2L-2

MPEG2デコーダにおけるコード最適化の適用とその効果

佐藤 和史 † 矢野目 秀人 † 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

年々高まる計算機システムの高性能化の要求に対し，プログラムの解析による最適化が研究されている．しかし，静的な情報を元にした最適化による性能向上には限界がある．本研究ではプログラム実行時の挙動を動的な情報として得て，プログラム中の実行頻度の高いコード領域に対して各種最適化を施すことにより，そ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 のデコーダプログラムである mpeg2decode 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である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 mpeg2decode の解析

2.1 ホットループ検出

mpeg2decode 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のサンプリング回数が多い順の上位 5 位までをホットループとして検出する．表 1 がホットループの検出結果である．ホットループの上位 5 位までを#1〜#5Loop と表す．%はプログラム全体でのサンプリング回数に対する各ホットループの占める割合を示す．#5Loop の割合が少ないために最適化の効果が期待できないと考え，mpeg2decode ではホットループの上位 4 位までを最適化の対象とする．ここで，ホットループの説明をすると#4Loop のループ内に#1Loop が，#3Loop のループ内に#2Loop が含まれている．

2.2 ホットコール関数検出

mpeg2decode 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求める．呼び出し回数が多い順の上位 5 位までをホットコール関数とする．表 2 がホットコール関数の検出結果である．ホットコール関数の上位 5 位までを#1〜#5Callee と表す．#5Callee の呼び出し回数が他のホットコール関数と比べ，かなり少ないために最適化の効果が期待できないと考え，mpeg2decode ではホットホットコール関数の上位 4 位までを最適化の対象とする．

3 ループアンローリング

3.1 ループアンローリングの適用方法

表 1 に示したホットループを対象とする．この最適化手法はループ 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ことができる．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) は繰り返し回数の半分程度まで増やして評価を行う．ただし，ループ内の要素を全て展開する場合はループ制御を削除する．mpeg2decode の各最適化対象ホットループの繰り返し回数は全て 8 回である．また，#3Loop と#4Loop を展開する場合は，内側のループを全て展開した状態を要素として展開していく．今回，端数処理が必要となるのはアンロール 3，5，6，7 回の場合である．そのうち，アンロール 5，6，7 回では展開した部分が 1 回しか実行されずループになっていないので評価対象としない．

表 1: mpeg2decode のホットループ

	ループを含む関数名	命令数	(%)
#1Loop	Reference_IDCT	14	31.962
#2Loop	Reference_IDCT	12	22.732
#3Loop	Reference_IDCT	26	10.002
#4Loop	Reference_IDCT	39	7.949
#5Loop	Add_Block	12	1.711

表 2: mpeg2decode のホットコール関数

	関数名	命令数	呼び出し回数
#1Callee	putbyte	20	506880
#2Callee	Show_Bits	5	103514
#3Callee	Flush_Buffer	85	102121
#4Callee	Get_Bits	12	49847
#5Callee	form_component_prediction	129	8238

3.2 ループアンローリングの評価結果

評価には SimpleScalar 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．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．この評価環境は他の最適化の場合でも同じである．

図 1 は mpeg2decode の#4Loop 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ミス率である．どちらも横軸がアンロール回数である．ループアンローリングで一番の速度向上を得たのは#4Loop のアンロール 8 回のときの 1.3606 倍である．#4Loop の速度向上率が他より良かったのは，#1Loop が内側のループであるため，#1Loop の最適化による速度向上の影響も加えられたと考えられる．#3Loop 以外では端数処理が無く，アンロール回数が多いほど速度向上した．また，アンロール回数が増えると命令キャッシュミス率も上昇した．#3Loop の場合はアン

Application of Code Optimizations to MPEG2 Decoder and Their Effect
†Kazufumi Sato, Hideto Yanome, Kyohei Takahashi, Tomohisa Masuho, Kanemitsu Ootsu, Takashi Yokota and Takanobu Baba
Department of Information Science, Faculty of Engineering, Utsunomiya University (†)

ロール 1 回より速度向上は良くならなかった．#3Loop にループアンローリングを適用したときの関数内の命令の実行回数を調べてみるとアンロール 1 回のときが一番少なかった．しかし，#1，#2，#4Loop ではアンロール回数を増やすと命令の実行回数は減少していた．このことから，最適化の影響による命令の実行回数の増加が，速度向上しなかった原因と考えられる．

4 関数のインライン展開

4.1 関数のインライン展開の適用方法

表 2 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．

4.2 関数のインライン展開の評価結果

一番の速度向上を得られたのは#1Callee の第 1 位の caller に展開したときの 1.0183 倍である．これは，#1Callee の第 1 位の caller の呼び出し回数が他の caller よりも多いために，関数呼び出しのオーバーヘッド削減による高速化の影響が大きくなったためである．インライン展開のほとんどは呼び出し回数が多いほど速度向上していたが，#3Callee の第 1 位の caller に展開したときは速度低下した．この原因を調べるために最適化オプションを変更して評価を行ったところ，最適化オプションの fstrength-reduce をはずした場合は，#3Callee の第 1 位の caller にインライン展開を適用しても速度低下しないことがわかった．

5 関数の特殊化

5.1 関数の特殊化の適用方法

表 2 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．mpeg2decode では#1Callee の引数がポインタ変数であるため対象から外す．また，引数が定数であった場合は引数値の判定は行わず，特殊化した関数を呼び出すようにする．

5.2 関数の特殊化の評価結果

1 番の速度向上を得られたのは，#2Callee の第 5 位の caller を対象に特殊化したときの 1.0007 倍である．全体として#2Callee の第 2〜5 位の caller に適用した場合以外は，速度向上しなかった．その原因としては，#2〜#4Callee の関数の命令数と特殊化した関数の命令数を比べると，#2Callee では特殊化することで命令数が 1 つ減っていたが，他の Callee では変わらなかった．そのため，特殊化のメリットである無用命令の削除による速度向上が表れなかったと考えられる．また，

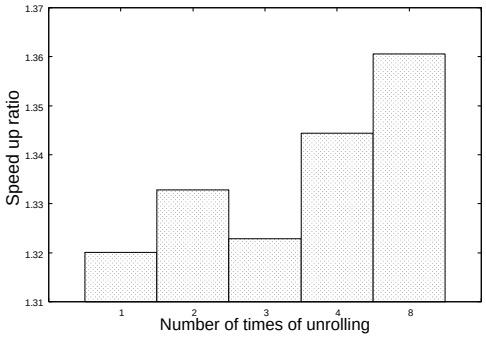

図 1: # 4Loop を対象としたループアンローリングの速度向上率

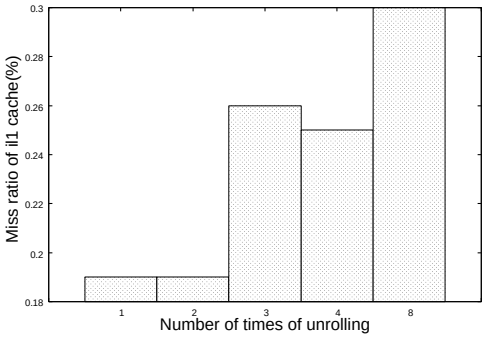

図 2: # 4Loop を対象としたループアンローリングの命令キャッシュミス率

#2Callee の第 2〜5 位の caller の引数は定数であったが，#2Callee の第 1 位の caller は変数であるために，引数値の判定による命令数の増加が速度低下の原因と考えられる．インライン展開と関数の特殊化を比べると，インライン展開の方が効果が大きい．これは，多くの caller の引数が定数であるために，関数の特殊化を行うよりはインライン展開を行う方が無用命令の削除に加え，関数呼び出しのオーバーヘッドを削減できるためと考えられる．

6 おわりに

本稿では MPEG2 デコーダを対象にコード最適化の効果について定量的な評価を行った．今後の課題としては，各最適化を組み合わせて適用した場合にどれだけ速度向上をするか評価することがあげられる．また，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も重要となる．

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] 中田育男，“コンパイラの構成と最適化，” 朝倉書店, pp.229-482， 1999.09