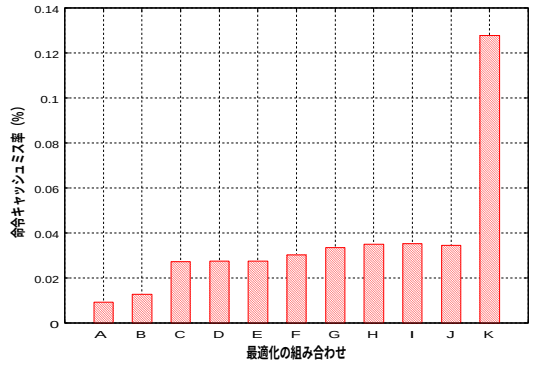

図 3: 最適化の組み合わせによる命令キャッシュミス率

5 おわりに

本稿では，MPEG2 の mpeg2encode を対象にプロファイル情報を用いた各最適化の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，関数 dist1 以外に対しても評価を取り，全体的な最適化を得る．また，他のベンチマークでの評価などが上げられる．

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)18300014, 同 (C)16500023, 若手研究 (B)17700047）および宇都宮大学重点推進研究プロジェクトの援助による.

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

[3] SimpleScalar LLC http://www.simplescalar.com/.