C-024

非局所的な入れ子構造のループに対するスレッドレベル並列性の抽出可能性
The Possibility to Extract Thread-Level Parallelism from Non-Locally Nested Loops

上杉 晴信† 布目 淳‡ 平田 博章‡ 柴山 潔‡
Harunobu Uesugi Atsushi Nunome Hiroaki Hirata Kiyoshi Shibayama

1. はじめに

マルチコアプロセッサが商用のマイクロプロセッサとして市販される一方で，まだ多くのプログラムは逐次実行されることを前提として記述されており，マイクロプロセッサの性能を十分に活用できない状況にある．そこで，我々は，逐次実行を前提として記述されたプログラムに対して，実行時にスレッドレベルの並列性を抽出するシステムの開発を進めている．ループの各イタレーションをそれぞれスレッドとして投機的に並列実行し，イタレーション間の依存関係が破壊されていないことを確認した上で，そのスレッドの実行結果をコミットする．本稿では，具体的な並列化の手法を提案する前の準備段階として，スレッドレベル並列性の抽出可能性について調査し，プログラム中のどのループを並列化すべきかを検討する．

2. 多重ループと並列処理粒度

従来より，ループを対象とする並列化手法については多くの研究がなされてきた．近年は，スレッドレベルの並列性を抽出することに関心が高まり，ループのイタレーションをスレッドとして実行する以外にも，手続き（関数）呼び出しを契機としてスレッドを生成したり，基本ブロックをベースにスレッドを生成して投機実行を行う方式が提案されている[1][2][3][4][5].

非数値計算分野のプログラムを含む一般的なプログラムにおいては，ループ本体で手続き呼び出しを行い，その手続きの中でまた別のループを実行することは少なくない．この場合，手続き呼び出しを介して非局所的に多重ループが構成されていることになる．本稿では，このようなプログラムを対象に，非局所的な入れ子構造を成す多重ループのどのループで並列実行できるかを調査する．

多重ループにおいて，どのループで並列化できるかはイタレーション間の依存関係に左右される．一方，外側と内側のどちらのループで並列化可能であるのが望ましいかは，並列処理機構の構造によって異なる．例えば，ベクトルコンピュータの場合は，ベクトル演算との関係で，原則として最内ループのみを並列化の対象とする．上記のスレッドレベル投機実行の場合は，最内ループかそれに近い内側のループに並列化の対象を広げている．しかし，本来は，並列化対象を必ずしも内側のループに限定する必要はない．例として，図 1 に，逐次実行環境下で 2 重ループを実行する場合の制御の流れを模式的に示す．A が内側ループのループ本体の命令列を表わし，B_1 と B_2 が A 以外の外側ループのループ本体を構成する命令列を表わす．内側ループで並列化する場合は，A の部分のみを並列実行し，B_1 と B_2 については逐次実行することになる．Amdahl の法則によれば，逐次実行部分の割合が並列処理効果に大きく影響するので，どちらのループも並列実行可能であるならば，外側のループを並列実行するほうが並列処理効果が高い．

図 1　2 重ループの制御の流れ

投機スレッドの実行開始時および終了時には，データの予測処理やコミット処理をそれぞれ行う必要があり，これらの処理に要する時間がオーバヘッドとなり得る．並列処理粒度の細かい多数のスレッドを投機実行するよりも，粒度の粗い少数のスレッドを投機実行するほうがオーバヘッドが生じる回数は少ない．この点でも，多重ループにおいては，できるだけ外側のループを並列化できることが好ましい．

一般的には，粒度を粗くすると，スレッド間に依存関係を生じさせるデータの量も増加し，投機実行が失敗する可能性も高くなると考えられる．しかし，プログラマがアルゴリズムに基づいてプログラムの並列化を行う場合は，並列化コンパイラが行うような細かいデータの依存解析を行うことはむしろ稀で，粗い粒度でスレッドを生成するよう，最外ループかそれに近い外側のループを並列化することがある．この場合，スレッド間に依存関係がまったく存在しないとは限らないまでも，スレッド間の通信量が少なくなるような，粗い粒度の処理単位の選択が可能であることを示唆している．

投機実行を可能とするためには，コミット前のデータを格納するための機構を備える必要がある．我々は，このようなデータを 1 次キャッシュに格納するとともに，スレッド間でのデータアクセスにおいて逆依存や出力依存の関係を除去するメモリリネーミング機構[6]を既に提案している．本機構はキャッシュあふれにも対応しており，そのデータ量について論理的な制限はない．以降の解析では，本機構を使用することを前提に議論を進める．

3. 調査

3.1 調査環境

PowerPC[7]の命令セットを対象として，マシン命令レベルでプログラム実行のシミュレーションを行い，プログラム中のすべてのループについて，イタレーション間に存在する依存関係を調査する．調査用のテストプログラムには SPEC2006 ベンチマークプログラム[8]の中の 429.mcf と 433.milc（入力データセットは test）を用いる．

†京都工芸繊維大学大学院工芸科学研究科情報工学専攻
‡京都工芸繊維大学大学院工芸科学研究科情報工学部門
Dept. of Information Science, Kyoto Institute of Technology

今回は，ループを正確に検出するために，GNU コンパイラを改変して，ループの出入り口にアノテーション用のコードを挿入する．

ループ中のあるイタレーションの実行において，先行するイタレーションで生成したメモリデータを読み出すロード命令の数を計測し，これを**依存関係の強度**と定義する．強度が大きいほど，スレッドを投機的に実行してもアボートしなければならない状況が多く発生すると考えられる．従って，依存関係の強度が十分に小さいループを並列実行の候補に選択するのが望ましい．

3.2 調査結果

手続き呼び出しを介して非局所的に多重ループが構成されているプログラムにおいて，そのループが入れ子構造の何階層目かを表わすために，**ループのレベル**を定義する．最外ループのレベルを 1 とし，1 階層深くなるごとにレベルを 1 ずつ増加する．静的に同一のループであっても，そのループが記述されている手続きがどの手続きから呼び出されるかによって，動的には異なるレベルのループとして識別する場合がある．

429.mcf の実行中のある時点において，多重ループの各レベルにおけるイタレーション間の依存関係の強度を示したグラフが図 2 である．このグラフにおいて，イタレーション間の依存関係の強度が小さいループは，レベルが 3 と 4 のループである．このケースでは，レベルが 3 のループのループ本体で関数呼び出しを行い，その呼び出された関数内に記述されたループをレベル 4 のループ（最内ループ）として実行する．レベル 1，2 のループではイタレーション間の依存関係の強度が大きいことから，内側のループで並列投機実行するのが得策である．ただし，レベル 3 のループを並列化の対象として選択するためには，関数呼び出しを介する非局所的な多重ループの構造を把握して依存解析を行う必要がある．

433.milc について，図 2 と同様のスナップショットを図 3 に示す．図 3 のケースでは，プログラム全体で動的に 8 重ループを構成しており，レベル 2，4，7，8 のループにおいて，イタレーション間の依存関係の強度が小さい．ただし，レベル 2 のループについては，実際の繰り返し回数が 1 回であるため，並列化の対象から除外しなければならない．レベル 4 のループ本体には，図 3 のレベル 5 のループとは静的に異なる複数のループ（レベル 5 以降）や関数呼び出しを含んでいる．従って，このレベル 4 のループを並列実行すれば，レベル 7，8 のループを並列実行するよりも，プログラムの全実行時間に対する逐次実行割合を削減できる．依存関係の強度が 0 ではないので，投機実行方式としてこの点に注意する必要はあるものの，このように，外側のループを並列化対象として選択できる場合があることが確認できた．

4. むすび

本稿では，逐次プログラム中のループのイタレーションをスレッドとして並列投機実行する場合を想定して，関数呼び出しなどを介して非局所的な入れ子構造を成す多重ループのどのループで並列実行が可能かを，イタレーション間に存在するデータの依存関係をもとに調べた．SPEC CPU2006 の中のプログラムに対して調査した結果，

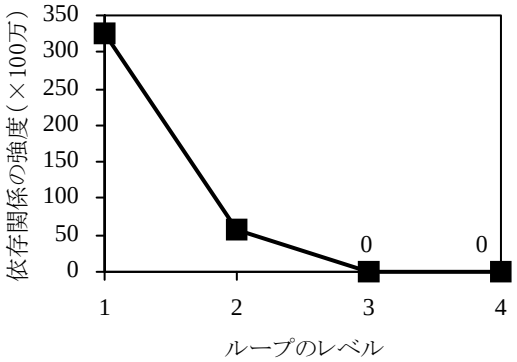

図 2　依存関係の強度（429.mcf）

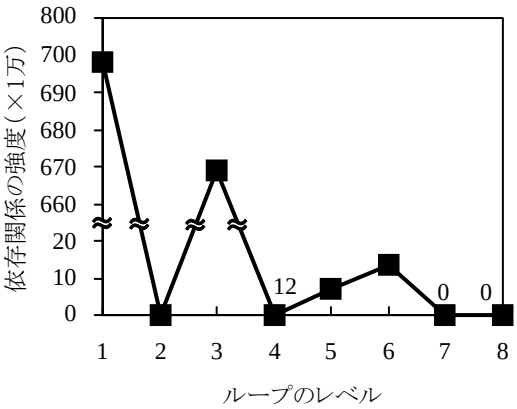

図 3　依存関係の強度（433.milc）

内側のループだけでなく，比較的外側のループからも並列性を抽出できる場合があることを確認した．

今後は，並列化対象として最適なループを検出する具体的な手法を開発する．

謝辞

本研究の一部は日本学術振興会科学研究費補助金（基盤研究（C）25330058）の補助による．

参考文献

[1] G. S. Sohi, S. E. Breach, T. N. Vijaykumar, “Multiscalar Processors,” Proceedings of the 22nd Annual International Symposium on Computer Architecture, pp. 414-425 (1995).
[2] H. Akkary, M. A. Driscoll, “A Dynamic Multithreading Processor.” Proceedings of the 31st Annual International Symposium on Microarchitecture, pp. 226-236 (1998).
[3] L. Hammond, B. A. Hubbert, M. Siu, M. K. Prabhu, M. Chen, K. Olukolun, “The Stanford Hydra CMP,” IEEE Micro, Vol. 20, No. 2, pp. 71-84 (2000).
[4] T. Ohsawa, M. Takagi, S. Kawahara, S. Matsushita, “Pinot: Speculative Multi-threading Processor Architecture Exploiting Parallelism over a Wide Range of Granularities,” Proceedings of the 38th Annual International Symposium on Microarchitecture, pp. 81-92 (2005).
[5] B. Hertzberg, K. Olukotun, “Runtime Automatic Speculative Parallelization,” Proceedings of the 9th Annual International Symposium on Code Generation and Optimization, pp. 64-73 (2011).
[6] 平田博章, 藤井崇弘, 藤皓平, 森田清隆, 布目淳, 柴山潔, “スレッドレベル並列投機実行のためのメモリリネーミング機構”, 第 11 回情報科学技術フォーラム論文集, Vol. 1, pp. 31-34 (2012).
[7] Freescale Semiconductor, “Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture” (2001).
[8] Standard Performance Evaluation Corporation, “SPEC CPU2006,” http://www.spec.org/cpu2006/.