B-002

ループ展開による Java バイトコード最適化システムの試作

Prototyping of Byte Code Optimization System by Loop Unrolling

寒川 明好† 岩澤 京子†
Akiyoshi Samukawa Kyoko Iwasawa

1. はじめに

最小の努力で最大の効果を得るには，プログラムの中でもっとも頻繁に実行されている部分を見つけ出し，その部分をできるだけ効率の良いコードにすればよい[1]．実際に改良の候補になるのは，プログラムのループ，それも最内側のループである．

そこで，実行時間と動的バイトコード数の削減を目的として，Java バイトコード[2]を解析しループ展開による最適化をおこない，実行可能なクラスファイルを出力するシステムを開発した．

本論ではこのシステムを紹介し，ループ展開による最適化の効果を，単純な科学計算ベンチマークプログラムで展開前と展開後の実行時間を比較する実験から確認した．

2. 設計方針

本システムでは，入出力をバイトコードである Java クラスファイルとした．

入力されたバイトコードの解析をおこない，インストラクションコードから制御フロー解析をおこなう．作成したフローグラフから最内側ループの検出し，ループ展開処理を施した新しいフローグラフの作成をおこなう．最後に実行可能な Java クラスファイルを作成して出力する．図-1 に本システムの全体構成図を示す．

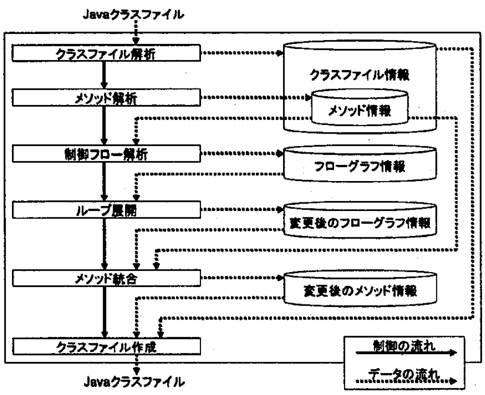

図-1 本システムの全体構造

ループを展開すると，ループ内実行文は展開数だけ増える．最も効率的な展開数はループ内のオブジェクト数により変動する．展開数を決める際に，オブジェクト数が少ない場合と多い場合の 2 つのパターンで効率な展開数を測定した結果，本システムでは展開数を 8 とした．

† 拓殖大学, Takushoku University

3. ループ展開方法

本システムでの展開方法は，次の手順でおこなう．

① 8 倍展開するループと，余りをおこなうループの 2 つに分ける．
② 8 倍展開するループでは，ループ内実行文とインダクション変数定義文を展開数だけ複写する．
③ ループ繰返し回数を展開数だけ削減する．
④ 余りをおこなうループでは，展開前のループの形を崩さずにインダクション変数定義文を受け継ぐ．

このループ展開のソース例を次の図-2 に示す．

```
for ( l=1 ; l<=loop ; l++ ) {
    for ( k=0 ; k<n ; k++ ) {
        x[(int)k] = q + y[(int)k]*( r*z[(int)k+10] + t*z[(int)k+11] );
    }
}
                        ⇩
for ( l=1 ; l<=loop ; l++ ) {
    for ( k=0 ; k<n-8 ;) {
      x[(int)k] = q + y[(int)k]*( r*z[(int)k+10] + t*z[(int)k+11] );  k++;
      x[(int)k] = q + y[(int)k]*( r*z[(int)k+10] + t*z[(int)k+11] );  k++;
      x[(int)k] = q + y[(int)k]*( r*z[(int)k+10] + t*z[(int)k+11] );  k++;
      x[(int)k] = q + y[(int)k]*( r*z[(int)k+10] + t*z[(int)k+11] );  k++;
      x[(int)k] = q + y[(int)k]*( r*z[(int)k+10] + t*z[(int)k+11] );  k++;
      x[(int)k] = q + y[(int)k]*( r*z[(int)k+10] + t*z[(int)k+11] );  k++;
      x[(int)k] = q + y[(int)k]*( r*z[(int)k+10] + t*z[(int)k+11] );  k++;
      x[(int)k] = q + y[(int)k]*( r*z[(int)k+10] + t*z[(int)k+11] );  k++;
    }
    for ( ; k<n ; k++ ) {
      x[(int)k] = q + y[(int)k]*( r*z[(int)k+10] + t*z[(int)k+11] );
    }
}
```

図-2 8 倍展開によるプログラムの変更例

ループ展開により削減される動的バイトコード数は，展開対象のループ数，展開数，ループ内オブジェクト数によって異なる．

ループ繰返し回数が定数で，その値が 8 以下のときに，本システムではループの解消をおこなう．ループ繰返し回数が定数 9 以上，または変数の場合には 8 倍展開をおこなう．表-1 に 8 倍展開によって削減される動的バイトコード数を示す．

表-1 8 倍展開により削減される動的バイトコード数

N:ループ回数
m:ループ内動的命令数

ループ回数Nの条件		8倍展開で削減される動的バイトコード数
ループ回数Nが定数	N<=8	4*N
	N>=9	26*(N/8)
ループ回数Nが変数	N>=8 && N%X≠0	
	N>=8 && N%X=0	-3+(5+m)*N-(14+8*m)*(N/8)
	N<8	-5

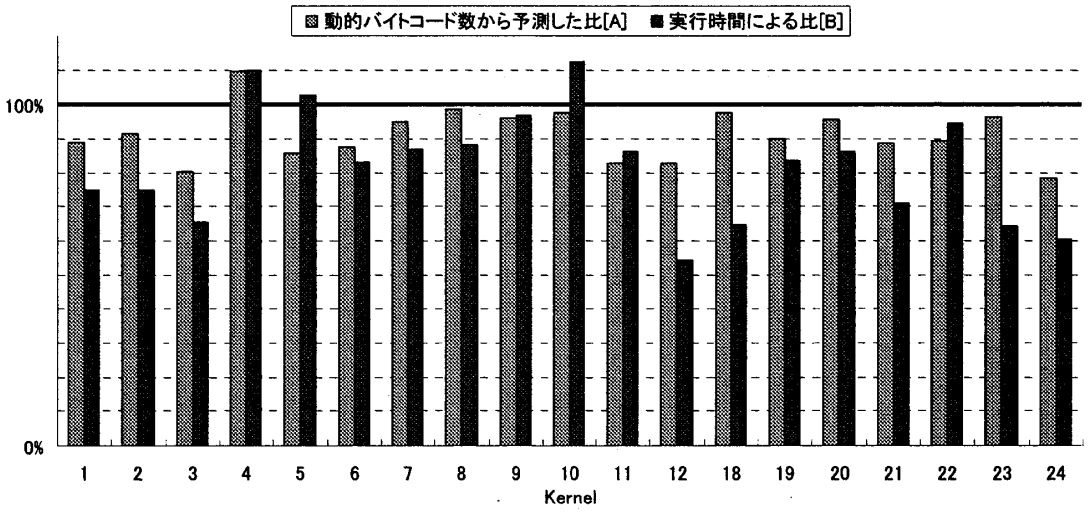

図-3 カーネルごとのループ展開前後の動的バイトコード数の比と実行時間の比

4. 性能評価実験

ループ展開によって，削減する動的バイトコード数以外にも基本ブロックが大きくなるため，他の最適化の効果が期待できる．そこで次の実験をおこなった．

4.1 テストプログラム

物理シミュレーションなどの数値計算でおこなわれる基本的な演算のセットであり，過去にベクトル型計算機のベンチマークとして用いられた，L.L.N.L（Lawrence Livermore National Laboratory）"C" Kernels[3]は，ループが小さく解析しやすいため，本実験で人手により Java に書き換えて用いた．このベンチマークプログラムには 24 個のカーネルがあり，goto ラベル文や計算型 IF 文を除いた 19 個のカーネルを対象におこなった．

4.2 実験方法

上記テストプログラムを元に，次の方法でこの実験をおこなった．

(1) カーネルごとに，ループ展開で削減される動的バイトコード数の比を求め，これを性能向上の予想値とした．[図-3 の A]

展開後動的バイトコード数／展開前動的バイトコード数

(2) 展開前と展開後の実行時間の実測により比を求め，(1)の予想値と比較した．[図-3 の B]

展開後の実行時間／展開前の実行時間

オリジナルループを 100%として，上記測定を次の環境でおこなった結果を図-3 に示す．

- OS：Microsoft Windows XP Professional
- CPU：Intel® Core™ Duo CPU E6850 3.00GHz (2 CPUs)
- Memory：2048MB RAM
- Java の実行環境：jdk1.6.0_06，jre1.6.0_06[4]

4.4 実験のまとめ

バイトコードの動的命令数の比較から，図-3 の結果を次の 3 つに分けた．

① 予想値より速いループ

カーネル 1,2,3,6,7,8,12,18,19,20,21,23,24

② 予想値と同じループ

カーネル 4,9

カーネル 4 で，予想値が展開前よりも増えているのは，ループ繰返し回数が展開数よりも小さいためである．

③ 予想値より遅いループ

カーネル 5,10,11,22

カーネル 5,10 は，展開前よりも実行時間が長くなった．しかしカーネル 11,22 では，予想値よりも削減しなかったが，展開前に比べると削減した．

5. おわりに

実行時間と動的バイトコード数の短縮を目的として，ループ展開をバイトコードでおこなうシステムの開発をした．システムの評価として，オリジナルループとシステムを用いて 8 倍展開したループでの動的バイトコード数の比(予想値)と，実際に計測した実行時間の比をまとめた．バイトコードに対してループ展開をおこなうと，約 2/3 は JIT コンパイラの最適化を促し，予想した実行時間よりも短縮することを確認した．しかし遅くなるループもあり，Java の環境ではループ展開が万能というわけではないことがわかった．

今後は，予想値より遅くなるケースを検討し，そして他の最適化法と組み合わせて，より実行時間の短縮を実現するシステムを研究していく．

参考文献

[1] A.V.エイホ，R.セシィ，J.D.ウルマン『コンパイラ II-原理・技法・ツール-』サイエンス社 1990 年

[2] ティム・リンドホルム，フランク・イェリン『Java 仮想マシン仕様 第 2 版』ピアソン・エデュケーション 2001 年

[3] http://www.netlib.org/benchmark/livermorec (2008/6/17 アクセス)

[4] 「Java SE Downloads」 <http://java.sun.com/javase/downloads/index.jsp> (2008/6/17 アクセス)