2H-9

2パス限定投機システムにおけるコードスケジューリング手法とその評価

福田 明宏 † 十鳥 弘泰 † 大津 金光 † 横田 隆史 † 馬場 敬信 †
† 宇都宮大学大学院工学研究科情報システム科学専攻

1 はじめに

我々は投機的マルチスレッド処理方式の一つとして2パス限定投機方式 [1] を提案している．2パス限定投機方式はループ中の実行経路 (パス) のうち実行頻度上位2本に限定した投機的マルチスレッド処理により高速化を達成するものである．さらに同方式を実現するアーキテクチャとして2パス限定投機システム PALS[2] を設計し，PALS 向けの並列化コードをアセンブリレベルで生成するコード生成器の開発を行っている．

一般にマルチスレッド実行ではスレッド間の同期待ち時間やスレッド制御のオーバーヘッドが速度向上の妨げとなっており，これは PALS においても例外でない．そこで本稿では PALS 上で効率の高い並列実行を実現するために PALS 向け並列化コードに対応したコードスケジューリング手法を提案する．そして PALS システムシミュレータを用いて評価を行う．

2 PALS 向け並列化コード

2.1 2パス限定投機システム PALS

2パス限定投機システム PALS ではループ中の実行経路 (パス) のうち最も実行頻度の高いパス (#1 パス) と 2 番目に実行頻度の高いパス (#2 パス) に沿って基本ブロックを集約したコード (投機スレッドコード) と元のループ構造を保持したコード (非投機スレッドコード) を生成する．プログラム実行時にはこれらのパスのどちらが実行されるかをイテレーションごとに予測しながら投機的にマルチスレッド実行し，高速化を達成する．投機に失敗した場合はスレッドの破棄と，実行前にレジスタの状態を書き戻す回復処理を行い，もう一方のパスを投機する．2回目の投機にも失敗した場合には非投機スレッドコードを実行する．

2.2 PALS 向け並列化コード

PALS 向けの並列化コードはループごとに2つの投機スレッドコードと非投機スレッドコードから構成されている．投機スレッドコードにはスレッドの終了を示す thread end bit とレジスタデータ送信を実現する forward bit という制御ビットを命令に付加する．thread end bit は投機スレッドコードの最後の命令に付加する．forward bit は投機スレッドコード中で各レジスタに対して最後に書き込みを行う命令に付加する．また，投機の成否を判定する assert 命令を分岐命令に置き換えて挿入する．非投機スレッドコードにはイテレーション境界の命令に thread end bit を付加する．また，PALS ではループにのみマルチスレッド実行するためループ外に制御が遷移する位置にマルチスレッド実行終了を宣言する stop2path 命令を挿入する．さらにマルチスレッド実行開始を宣言する start2path 命令をループ内に制御が遷移する直前に挿入する．図 1 に PALS 向け並列化コードの例を示す．各投機スレッドコードの */.fwd* や */.end* はそれぞれ forward bit，thread end bit を表す．また，ane 命令や aeq 命令は assert 命令である．

```
    .set noat
    la $1,thrdstruct
    start2path $1
    .set at
# == loop #1path ==
$L_no1path:
    lw          $2,0($3)
    ane         $2,$0
    addu/.fwd $5,$5,$4
    addu/.fwd $3,$3,$8
    slt         $2,$7,$5
    aeq/.end  $2,$0
```

#1投機スレッドコード

```
# == loop #2path ==
$L_no2path:
    lw          $2,0($3)
    aeq         $2,$0
    sw          $10,0($3)
    addu/.fwd $5,$5,$4
    addu/.fwd $3,$3,$8
    slt         $2,$7,$5
    aeq/.end  $2,$0
```

#2投機スレッドコード

```
# == non_spec ==
$L_nonspec:
$L27:
    lw      $2,0($3)
    bne     $2,$0,$L28
    sw      $10,0($3)
$L28:
    addu  $5,$5,$4
    addu  $3,$3,$8
    slt     $2,$7,$5
    beq     $2,$0,$L_end
    stop2path   $L22
$L_end:
    j/.end   $L_nonspec
```

非投機スレッドコード

図 1: PALS 向け並列化コードの例

3 コードスケジューリング手法

PALS のプログラム実行において速度向上の妨げの要因となるのは同期待ち時間とスレッド制御のオーバーヘッドである．PALS 上での効率の良い並列実行を実現するためにこれらの要因を緩和するコードスケジューリング手法を述べる．

3.1 同期待ち時間削減のための命令移動

同期待ち時間削減のためには同期待ちしているデータを早期に生成して実行再開を早める方法と，同期待ちするタイミングを遅らせる方法が挙げられる．前者を実現するためにはスレッド間で依存変数の値を定義する命令 (値定義命令) を前方移動することが必要となる．後者を実現するためにはスレッドの最初で依存変数を使用する命令 (値使用命令) を後方移動することが必要となる．

3.2 投機失敗による遅延低減のための命令移動

スレッド制御オーバーヘッドのうち投機失敗による遅延低減のための assert 命令の命令移動について述べる．assert 命令が不成立であった時点で回復処理を開始する．そのため投機失敗の検出が遅くなればなるほど投機実行の再開までに時間を要する．そこで assert 命令の前方移動を行う．assert 命令を前方移動することで速やかに投機失敗を検出し，投機失敗に要するオーバーヘッドを低減する．

A code scheduling method and its evaluation for Two-Path Limited Speculation System

†Akihiro Fukuda, Hiroyoshi Jutori, Kanemitsu Ootsu, Takashi Yokota and Takanobu Baba
Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University (†)

3.3 投機スレッドコードのスケジューリング手順

同期待ち時間削減や投機失敗による遅延低減のための命令移動アルゴリズムについて述べる．命令移動を行う際には命令間のデータ依存や制御依存を満足する必要がある．しかし，投機スレッドコードはパスに沿って基本ブロックを集約したコードであり，分岐命令を一切含まないため制御依存を考慮する必要はなく，データ依存のみを考慮した命令移動を行えばよい．本手法ではリストスケジューリング [3] を基に同期待ち時間削減や投機失敗による遅延低減のためのコードスケジューリングを行う．

まず，命令間のデータ依存を示したデータ依存グラフを構築する．データ依存グラフを用いた命令移動アルゴリズムは以下の通りである．

(1) グラフ中で先行ノードを持たないノードを配置可能集合に入れる．
(2) 配置可能集合の中から資源の競合のない命令を配置する．候補が複数ある場合には以下に示す優先順で命令を配置する．
  (i) 値定義命令
  (ii) クリティカルパス上の命令
  (iii) assert 命令
  (iv) 通常の命令 (i〜iii と v の条件を満たさない)
  (v) 値使用命令
(3) (2) によって使用される資源をロックする．
(4) サイクルカウンタを 1 増やす．そして使用中の資源が使用可能になったならばロックを解除する．
(5) 新たに配置可能となったノードを配置可能集合に加える．配置可能集合が空になったらアルゴリズムを終了する．

4 性能評価

評価には PALS システムシミュレータを用いる．シミュレーションパラメータを表 1 に示す．PALS は表に示すスレッド制御機構，スレッド実行機構，メモリアクセス機構により構成されている．

表 1: シミュレーションパラメータ

マルチスレッド制御機構	スレッド生成 1 サイクル 2 レベル分岐予測を基にした 2 レベルパス予測
スレッド実行機構	4 命令同時実行 アウトオブオーダ スレッド間同時通信可能レジスタ数 4
メモリアクセス機構	レイテンシ 1 サイクル エントリ数 32
キャッシュ	4-way セットアソシアティブ 置換方式 LRU サイズ 16k バイト 命令・データ分離 レイテンシ 2 サイクル
メインメモリ	レイテンシ 100 サイクル
ハードウェア機構間通信	単方向送信レイテンシ 1 サイクル

評価対象には *SPEC CINT2000* アプリケーション *181.mcf* の関数 *dual_feasible()* と関数 *primal_bea_mpp()* それぞれに含まれるループを使用した．ループはパスが 2 つ以上存在し，関数呼び出しを含まず，ループ構造を含まないものを対象とした．図 2 に速度向上率を示す．速度向上率は逐次実行したときの実行サイクル数をマルチスレッドの実行サイクル数で割ったものである．TU 台数はスレッドを実行するユニットの台数を示しており，TU 台数が 2 であった場合には同時に 2 スレッドまで実行できる．

dual_feasible() ではスケジューリングの適用により速度向上率に改善はみられたものの速度低下を引き起こしている．速度低下の要因として投機スレッドコードの命令数が 20 命令程度と少なく，パス予測やスレッド生成のオーバーヘッドが 1 スレッドの実行に対して相対的に大きくなってしまったことが挙げられる．一方，*primal_bea_mpp()* ではスケジューリングの適用により TU2 台で速度向上を達成した．これはスケジューリング適用により同期待ち時間が大きく削減されたためである．

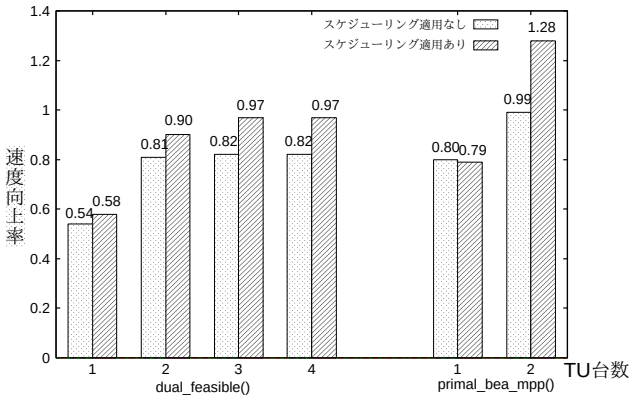

図 2: 速度向上率

5 おわりに

本稿では 2 パス限定投機システム上の効率の良い並列実行を支援するために同期待ち時間の削減やスレッド制御オーバーヘッドの低減を狙ったコードスケジューリング手法を提案した．そして提案手法を用いた実験ではコードスケジューリング適用前と比べて速度向上を達成した．

今後の課題としてはメモリ依存に対応することが挙げられる．

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 (C)20500047，同 (C)21500049，同 (C)21500050) および宇都宮大学若手萌芽的研究プロジェクトの援助による．

参考文献

[1] 横田 隆史ほか: “2 パス限定投機方式の提案”, 情報処理学会論文誌: コンピューティングシステム, Vol.46, No.SIG 16(AGS-12), pp.1-13, 2005.

[2] 十鳥 弘泰ほか: “2 パス限定投機方式を実現するマルチコアプロセッサ PALS の提案”, 信学技報, Vol.109, No319(CPSY2009-46), pp.19-24, 2009.

[3] 中田 育男: ”コンパイラの構成と最適化”, 朝倉書店, pp.358-362, 1999.