バイナリレベルマルチスレッド化における スレッドコード最適化

阿久津 徳寿 佐藤 智一 月川 淳 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

我々はシングルスレッドコードのマルチスレッド化および最適化をバイナリレベルで行なうシステム[1]の構築を進めている。

このシステムを実現するためには、バイナリレベルでのマルチスレッド化手法、およびスレッドコード自体の最適化手法の検討が必要である。本稿ではスレッドコードの最適化手法に焦点を当て、バイナリ変換によるマルチスレッド化の際に適用可能かつ有効な最適化手法の検討を行う。

2 マルチスレッド実行モデル

本研究での、マルチスレッド実行モデルは、図1に示すスレッドパイプライニングモデルをベースとしている。スレッドパイプライニングモデルでは、1スレッドを次のスレッドを起動するために必要なループ変数を計算するContinuation Stage、スレッド間依存データのアドレスを通知するTSAG(Target Store Address Generation) Stage、スレッドに割り当てられた処理を実行するComputation Stage、計算結果をメモリへ書き戻すWrite-Back Stageの4つのステージに分けて実行される。

3 最適化手法

本研究では、バイナリレベルポインタ解析[2]によって得られる最適化に必要なループイテレーション間依存レジスタなどの情報を元に、以下に示す手順でスレッドコードの最適化を行った。

(1) ループアンローリング

マルチスレッド化の対象となるループにループアンローリングを適用する。1スレッドあたりの処理量を増やすことによってスレッド制御のオーバーヘッドを削減し、後述する共通部分式の削除と無用命令の削除を適用する機会を増やす。

(2) 共通部分式の削除

ループアンローリングによってスレッドコード内に生じた共通部分式の削除を行う。

(3) 演算の置き換え

いくつかの定数とループ変数に対して演算を行い得られる結果は、イテレーションごとに定数変化する。これを利用して、初回の演算結果をレジスタへ保存し、以降この値に対して定数を加算する演算へと処理を置き換えることによって、実行される命令数を削減する。

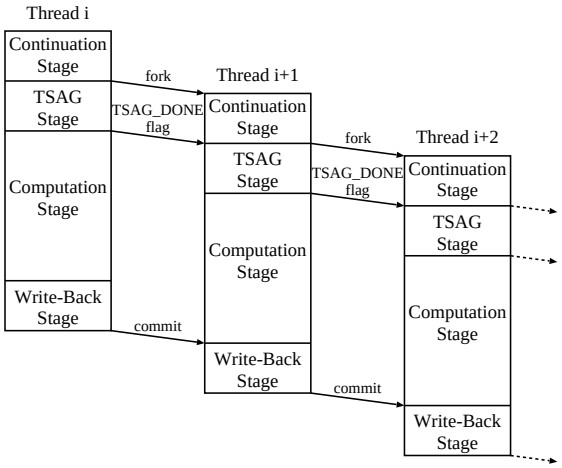

図 1. スレッドパイプライニングモデル

(4) 命令のループ外への移動

ループ外で計算可能な処理はループ外へ移動し、命令の実行頻度を下げる。

(5) 無用命令の削除

最適化によってスレッドコード内で参照されなくなったループ変数の加算や、冗長なループ終了判定などの命令を削除する。

これらの最適化の適用例を図2に示す。図2(a)は最適化される前のループ1イテレーション分のコード、図2(b)はアンローリング回数を2回とした最適化後のコードである。

図2(a)に示す(*)部分のコードは、メモリからロードした値とレジスタに格納されている値に対して演算を行い、最終的に$3へ結果を保存する処理を行っている。この処理過程で使われる値のうち、ループイテレーションごとに変化するのは$9に格納されている値のみであることがバイナリレベルポインタ解析[2]によってわかるので、これらのコードは演算の置き換えにより、以降図2 (b)の(4)に示すコードへと置換できる。同様に図2(a)の(#) 部分のコードも図2(b)の(5)へと置き換えられる。

図2(a)の(1)、(2)はループ変数の加算であるが、前述した演算の置き換えによって、以後スレッドコード内でループ変数は参照されなくなるため、無用命令となり

Thread Code Optimization for Binary-Level Multithreading
† Noritoshi Akutsu, Tomokazu Satou, Atsushi Tsukikawa, Kanemitsu Ootsu, Takashi Yokota, Takanobu Baba
‡ Department of Information Science, Faculty of Engineering, Utsunomiya University

```
(*) lw     $19,116($sp)
(*) addu   $3,$9,$19
(*) sll    $3,$3,0x3
(*) addu   $3,$3,$6
(#) lw     $19,132($sp)
(#) addu   $2,$8,$19
(#) sll    $2,$2,0x3
(#) addu   $2,$2,$4
    l.d    $f2,-8($3)
    l.d    $f0,0($2)
    add.d  $f2,$f2,$f0
(1) addiu  $9,$9,2
(2) addiu  $8,$8,1
(3) slt    $2,$11,$8
    s.d    $f2,-8($3)
```

(a) 最適化前

```
    lw     $19,116($sp)
    addu   $3,$9,$19
    sll    $3,$3,0x3
    addu   $3,$3,$6
    lw     $19,132($sp)
    addu   $2,$8,$19
    sll    $2,$2,0x3
    addu   $2,$2,$4
    l.d    $f2,-8($3)
    l.d    $f0,0($2)
    add.d  $f2,$f2,$f0
    s.d    $f2,-8($3)

(4) addiu  $3,$3,16
(5) addiu  $2,$2,8
    l.d    $f2,-8($3)
    l.d    $f0,0($2)
    add.d  $f2,$f2,$f0
    s.d    $f2,-8($3)
```

(b) 最適化後

図 2. 演算の置き換え、無用命令の削除による最適化例

削除される。(3) はループ変数をもとにループの終了判定を行うコードであるが、ループ変数は Continuation Stage で計算されるため、Computation Stage のループ終了判定は削除される。

4 評価

前述した最適化手法の有効性を検証するため、バイナリレベルのマルチスレッド化において最適化を適用していないコード、ループアンローリングのみを適用したコード、および第 2 節で述べた全ての最適化を適用したコードの実行速度を比較した。

評価にはスレッドパイプライニングモデルシミュレータ SIMCA[3] を用いた。対象アプリケーションとして、SPECfp95 の 107.mgrid を用い、入力データセットは test を使用した。SPECfp95 は FORTRAN で記述されているため、f2c を用いて一度 FORTRAN コードから C コードへ変換した後、SIMCA 用 gcc クロスコンパイラに最適化オプション-O2 を適用し、対象コードを生成した。マルチスレッド化の対象としたのは 107.mgrid

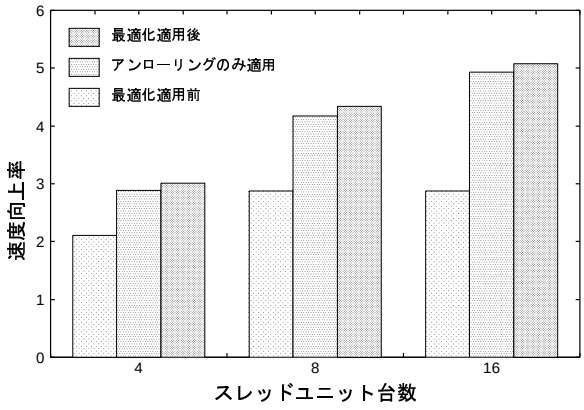

図 3. 107.mgrid における速度向上率

の中で実行頻度の高い上位 4 つのサブルーチン resid_()、psinv_()、interp_()、rprj3_() である。これらのサブルーチンのプログラム全実行に占める実行頻度は 96.27% である。サブルーチンに含まれるループのうち、マルチスレッド化しにくい入出力を含むループを除き、全てのループをマルチスレッド化した。最適化を適用する際のループアンローリング回数は resid_()、psinv_()、rprj3_() で 4 回、interp_() は 1 イテレーション当たりの処理量が少ないため 8 回とした。

スレッドユニット台数を 4 台、8 台、16 台とし、各台数に対して速度向上率＝(シングルスレッド実行サイクル数)/(マルチスレッド実行サイクル数) を測定した結果を図 3に示す。縦軸は速度向上率を、横軸はスレッドユニット台数を表し、各スレッド台数の毎に左から最適化を適用しない場合、ループアンローリングのみを適用した場合、全ての最適化を適用した場合の速度向上率を表す。

最適化後の速度向上率はスレッドユニット 4 台で 3.02 倍、8 台で 4.34 倍、16 台で 5.07 倍となり、全ての台数において最適化前のコードと比較し 1.4 倍以上の速度向上を得た。

5 おわりに

本稿では我々が研究開発しているバイナリレベルマルチスレッド化システムにおいて、マルチスレッド化の際に適用すべき最適化手法の検討を行い、最適化によってアプリケーションの実行速度が向上することを示した。

今後の課題としては、本稿で示した最適化手法の有効性を検証するため、評価対象とするアプリケーションを増やすことが挙げられる。また、速度向上に大きく貢献するループアンローリングについては、1 スレッドコード内の命令数などの情報から、最適なアンローリング回数を決定する必要がある。

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)14580362、若手研究 14780186) の援助による。

参考文献

[1] 大津 金光, 小野 喬史, 横田 隆史, 馬場 敬信, “バイナリレベルマルチスレッド化コード生成手法とその評価,” 情報処理学会論文誌ハイパフォーマンスコンピューティングシステム, Vol44, No.SIG-1(HPS6), pp.70-80, 2003.

[2] 佐藤 智一, 三木 大輔, 横田 昌之, 月川 淳, 大津 金光, 横田 隆史, 馬場 敬信, “バイナリレベルポインタ解析を用いた自動マルチスレッド化,” 情報処理学会第 66 回全国大会 6T-1, 2004.

[3] Jian Huang, “The SImulator for Multithreaded Computer Architecture, Release 1.2,” http://www-mount.ee.umn.edu/~lilja/SIMCA/index.html