バイナリレベル変数解析に基づいた自動並列化システムの実装

白戸 卓志† 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

近年，マルチスレッド実行による高速化が重要となっているが，プログラムによってはソースコードが必ずしも参照できるとは限らない．そこで，我々はバイナリレベルで既存のプログラムをマルチスレッドコードへ変換する自動並列化システム [1] を開発している．

マルチスレッド化において，スレッド間で依存のある変数を正しく検出できることが重要である．しかし，バイナリコードにおいて，レジスタを用いた間接指定によってアクセスするメモリ上の変数については，どの変数にアクセスしているのか判別し難く依存関係を把握することが困難である．この問題に対し我々は，バイナリレベル変数解析手法 [2] を提案した．

本稿では，バイナリレベル変数解析を実装することにより，メモリ上の依存変数の検出に対応した自動並列化システムについて説明し，同システムを用いて生成したマルチスレッドコードの性能評価について述べる．

2　マルチスレッド実行モデル

本研究が前提とするマルチスレッド実行モデルであるスレッドパイプライニングモデルについて述べる．図1に，スレッドパイプライニングモデルの流れを示す．スレッドパイプライニングモデルは，各スレッドが以下の4つのステージで構成される．Continuation ステージでは，ループにおける誘導変数など後続スレッドを起動するために必要な計算を行う．TSAG ステージでは，スレッド間で同期を取る必要がある変数を登録する．Computation ステージでは，計算処理本体のコードを実行する．Write-Back ステージでは，各スレッドの計算結果をメモリへ書き戻す．

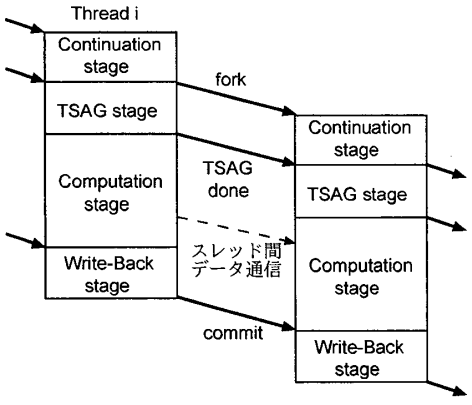

図 1: スレッドパイプライニングモデル

An Implementation of Multithreading System Based on Binary-Level Variable Analysis
†Takashi Shiroto, Kanemitsu Ootsu, Takashi Yokota, Takanobu Baba
Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University (†)

3　従来システムの問題点

自動並列化システムは，変数の依存関係について解析し，そこで得た情報を基にマルチスレッドコード生成を行う．しかし，従来のシステムが行っている解析では，レジスタ上の変数の依存は解析できるのだが，メモリ上の変数の依存解析についてはローカル変数にしか対応していないという問題がある．ここでまず，この問題について詳しく述べる．

バイナリコードにおいては，メモリ上の変数はレジスタを用いた間接指定によってアクセスされる．しかし，レジスタは再利用されるため同一のレジスタが異なる値を格納することがあり，また，異なるレジスタが同じ値を格納することもある．そのため，メモリ上のどの変数に対してアクセスしているのか判別が困難で，依存関係の把握が困難となる．変数の依存関係が把握できなければ，正しいマルチスレッドコードの生成は不可能である．我々の従来システムにおいて，メモリ上の変数の依存解析については，解析が容易なローカル変数に限られていて，大域変数については依存解析の対象外となっている．

この問題に対して，次節で述べるバイナリレベル変数解析手法をシステムに実装することで対処する．

4　バイナリレベル変数解析

バイナリレベル変数解析は，メモリ上の変数の依存解析を行うための手法である．図2に，バイナリレベル変数解析の様子を示す．解析処理の流れとしては，まずレジスタの内容を関係するレジスタと定数から構成される木構造 (以下，データフロー木) で表す．次に，構築したデータフロー木の正規化をする．正規化は，不要ノードの削除，ノードの整列，同類項の要約などの処理で行われ，これにより内容が等価な場合は，全て同じデータフロー木となる．メモリアクセスの際に間接指定に使用されているレジスタのデータフロー木を比較し，データフロー木が同じであればメモリアクセスの対象アドレスも同じであると判断する．図2における二つのデータフロー木は，構成要素であるレジスタ・定数の値が異なるので同一アドレスではないと判断する．

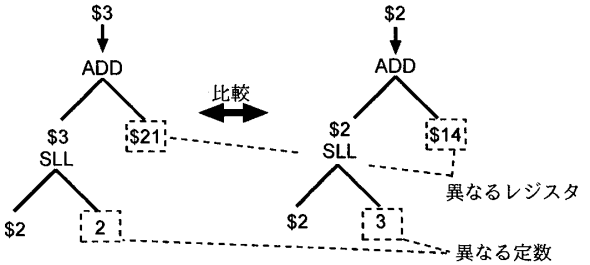

図 2: データフロー木の比較による解析

5 メモリ上の依存変数に対応した自動並列化システム

5.1 メモリ上の依存変数の種類

バイナリレベル変数解析を実装することによって，メモリ上の依存変数が検出可能となる．マルチスレッドコード上では，メモリ上の依存変数があるアドレスを計算する命令を挿入し，TSAGステージでそのアドレスを登録することで対処する．しかし，以下のようなメモリ上の依存変数の場合，マルチスレッド化が不可能となる．

スレッドパイプライニングモデルにおいて，依存変数はcomputationステージにおいて値の受渡しを行う．仮に，メモリ上の依存変数が，このようなcomputationステージで値の受渡しが行われる依存変数をアドレス計算に用いていた場合，TSAGステージではメモリ上の依存変数のアドレスがわからないことになる．しかし，メモリ上の依存変数のアドレスがTSAGステージに登録できなければ，正しいマルチスレッド実行ができない可能性がある．

このようなメモリ上の依存変数を判別・対処できなければ，生成したマルチスレッドコードで正しい実行結果が得られるかどうか保証できなくなる．そこで，次のようにメモリ上の依存変数のアドレス計算の過程に，他の依存変数が含まれるかどうかでメモリ上の依存変数を分類しマルチスレッド化の可否を決定する．

- アドレス計算に他の依存変数が含まれない場合:
  この場合は，メモリ上の依存変数のアドレスは，スレッド毎に変化せず常に固定である．そのため，ループ開始直前にメモリ上の依存変数のアドレス計算を行い，各スレッドはその固定のアドレスをTSAGステージで登録することによって正しいマルチスレッド実行を実現する．
- アドレス計算に誘導変数が含まれる場合:
  この場合は，メモリ上の依存変数のアドレスが，スレッド毎に変化する．しかし，誘導変数の値はcontinuationステージで既に定まっているので，各スレッドはTSAGステージにおいて，その誘導変数の値を基にメモリ上の依存変数のアドレス計算を行い，そのアドレスを登録する処理を行うことによって正しいマルチスレッド実行を実現する．
- アドレス計算に依存変数が含まれる場合:
  メモリ上の依存変数のアドレス計算に必要となる他の依存変数の値がcomputationステージで決まるので，メモリ上の依存変数のアドレスがTSAGステージではわからない．そのため，正しいマルチスレッド実行が行われない可能性がある．このようなメモリ上の依存変数を含むループについては，マルチスレッド化の対象外とする．

5.2 マルチスレッド化の処理の流れ

バイナリレベル変数解析を実装し，メモリ上の変数の依存解析に対応したシステムの処理の流れについて述べる．まず，入力されたバイナリコードを読込み，基本ブロック分割および制御フロー解析を行いループ構造を検出する．そして，並列化対象のループに対してバイナリレベル変数解析を行い，依存変数の検出を行う．そして，この依存変数の情報を基に，正しいマルチスレッド実行を行うためのスレッド制御命令の挿入したマルチスレッドコードを生成する．

6 評価

6.1 評価方法

評価には，スレッドパイプライニングシミュレータであるSIMCAを用いる．対象となるプログラムのバイナリコードは，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"を適用して生成する．そのバイナリコードを，自動並列化システムに入力しマルチスレッドコードを生成する．

評価対象には，SPEC CFP2000の173.appluの関数bltsが含むループを使用した．このループは，大域変数の依存を含むために，従来のシステムのマルチスレッドコードでは正しい実行結果を得られないループである．

評価は，マルチスレッドコードの実行結果の正否，シングルスレッド実行サイクル数との比によって求められる速度向上率によって行う．

6.2 評価結果

マルチスレッドコードを実行した結果，正しい計算結果が得られることを確認した．また，速度向上率についての結果を示した表1を見ると，最大で1.57倍の速度向上を実現しているが，スレッドユニット台数の増加による速度向上は大きくないことがわかる．原因として，このループはイテレーション数が5回であるために，スレッドユニットが8,16台利用可能であっても実際には5台分しか使用しないためである．

表1: マルチスレッド実行による速度向上率

	4台	8台	16台
test	1.21	1.17	1.17
train	1.54	1.57	1.57

7 おわりに

本稿では，メモリ上の依存変数の検出の問題に対し，バイナリレベル変数解析を実装することで対処した自動並列化システムについて述べた．また，同システムの性能評価を行った．今後は，より高速なマルチスレッドコードの生成，適用可能ループの拡大の点から自動並列化システムの高性能化を目指す．

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)および宇都宮大学重点推進研究プロジェクトの援助による．

参考文献

[1] 芝崎 諒, 大津 金光, 横田 隆史, 馬場 敬信 "複雑な制御流に対応したバイナリレベル自動並列化処理の実装", 電子情報通信学会コンピュータシステム研究会(CPSY), 信学技報, Vol.107, No.105, pp.101-106, Aug. 2007

[2] 佐藤 智一, 大津 金光, 横田 隆史, 馬場 敬信 "マルチスレッド化のためのバイナリレベル変数解析手法", 情報処理学会研究報告，計算機アーキテクチャ研究会(2004-ARC-158) Vol.2004, No.48, pp.1-6, 2004.05