バイナリ変換による投機的マルチスレッド化方式の検討

横田 昌之　安濃 隆弘　佐藤 智一　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をバイナリレベルでマルチスレッドコードに変換し，アプリケーションの性能を向上させるシステムを研究開発してきた[1]．

マルチスレッド実行において，スレッド間に依存が存在する場合，同期をとるためにスレッドの実行を停止させなければならず，実行性能が低下する．この問題は，特定のパスに沿ってスレッドの実行を投機的に開始することにより制御依存を削除する制御投機実行と，スレッド間依存のあるデータの値を予測し，スレッド間依存を解消するデータ投機[2]を用いることにより緩和できる．

本稿では，制御投機に焦点を当て，バイナリレベルでの投機的マルチスレッド化方式の，並列性の抽出が困難な整数系アプリケーションを対象に適用を検討する．

2　投機的マルチスレッド化

本研究では，マルチスレッド実行モデルとして，各スレッドが，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つのステージから成るスレッドパイプライニングモデル[3]をベースとする．

制御投機を用いたマルチスレッド実行では，ループの各イテレーションについて頻繁に実行されるパスに沿ってマルチスレッド化を行うことにより，スレッド間の制御依存を解消する．

図1(a)の制御フローグラフで表されるループを例に，制御投機実行の進め方を説明する．ここで，A〜Gはプログラムの基本ブロックを表している．ループのイテレーションを制御投機を用いずにスレッドに割り当てる場合，ブロックFからブロックDへの逆フロー依存があるとすると，スレッド間に依存が存在することになる．例えばB→C→Fというパス（BCFとする）に沿ってマルチスレッド実行を行うと，この依存は削除される．BCFに沿って投機実行を行う場合，以下に示す(1)〜(6)の手順で実行を進める（図1(b)）．

(1) ループの先頭で，BCFを通るか否かを判定する．(2) BCFを通らないと判定された場合，この１イテレーション分はシングルスレッド実行を行い，(1)に戻る．(3) BCFを通ると判定された場合，このパスに沿ってマルチスレッド実行を開始する．(4) Computation Stage終了後，次スレッドの投機の成功/失敗を判定する．(5) 失敗と判定された場合，全ての後続スレッドを破棄し，シングルスレッド実行に移る．(6) 成功と判定された場合はマルチスレッド実行を続ける．

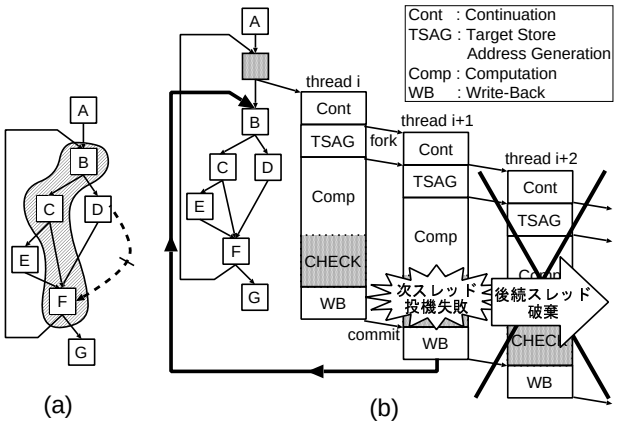


図 1. BCFに沿った制御投機実行

3　評価

前節で述べた制御投機実行の有効性を検証するために，制御投機を用いて投機的にマルチスレッド実行を行うコードと，投機を行わないマルチスレッドコードをバイナリレベルで作成し，それら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**[4]を用いた．また，対象となるバイナリコードはSIMCA用gccクロスコンパイラ（version 2.7.2.3）に最適化オプション-O3を適用し生成した．対象アプリケーションとして，SPECint95のcompressを用い，入力データセットはtestとtrainを使用した．マルチスレッド化の対象とした，関数compress()内のループ部分の制御フローグラフを図2に示す．各入力データに対して実行頻度の高いパスをプロファイリングにより求め，その中で実行頻度の高い２つのパスI→J→Q（以下IJQ），I→J→K→S→T（以下IJKST）それぞれに沿ってマルチスレッド化を行った．ここで，ブ

A Consideration of a Speculative Multithreading Technique with Binary Translation Technology
† Masayuki Yokota, Takahiro Annou, Tomokazu Satou, Kanemitsu Ootsu, Takashi Yokota and Takanobu Baba
‡ Department of Information Science, Faculty of Engineering, Utsunomiya University

ロックIで関数getbyte()，ブロックSで関数output()，ブロックWで関数cl_block()が呼び出されるが，これらの関数の中も同様にプロファイルし，最も頻度の高いパスをIJQ，IJKSTとする．これらのパスの実行頻度（このループの全パスの実行回数に占めるIJQおよびIJKSTの実行回数の割合）は，入力データがtestのときはIJQが50.6%，IJKSTが43.0%，trainのときはIJQが54.5%，IJKSTが22.4%である．

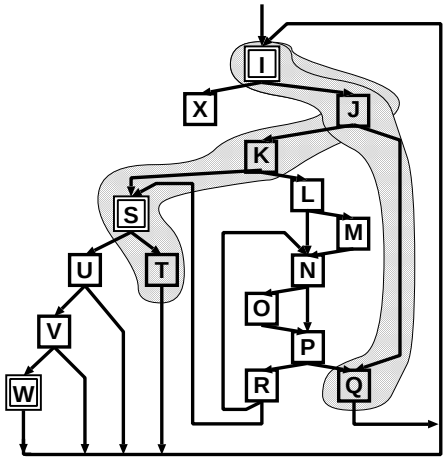
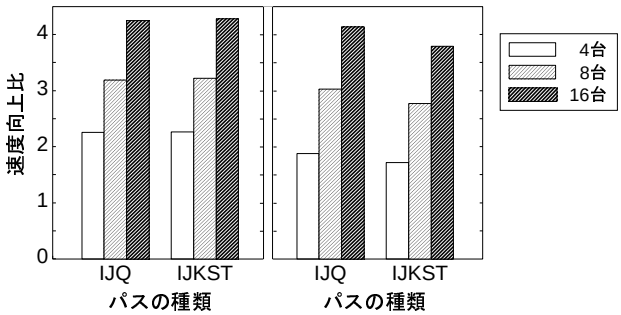


図 2. compress()のループ部分の制御フローグラフ

IJQ，IJKSTそれぞれのパスに沿って投機的にマルチスレッド実行を行った．スレッドユニット台数は4台，8台，16台とし，各台数に対して速度向上比＝(非投機的マルチスレッド実行サイクル数)/ (投機的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．縦軸は速度向上比を表し，横軸はどのパスに沿ってマルチスレッド化したかを表す．データセットtestを入力とした場合，IJQ，IJKSTともにスレッドユニットが4台のとき2.3倍，8台のとき3.2倍，16台のとき4.3倍であった．また，trainを入力とした場合は4台，8台，16台それぞれについて，IJQでは1.9倍，3.0倍，4.1倍，IJKSTでは1.7倍，2.8倍，3.8倍となり，どの条件においてもスレッドユニットが多いほど速度が向上した．これは投機的マルチスレッド実行の場合はスレッドユニット台数によらず実行サイクル数が一定であったのに対し，非投機実行ではスレッドユニットが多いほど実行サイクル数が増加したためである．



(a) データセットtest (b) データセットtrain

図 3. 非投機的マルチスレッド実行に対する速度向上比

投機実行において実行サイクル数が一定であったのは，IJQおよびIJKSTが4回以上連続して実行される回数が少なかったために，すべてのスレッドユニットが並列に実行されていることが少なかったことが原因である．また，非投機実行において実行サイクル数が増加した理由は，スレッド間依存が大きく，同期をとるためにスレッドを停止させる時間が，スレッドユニットを多くするほど長くなったためである．

4 おわりに

本稿では，我々が研究開発しているマルチスレッド化バイナリ変換システムにおいて，SPECint95のcompressを対象とした場合，制御投機スレッドを適用することにより投機を行わない場合と比べてプログラムの実行速度が向上することを示した．

今後の課題として，バイナリ変換による投機的マルチスレッド化方式が有効であるかを，様々なアプリケーションに対して検証することが挙げられる．また，制御投機だけでなくデータ投機を利用したマルチスレッド化手法を適用し，評価をとる必要がある．

さらに，今回は投機スレッド生成の対象を1本のパスに限定したが，複数のパスに沿ってマルチスレッド化し，プログラムの挙動に従いそれらを動的に切り替えることにより更なる速度向上が得られると考えられる．

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)14580362，若手研究14780186）の援助による．

参考文献

[1] K. Ootsu, T. Ono, et al., “A Methodology of Binary-Level Multithreading and its Preliminary Evaluation,” Proc. 14th IASTED International Conference on Parallel and Distributed Computing and Systems, pp.797-802, Nov. 2002.

[2] Pedro Marcuello and Antonio González, “Clustered Speculative Multithreaded Processors,” International Conference on Supercomputing, pp.365-372, 1999.

[3] J. Y. Tsai, J. Huang, et al., “The Superthreaded Processor Architecture,” IEEE Transactions on Computers, Vol. 48, No.9, 1999.

[4] J. Huang, “The SImulator for Multi-threaded Computer Architecture (Release 1.2),” http://www.cs.umn.edu/Research/Agassiz/Tools/SIMCA/simca.html.