5ZB-3

データ投機型マルチスレッド実行方式のSPEC2000ベンチマークへの適用

小林 崇彦　三木 大輔　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するシステムの研究開発を行なっている[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，同期をとる為に後続の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，特定の制御の流れに沿ってスレッド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や，スレッド間依存のあるデータの値を予測し，投機的にプログラムを実行する事でスレッド間依存を解消するデータ投機[2]を用いる事で緩和する事ができる．我々は今までにSPEC95に対しデータ投機を適用してきた[3].

本稿では，データ投機をSPEC2000中で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し，バイナリレベルマルチスレッド化におけるデータ投機の効果をシミュレータにより評価する．

2　データ投機マルチスレッド実行

図1に本稿で検討したデータ投機マルチスレッド実行の流れを示す．初期段階では依存変数の予測性が不明であるので，シングルスレッドで実行を行う．シングルスレッド実行時には依存変数の過去の値を保持しそれにより値予測を行う．予測した値が実際に計算した値と等しいならその依存変数に対し予測性が有ると判断しデータ投機マルチスレッド実行に移行する．

マルチスレッド実行に移行後，各スレッドはスレッド実行の先頭で自スレッドの依存変数を予測し，予測した値を用いてスレッドを実行する．そしてスレッド実行の最後に，実際の次スレッドの値と次スレッドで予測される値とを比較する．この２つの値が等しいならば，予測が正しいことになり，次スレッドの予測も成功でありマルチスレッド実行を継続する．しかし，２つの値が異なる場合，後続スレッドは誤った値で実行されているため投機失敗とし後続スレッドの実行を破棄する．その後，一度シングルスレッド実行に移行し，再び値に予測性があると判断した時，データ投機マルチスレッド実行に移行する．以降これを繰り返す．

Application of Data-Speculative Multithread Execution to some SPEC2000 Benchmarks
† Takahiko Kobayashi,Daisuke Mitsugi, Kanemitsu Ootsu, Takashi Yokota and Takanobu Baba
‡ Department of Information Science, Faculty of Engineering, Utsunomiya University

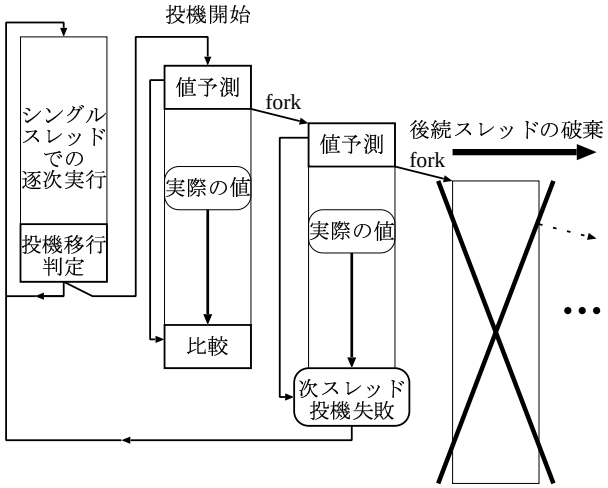

図 1. データ投機の流れ

3　値予測方式

値の予測方式として先行研究[4]で用いられているのは，LastValue，StrideValue，Context-Base，StrideValueとContext-Baseを組み合わせたHybridである．しかし本研究では値予測を全てソフトウェアで行っているため，複雑な予測方式では予測のオーバーヘッドが大きくなる．Context-Baseはバイナリレベルで実装する事は複雑であるため今回はContext-BaseとHybridの２つの値予測を用いる事は見送った．そこで，予測方式としてコストが低いLastValueとStrideValue の２つの方式を用いた．LastValueは前回の変数値をそのまま予測値とし，StrideValueは過去２回分の変数値の差分を前回の変数値に加算し値を予測値する．

4　評価

4.1　評価手順

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[5]を用いた．評価は，対象アプリケーションのソースコードを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を適用してコンパイルする．生成されたバイナリコードに対してデータ投機を行うマルチスレッドコードへ変換する．各スレッドの処理はループイテレーション単位で割り当てるものとする．そのため評価対象には，プログラム中のホットループでループイテレーション間で依存変数を持つものを選ぶ．本稿ではSPECint2000のgzipの関数updcrc()とmcfの関数primal_bea_mpp()の中のホットループ部分をマルチスレッド化する．そしてループをマルチスレッド化する際のスレッド間依存変数に対し

データ投機を適用する．評価は，マルチスレッド実行対象コードにおける実行サイクル数を計り，速度向上率 (＝シングルスレッド実行のサイクル数／マルチスレッド実行のサイクル数) を求めることで評価する．入力データセットにはtestを用い，スレッドユニット台数を4台として測定した．

4.2 評価結果

図2の (a) に関数updcrc() のホットループをスレッド間依存変数に対し同期を行い非投機でマルチスレッド化した場合と，値の予測としてLastValue，StrideValueをそれぞれ用いてデータ投機を適用した速度向上率を示す．速度向上率は非投機では0.26倍となった．これに対しLastValueで0.99倍，StrideValueで0.98倍となり，非投機より速くはなったがシングルスレッド実行よりは遅い結果となった．非投機では同期の為の待ち時間が速度低下を招き，データ投機では予測の成功率 (＝予測の成功したスレッド数／全実行スレッド数× 100) が低いことが速度低下を招いた．関数updcrc() のホットループでは，データ投機を適用したスレッド間依存変数の予測の成功率が0％であった．そのためマルチスレッド実行に移行してもすぐに投機失敗となり実行の並列度が低くなる．よって値予測やマルチスレッド実行のためのオーバーヘッドが並列実行で得られる速度向上より大きくなり速度低下を招いた．

また図2の (b) に関数primal_bea_mpp() のホットループをスレッド間依存変数に対し同期を行い非投機でマルチスレッド化した場合と，値の予測としてLastValueとStrideValueをそれぞれ用いてデータ投機を適用した速度向上率を示す．速度向上率は非投機で1.86倍となった．これに対しLastValueは1.49倍，StrideValueは1.49倍となり非投機に比べ遅い結果となった．データ投機では，予測成功率がLastValueで71％，StrideValueで69％となり高い予測成功率を得ることにより速度向上を得た．しかし非投機に比べて遅くなった理由として，ループ内の制御の流れによっては依存変数を計算に用いることが無いため同期による待ち時間が発生しない事，同期を取る際でも待ち時間が8命令分と短い事，値予測などのオーバーヘッドが挙げられる．

5 おわりに

本稿では，SPEC2000でバイナリレベルでシングルスレッドコードをマルチスレッドコードに変換し，変換されたマルチスレッドコードのスレッド間でデータ依存のある変数に対し，データ投機を適用した．データ投機を適用した場合，予測成功率によりシングルスレッド実行よりも遅くなる事がある．また条件によってデータ投機を適用するよりも同期をとってマルチスレッド化した方が高速化したが，データ投機の適用により最大で非投機の2.39倍の速度向上を得た．

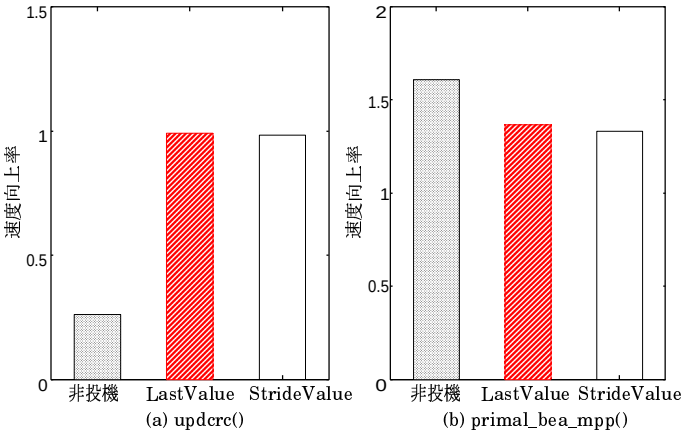

図 2. 速度向上率

本研究により値に予測性が有り，同期による待ち時間が長いループに対しデータ投機が有効であると考えられる．我々の過去の研究でのSPEC95でも同じ傾向がある．よって今後の課題として，データ投機を適用するアプリケーションを増やし，データ投機の有効な条件をより明確にする必要がある．また今回のデータの予測方式として，LastValueとStrideValueを用いたが，値の予測成功率が高い別の予測方式の適用も考えられる．

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)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，Vol.44, No.SIG-1 (HPS 6), pp.70-80, 2003.

[2] J. Gregory Steffan, Christopher B. Colohan, Antonia Zhai and Todd C. Mowry, “Improving Value Communication for Thread-Level Speculation,” HPCA, pp.65-75, 2002.

[3] 三木 大輔，三田 翼，月川 淳，大津 金光，横田 隆史，馬場 敬信，“バイナリレベルマルチスレッド化におけるデータ投機の検討” 情報処理学会第66回全国大会講演論文集, pp.1-127〜1-128, 2004.

[4] Pedro Marcuello, Jordi Tubella and Antonio Gonzalez, “Value Prediction for Speculative Multithreaded Architectures,” International Symposium on Microarchitecture, pp.230-236, 1999

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