計算機アーキテクチャ 139－27
（2000．8．5）

ループの並列投機実行におけるデータ値予測の予備的評価

安達 浩次†, 山村 周史†, 平田 博章†, 新實 治男‡, 柴山 潔†

† 京都工芸繊維大学 工芸学部 電子情報工学科
〒606-8585 京都市左京区松ヶ崎御所街道町
‡ 京都産業大学 工学部 情報通信工学科
〒603-8555 京都市北区上賀茂本山

本稿では，ループに対してデータ値予測を用いて並列かつ投機的に実行する命令実行方式の可能性に関する予備的評価を行なう．ループイタレーション間のデータ依存関係に対してデータ値予測を用いることにより，同期による待ち時間またはデータ投機実行失敗の発生回数の削減を図る．シミュレーションによる評価の結果，スレッドレベル並列性の抽出にデータ値予測を用いることの有効性を確認した．

Preliminary Evaluation of Data Value Prediction for A Speculative Parallel Execution of Loop Iterations

Koji Adachi†, Shuji Yamamura†,
Hiroaki Hirata†, Haruo Niimi‡, Kiyoshi Shibayama†

† Dept. of Electronics and Information Science, Kyoto Institute of Technology
Matsugasaki, Sakyo-ku, Kyoto, 606-8585 JAPAN
‡ Dept. of Information and Communication Sciences, Kyoto Sangyo University
Motoyama, Kamigamo, Kita-ku, Kyoto, 603-8555 JAPAN

In this paper, we preliminarily evaluate the effectiveness of an execution scheme which employs a data value prediction in the parallel and speculative execution of coarse-grained loops. The data value predictim for inter-loop dependent data is respected to reduce the waiting time in inter-thread synchronization or the frequency of data speculation failure. Our simulation results show the value predicim is very promising to extract much more thread-level parallelism.

1 はじめに

現在まで，マルチスレッドアーキテクチャを対象とした様々なループの並列投機実行方式が提案されている [5]．これら既存の方式では，スレッド間にデータ依存関係が存在した場合，依存関係が解消するまで当該命令の実行を一時的に中断 (同期) するか，依存関係が存在しないものと仮定して命令を投機的に実行 (データ投機) する．しかし，この場合，同期またはデータ投機の失敗により発生する状態復帰処理のオーバヘッドが大きく，高い性能向上率は望めない．

そこで，本研究では，スレッド間のデータ依存関係に対してデータ値予測を適用し，この問題の解決を試みる．データ依存関係にある命令に対し，データ値予測が的中する場合，同期による待ち時間，またはデータ投機実行失敗時にかかる時間を削減することが可能となるため，実行時間の大幅な短縮を図

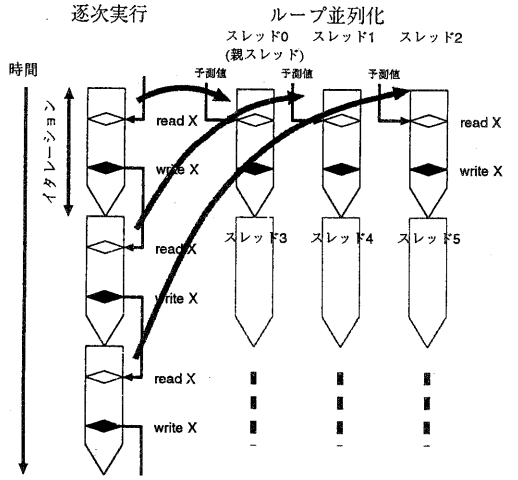

図 1: ループの並列投機実行

ることができる.

本稿ではこの方式の実現のための予備的評価として，スレッド間のデータ値予測によって，データの同期待ちまたはデータ投機の失敗がどれだけ削減可能であるかについて調査する.

以降，2 節で本稿で行なった予備的評価の前提となる条件について述べ．それを基にした予備的評価の結果を報告する．つづく 3 節で関連研究を紹介し，最後に 4 節でまとめとする.

2 予備的評価

2.1 評価条件

2.1.1 前提とする並列実行方式

本稿で扱うプロセッサ全体のアーキテクチャは文献 [1] で述べられたようなマルチスレッドプロセッサをベースに，これを逐次プログラムの並列実行が可能なように改良する方向で議論を進める．例えば，データに関する投機実行を行なうため，メモリアクセスにおいてデータの依存関係をチェックする機構，依存関係が犯された場合の回復機構を設けていることを前提とする.

ループの各イタレーションをスレッドとして扱い，データ依存関係に対してデータ値予測を適用し，投機的実行を行なう (図 1).

各スレッドに関しては，逐次実行をもとに世代定義を行ない，スレッド間での世代管理を行なうものとする．本稿では，現在実行中のスレッドの中で，実行が確定しているスレッドのことを親スレッドと呼び，また，投機実行中のスレッドのことを子スレッドと呼ぶことにする．プロセッサ内に親スレッドは一つしか存在しない.

2.1.2 前提とするデータ予測方式

我々の目的は，1 節で述べたように，データ予測を用いてスレッドレベル (ループイタレーションレベル) の並列性をより多く抽出することにある．この観点からデータ予測について検討する場合，

- スレッド間のデータ依存関係の存在をどのようにして識別するか.
- 依存関係にあると判断した (あるいは予測した) データの値をどのようにして得るか.

の 2 点にわけて考える必要がある.

データ依存関係

スレッド間のデータ依存関係には，レジスタアクセス上の依存関係とメモリアクセス上の依存関係の 2 種類が存在する．レジスタアクセス上の依存関係はコンパイラによって静的に解析可能であり，本稿ではコンパイラによる依存解析を前提とする．つまり，どのレジスタが依存関係にあるかという情報をコンパイラで生成し，これをプロセッサが利用する手段 (例えば，専用命令等) を備えているものと仮定する.

一方，メモリアクセス上の依存関係はコンパイラで完全に解析することは困難である．本稿では，ロード命令のアクセス対象となるデータアドレスについて，次の 2 つの条件:

- 自スレッドでストアしたことがない.
- 親スレッドよりも古い世代においてストアしたことがある.

を実行時に動的にチェックし，この 2 つの条件を同時に満たす場合に限り，スレッド間で依存関係があるものと予測しそのロード値の予測を行う．それ以外の場合は実際にメモリからロードした値を用いて計算を進める.

レジスタデータ値予測

本稿で用いるレジスタデータの値予測手法はStride予測方式[3]に基本的な発想を得ている．しかし，レジスタ値の履歴テーブルは，命令アドレスではなく，レジスタ番号でインデックスする．各エントリには親スレッドの開始時のレジスタ値とstride値を格納する．

親スレッドからi世代後のスレッドのレジスタデータの予測値は，以下の式で計算する．

$$\text{予測値} = (\text{親スレッドのレジスタ値}) + (\text{stride 値}) * i$$

メモリデータ値予測

メモリデータの値予測手法も基本的にStride予測方式を採用する．ただし，通常のメモリデータ予測と異なり，履歴テーブルのインデックスにはデータアドレスを用いる．

ロード命令のデータアドレスに対して，親スレッドよりも古い世代のスレッド(これらのスレッドの実行の終了は確定している)の中で，そのアドレスにストアした最も若いスレッドがi世代前のスレッドであった場合，ロードデータの予測値は以下の式で計算する．

$$\text{予測値} = (\text{履歴テーブル中のストア値}) + (\text{stride 値}) * i$$

2.2 評価方法

評価はSPARCプロセッサの機能シミュレータを用いて行なった．本稿で使用したこのシミュレータはプロセッサのハードウェア構成を反映したシミュレーションを行なえないため，実行時間を見積もることは不可能である．このため，スレッドの生成ポリシを次のように決定した．すなわち，すべてのスレッドは同一時間で実行を終了するものとして，並列度が許す限り並列に実行を開始するものとした．例えば，図1の例では並列度が3であり，スレッド0,1,2が同時に実行を開始し，同時に終了する．その後，スレッド3,4,5が同時に実行を開始する．

2.2.1 レジスタデータ予測の評価方法

本方式では予測が必要なレジスタの識別は，コンパイラで依存解析を行ない選出するが，今回のシミュレーションでは事前に評価プログラムを実行してプロファイリングを行ない，手作業で依存解析を行なって，ループ間で依存関係のあるレジスタを洗い出した．

具体的には次の条件を満たすレジスタをデータ依存関係があるレジスタと判定した．

- イタレーション内の当該レジスタへの最初のアクセスがRead
- イタレーション内で，当該レジスタへのWriteが存在

シミュレーションでは，レジスタデータの予測成否を，レジスタデータの予測値と先行スレッドのスレッド終端でのレジスタ値とを比較して判断した．レジスタデータの予測成功率を以下の式で定義する．

$$\text{予測成功率} = \frac{\text{予測成功回数}}{\text{全予測回数}}$$

2.2.2 メモリデータ予測の評価方法

まず，比較対象として，本稿のデータ予測を用いずに，メモリデータに依存関係が存在しないものと仮定して，イタレーションを並列に投機実行する場合について考える．図2(a)に示すように，あるロード命令に関して，同時に実行している先行スレッドの中に，ロードアドレスと同じアドレスへのストアが存在していれば，依存関係が破壊されて投機実行が失敗し，再実行を余儀なくされる．アグレッシブな投機実行方式では，例えば図2(b)のような関係の場合，たまたま投機に失敗していないものを扱うことが可能かもしれない．しかし，本実験では時間ベースのシミュレーションが不可能であり，またプロセッサのハードウェア構成等にも影響される要因

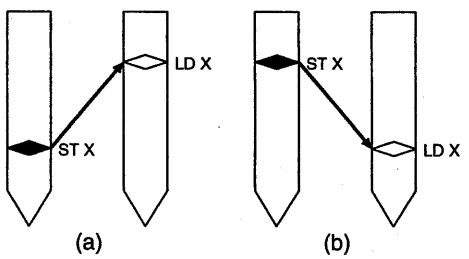

図2: バイオレーションの判定

でもあるため，図 2(b) の場合も投機失敗として集計するものとする．

結局，投機失敗率を以下のように定義する．

$$\text{投機失敗率} = \frac{\text{再実行の原因となるロード命令の数}}{\text{動的な全ロード命令数}}$$

依存関係を破壊する可能性のあるロード命令に対してデータ値予測を行なう場合，データ値予測が的中すれば再実行の必要性はない．この場合の投機失敗率も上式で定義されるが，分子の値は小さくなるものと期待される．

なお，今回はメモリデータ予測に用いる履歴テーブルのサイズは無限大とする．

2.2.3 評価プログラム

評価用のアプリケーションとして，SPECint95 ベンチマークの中から表 1 に示す 2 つのプログラムを使用した．各プログラムは最適化オプション-O2 とレジスタウィンドウを使用しないオプションを付けて Gnu C コンパイラで UltraSPARC 用にコンパイルした．

表 1: 評価プログラム

Program	Description	Input Set
m88ksim	Moto88K simulator	ctl.in
ijpeg	Graphic compression	specmun.ppm

シミュレーションでは，ループの始端と終端を手作業で選び出し，各プログラムのループ部分を抜き出して評価を行なった．ループの選択ポリシは，ループサイズが適度に大きなことと十分な繰り返し回数が存在することである．実際には，m88ksim については M88000 シミュレータ内の 1 つの命令の実行を単位とし，また，ijpeg では原画像の 1 ブロック分の DCT を単位として，それぞれスレッドとした．

2.3 評価結果

ここではレジスタデータ予測とメモリデータ予測のそれぞれに関する評価結果を述べる．なお，並列実行スレッド数が 2 と 4 の場合の 2 通りについて評価した．

2.3.1 レジスタデータ予測

データ依存関係が存在するレジスタの本数を表 2 に，また，レジスタデータの予測成功率を表 3 に示す．

表 2: データ依存関係の存在するレジスタの本数

Program	レジスタ数
m88ksim	8
ijpeg	8

表 3: レジスタデータ予測成功率

Program	並列度	予測ヒット率 (%)
m88ksim	2	99.99
m88ksim	4	99.99
ijpeg	2	97.07
ijpeg	4	97.07

SPARC アーキテクチャでは 32 本の汎用レジスタを搭載しているので，表 2 より全汎用レジスタのうち 4 分の 1 がデータ依存関係があると判定されている．また，表 3 の結果から，レジスタデータに関しては，データ値予測がほぼ正確にヒットすることが分かる．詳細に調べると，依存解析により依存関係があると判定されたすべてのレジスタについて，ループの先頭でのレジスタ値の変化は全く起こっていない．実際，予測に失敗したのは初期時のみである．

これより，データ依存関係があると判定されても，その中にはスタックポインタやフレームポインタなども含まれ，本質的な依存関係は存在しなかったことがわかる．その原因は，本稿で並列化を行なったループのサイズが十分に大きかったためである．

2.3.2 メモリデータ予測

図 3にメモリデータ予測による投機失敗率削減の効果を示す．図中の各プログラムに関するグラフのうち，左側はデータ予測を用いなかった割合，右側はデータ予測を用いた場合の結果である．

図 3から，一般に同時実行するスレッド数が多いほど，メモリデータに起因する投機失敗の発生割合が多くなることが分かる．

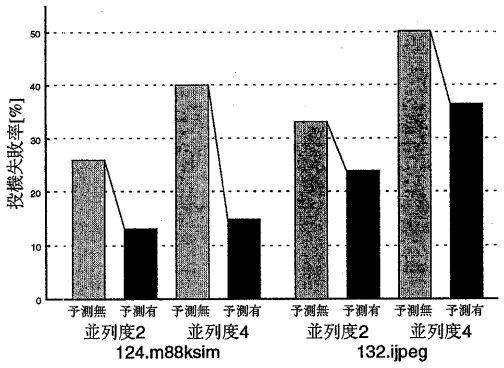

図 3: メモリデータ予測の効果

また，データ予測を行なう場合と予測を行なわない場合とを比較すると，データ予測によって投機失敗する可能性を削減できることは明らかである．

表 4: ストア履歴テーブルの使用エントリ数

Program	エントリ数	未使用率
m88ksim	8,490	76.0%
ijpeg	3,556	13.6%

なお，今回の評価では，予測に用いるメモリデータの履歴テーブルのサイズを無限大と仮定したが，実際に使用したエントリの数は表4のようになった．表の3カラム目は，テーブルに登録されたものの，一度も予測に使用されなかったエントリが全エントリ数に占める割合を示している．つまり，m88ksimの場合，テーブルに登録されながら予測に使用されなかったエントリが，実に76.6%も占めていたことを示している．

これらのデータ予測に使用されないエントリは履歴テーブルに登録しても，性能向上には影響を与えない．実際にはテーブルのサイズは有限であるため，場合によっては，予測に必要なデータの登録を妨げ，予測ヒット率の低下を招く恐れもある．そこで，不必要なエントリの履歴テーブルへの登録を抑制する方法を検討する必要がある．

3 関連研究

これまで，データ予測に関する多くの研究が行なわれている [2, 3]．しかし，そのほとんどは命令レベル並列性を抽出するという観点からの議論である．スレッド間のデータ依存関係に関する予測の研究 [4] も行なわれてはいるが，依存関係の存在のみを予測するものであり，データ値そのものの予測は検討されていない．

筆者らの知る限り，スレッドレベル並列性の観点でデータ予測を検討しているのは，Marcuello ら [6] による報告だけである．彼らの研究ではメモリデータ値予測により得られる性能向上が，レジスタデータ値予測により得られる性能向上に比較して小さいと考え，レジスタデータに着目して，予測方式を検討している．

本稿の結果が Marcuello らの内容および結果と異なるのは，彼らの研究では，イタレーションサイズが数十命令ほどの小さいループ (最内ループ) を並列実行の対象としていることによる．つまり，サイズの小さなループでは，メモリアクセス上の依存関係が少ない一方，帰納変数などがレジスタ上に割り付けられ，レジスタアクセス上の依存関係が多く存在する傾向がある．その結果，レジスタデータの予測が性能向上に大きく影響することになる．

これに対して，我々が並列実行の対象としたループは各イタレーションのサイズが数千命令に及ぶ大きなものであり，より粒度の粗い並列処理における投機実行の可能性を検討することを目的にしている．ループのサイズが大きくなるほど，2節で見たようにレジスタを介した依存関係は減少するが，逆に，メモリを介した依存関係は増加する．

4 おわりに

本稿では，ループの並列投機実行を行なう際に問題となるスレッド間のデータ依存関係に対し，データ予測を用いた場合の可能性を評価するための予備的実験を行なった．

値予測に用いた方式が Stride 方式をベースにした単純なものであるにも関わらず，データ予測がデータ依存関係の破壊による投機実行失敗の発生回数の削減，または同期待ち時間の短縮に非常に有効であることが確認できた．

本稿の結果は，スレッドレベル並列処理にデータ

予測を使用した場合の可能性を探るための予備的なものであり，実現性やコストについては考慮されていない．また，評価に用いたプログラムも少なく充分とは言えない．

しかし，それにもかかわらず，ここで得られた結果は今後の開発において少なからぬ意義を持つものであり，今後はそれらの課題もふまえてより具体的な検討を進めてゆく予定である．

謝辞

本研究の一部は，文部省科学研究費補助金 (10480062, 11480069, 11878052, 12780218) の補助による．

参考文献

[1] H. Hirata, K. Kimura, S. Nagamine, Y. Mochizuki, A. Nishimura, Y. Nakase and T. Nishizawa: "An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads," Proceedings of the 19th Annual International Symposium on Computer Architecture, pp. 136-145, (1992).

[2] M.H. Lipasti, C.B. Wilkerson, and J.P. Shen, "Value Locality and Load Value Prediction," Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operationg Systems, pp. 138-147, (1996).

[3] F. Gabbay and A. Mendelson, "Speculative Execution Based on Value Prediction," Technical Report, Technion, (1997).

[4] A. Moshovos and G. S. Sohi: "Streamlining inter-operation memory communication via data dependence prediction," Proceedings of the 30th Annual International Symposium on Microarchitecture, pp. 235-245, (1997).

[5] L. Hammond, M. Willey and K. Olukotun: "Data Speculation Support for a Chip Multiprocessor," Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 58-69, (1998).

[6] P. Marcuello, J. Tubella and A. Gonzlez: "Value Prediction for Speculative Multithreaded Architectures," Proceedings of the 32nd Annual International Symposium on Microarchitecture, pp. 230-236, (1999).