5ZB-6

マルチスレッドコード向け命令レベル最適化ツールの開発

阿久津 徳寿　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。

このシステムにおいてより高い実行性能を持つコードを生成するためには、効果的なスレッドコード最適化技術が必要となる。そこで、本研究ではマルチスレッド化の対象とするシングルスレッドコードにマルチスレッド実行の際に有効な最適化を行うツールを開発し、本ツールによって最適化されたコードをマルチスレッド化した場合と、ツールを適用していないコードをマルチスレッド化した場合の実行時間を比較することで、本最適化ツールの有効性について評価を行う。

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。

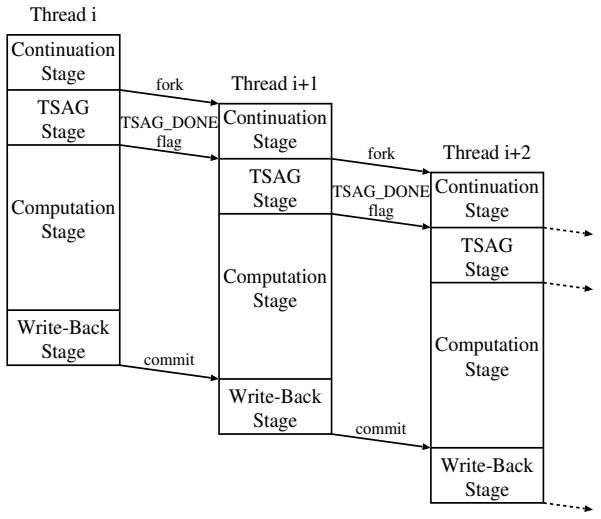

図 1. スレッドパイプライニングモデル

Development of an Optimizing Tool for Multithreaded Code
† Noritoshi Akutsu, Kanemitsu Ootsu, Takashi Yokota, Takanobu Baba
‡ Department of Information Science, Faculty of Engineering, Utsunomiya University

3　命令レベル最適化ツール

本最適化ツールは、前述のスレッドパイプライニングモデルに基づきマルチスレッド化可能なコードのうち、ループを処理の単位としてマルチスレッド化されるコードを最適化の対象としている。シングルスレッドコードのマルチスレッド化を行う前に、マルチスレッド化の対象とするループに本最適化ツール適用し、本ツールによって最適化されたコードをマルチスレッド化することで、より実行性能の高いマルチスレッドコードを得ることが出来る。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は、あらかじめバイナリレベル変数解析 [2] によって取得し、この情報をもとに最適化を行う。

本ツールは 5 つの最適化手法を以下に示す手順で適用することにより最適化を行う。

(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 スレッドに割り当てる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。

(4) **命令のループ外への移動**
ループ外で計算可能な処理をループ外へ移動し、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、マルチスレッド化の際に冗長となるループ終了判定などの命令を削除する。

本最適化ツールの適用例を図 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

```
(*) 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) 適用前

```
    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) 適用後

図 2. 最適化ツール適用例

であることがバイナリレベル変数解析 [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、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

本最適化ツールの有効性を検証するため、本ツールによって最適化されたコードをマルチスレッド化した場合と、最適化されていないコードをマルチスレッド化した場合の性能向上を比較した。

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 train 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 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。

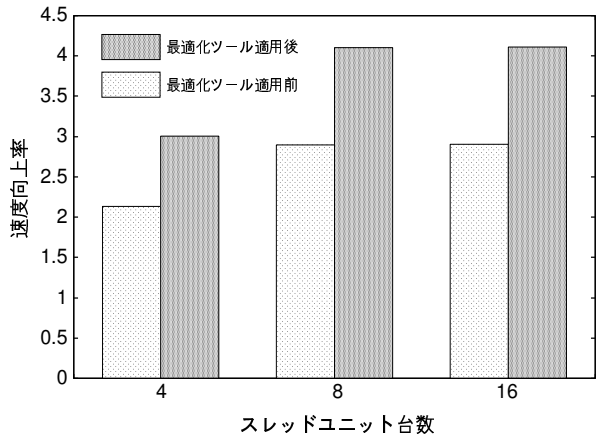

図 3. 107.mgrid における速度向上率

スレッドユニット台数を 4 台、8 台、16 台とし、各台数に対してマルチスレッド化したループ部分の速度向上率＝ (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、各スレッド台数毎に左側が最適化ツールを適用しない場合、右側が最適化ツールを適用した場合の速度向上率を表す。

ツール適用後の速度向上率はスレッドユニット 4 台で 3.00 倍、8 台で 4.10 倍、16 台で 4.11 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、マルチスレッドコード向け命令レベル最適化ツールを開発し、その評価を行った。

今後の課題としては、本稿で示した最適化ツールの有効性を検証するため、評価対象のアプリケーションを増やすことが挙げられる。

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 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] T.Satou, K.Ootsu, A.Tsukikawa, T.Yokota, T.Baba, ”A Methodology of Binary-Level Variable Analysis for Multithreading,” Proc. 16th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2004), pp.784–789, 2004.

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