計算機アーキテクチャ 149－4
（２００２．８．２２）

VLIW プロセッサのためのトレースキャッシュの提案

島尻 寛之　　吉田 たけお
琉球大学 工学部 情報工学科

あらまし： VLIW プロセッサは，非数値計算プログラムでは多くの NOP 命令を実行してしまうという問題点がある．本稿では，この問題点を解決するために，VLIW プロセッサ上で複数パス投機実行手法を実現するためのトレースキャッシュを提案する．提案手法では，トレースキャッシュを用いることによって，分岐する 2 つのパスを同時にフェッチする．フェッチした 2 つのパスに対して，一方のパスに含まれる NOP 命令の代わりに，他方のパスの有効な命令を実行することによって 2 つのパスを同時に実行し，複数パス投機実行を実現する．SPEC95 ベンチマークに対して本手法の評価を行った結果，遅延サイクル数が 1 の場合，遅延分岐方式を適用したときに比べて IPC を約 15 ～ 30% 向上できることがわかった．
キーワード： VLIW プロセッサ，トレースキャッシュ，複数パス投機実行，分岐予測

A Trace Cache for VLIW Processors

Hiroyuki SHIMAJIRI and Takeo YOSHIDA
Department of Information Engineering, Faculty of Engineering,
University of the Ryukyus

Abstract: VLIW processors have a potential to achieve good performance in numerical applications. However, it is difficult that VLIW processors achieve high performance in non-numerical applications. We propose a trace cache for VLIW processors. Also, we propose a multi-path speculative execution mechanism using the proposed trace cache in order to achieve high performance in non-numerical applications. During speculative execution, our mechanism replace NOP instructions in a execution path by effective instructions in a different path. The VLIW processor executes a merged path that proposed mechanism made. This paper shows performance evaluation and analysis of proposed method. The software simulation results on the SPECint 95 benchmarks show that our method improves a performance by 15 ～ 30% in a VLIW processor with a branch delay of 1 cycle.
KEYWORDS: VLIW Processor, Trace Cache, Multi-path Speculative Execution, Branch Prediction

1　はじめに

VLIW プロセッサは，科学技術計算プログラムのように並列性の高いプログラムでは，非常に高い性能を実現することができる．しかし VLIW プロセッサでは，動的なスケジューリングを行わないため，非数値計算プログラムのように並列性の低いプログラムでは，多くの NOP 命令を実行してしまう問題点がある [1,2].

一般に，非数値計算プログラムの実行効率は，投機実行手法によって向上できることが知られている [3]．しかし，VLIW プロセッサ上では，投機実行を行うパスにも多くの NOP 命令が含まれているため，単一のパスだけの投機実行を行っても問題の解決にはならない．この投機実行時に実行するパスに含まれる NOP 命令の代わりに，異なるパスの有効な命令を実行できれば，冗長な NOP 命令を削減でき，さらに複数のパスを同時に投機実行を行うことができると考えられる．

上記の複数パス投機実行手法を実現するためには，複数のパスの VLIW 命令を同時にフェッチする必要がある．これまでに，効率的な命令フェッチを実現する機構としてトレースキャッシュが提案されている [4, 5]．トレースキャッシュは，アドレスが非連続な命令列 (トレース) を同一のキャッシュブロック内に格納することにより，効率的な命令フェッチを実現する．このトレースキャッシュを VLIW プロセッサに応用すれば，複数の VLIW 命令を効率良くフェッチできると考えられる．

そこで本稿では，複数パス投機実行手法を VLIW プロセッサ上で実現するために，VLIW プロセッサ向けのトレースキャッシュを提案する．本稿で提案するトレースキャッシュは，分岐命令の分岐先のパスのトレースのみを保持する．投機実行を行う際には，トレースキャッシュから得られる分岐先のパスと命令キャッシュから得られる分岐命令の後続のパスに対して，両方の VLIW 命令に含まれる NOP 命令を利用し，1 つの VLIW 命令にまとめる．この 1 つにまとめた VLIW 命令をプロセッサに供給することによって，複数パス投機実行手法を実現することができる．

以降，**2** 章ではトレースキャッシュを用いた複数パス投機実行手法について説明する．**3** 章では，提案手法を実現するためのトレースキャッシュ機構と複数パス投機実行機構の構成を示す．続く，4 章で性能評価を行い提案手法の有効性を示す．

2 トレースキャッシュを用いた複数パス投機実行手法

2.1 複数パス投機実行手法

ここでは，VLIW プロセッサ上で複数パス投機実行手法を実現するためのトレースキャッシュの概要について述べる．なお以下では，分岐命令が不成立の場合に実行されるパスを後続パス，成立の場合に実行されるパスを分岐先パスと呼ぶ．

まず，VLIW プロセッサ上で実現する複数パス投機実行手法について述べる．本手法では，

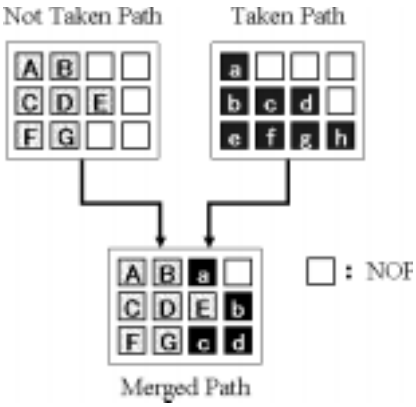

図 1: VLIW 命令の合成

フェッチした VLIW 命令を監視し，分岐命令を含む VLIW 命令を検出する．分岐命令を含む VLIW 命令をフェッチした次のサイクルから，分岐命令の分岐先が確定するまで投機実行を行う．投機実行中は，後続パスと分岐先パスの VLIW 命令を同時にフェッチし，図 1 に示すように，両方のパスに含まれる NOP 命令を利用して 2 つの VLIW 命令を 1 つの VLIW 命令にまとめる．このように，2 つのパスの VLIW 命令を 1 つの VLIW 命令にまとめる処理を合成と呼ぶことにする．

本手法では VLIW 命令の合成の際に分岐予測を行い，優先的に合成するパス (優先パス) を決める．優先パスの VLIW 命令は，1 サイクルで全ての命令が合成される．一方，非優先パスの VLIW 命令は，優先パスの VLIW 命令内に NOP 命令が含まれている場合に合成される．合成しきれなかった非優先パスの命令は，次サイクルの優先パスの VLIW 命令と合成される．図 1 の例では後続パスを優先パスとしている．図 1 において，後続パスの 2 番目の VLIW 命令には NOP 命令が 1 つしか含まれていないため，分岐先パスの 2 番目の VLIW 命令が 2 サイクルにまたがって合成されている．分岐命令の分岐先が確定した後は，プログラムの実行結果に矛盾が生じるのを防ぐために，分岐 (実行) しなかった方のパスの実行結果を無効化する．

なお，合成しきれなかった非優先パスの VLIW

命令は，複数のサイクルにまたがって実行されることになる．そのため，バラバラに発行されたVLIW命令の内部命令間で逆依存が発生する恐れがある．本手法では，この問題を回避するために，非優先パスの実行結果を実行ステージ内で一時的に保持し，非優先パスのVLIW命令の内部命令が全て実行されてから，実行結果を後続のステージに出力する．これにより，バラバラに発行されたVLIW命令の内部命令間の依存関係を保証する．

2.2 トレースキャッシュの動作

2.1で述べた複数パス投機実行手法では，後続パスと分岐先パスのVLIW命令を同時にフェッチする必要がある．後続パスのVLIW命令をフェッチする場合，プログラムカウンタ(PC)の値をインクリメントすることによって，命令キャッシュからフェッチすることができる．一方，分岐先パスのVLIW命令をフェッチする場合，まず分岐命令の分岐先アドレスを計算しなければならない．そこで本稿では，トレースキャッシュを用いて分岐先パスのトレースを保持する．これにより，後続パスと分岐先パスのVLIW命令を，それぞれ命令キャッシュとトレースキャッシュから同時にフェッチすることができる．

本稿で提案するトレースキャッシュは，以下の2つの理由により，分岐先パスのトレースのみを格納するものとする．1つ目の理由は，VLIW命令はビット長が長く，後続パスと分岐先パスの両方のトレースを格納する場合，非常に多くのキャッシュ容量が必要となるためである．また2つ目の理由は，後続パスと分岐先パスを合成したトレースを格納した場合，優先パスを変更する度に分岐先パスのトレースを改めてフェッチしなければならないためである．

トレースキャッシュに格納するトレースは，分岐命令が成立したとき(分岐先パスが実行されたとき)に生成される．投機実行の際に，トレースキャッシュ内に該当するトレースが存在しない場合は，その分岐命令は過去に成立したことが無いと仮定して後続パスのみを投機実行する．これにより，一度も成立していない分岐命令に関してはトレースを生成しないため，トレースキャッシュを効率的に使用することができる．

トレースキャッシュは，分岐先パスが実行されたときに命令キャッシュから出力される分岐先パスのVLIW命令を保持することによってトレースを生成する．トレースの生成中に分岐命令が現れた場合は，そのときに実行した方のパスのVLIW命令をトレースとして保持する．生成したトレースを格納する際に，トレースキャッシュ内に同じ分岐命令から生成されたトレースが存在した場合には，古いトレースを新しいトレースで上書きする．

3 提案手法のハードウェア構成

ここでは提案手法を実現するための機構について説明する．図2に提案手法の構成例を示す．図2に示すように提案手法を実現する機構は，トレースキャッシュ機構と複数パス投機実行手法を実現する機構に分けられる．

3.1 トレースキャッシュ機構

本稿で提案するトレースキャッシュ機構は，トレースを生成するフィルユニットとトレースキャッシュ本体で構成する．

1. フィルユニット (Fill Unit)

フィルユニットは，トレースキャッシュに格納するトレースを生成するためのユニットである．フィルユニットは，分岐命令が成立した後に実行されるVLIW命令を，内部のバッファに格納することによってトレースを生成する．バッファにVLIW命令が貯まった時点で，バッファ内のVLIW命令(トレース)をトレースキャッシュに出力する．

2. トレースキャッシュ

トレースキャッシュは，フィルユニットで生成されたトレースを格納し，投機実行時にトレースと分岐予測の結果を出力する．トレースキャッシュ内には，トレースの先頭アドレスやトレースの後続命令のアドレスの他，分岐予測に用いる分岐予測カウンタを格納している．投機実行

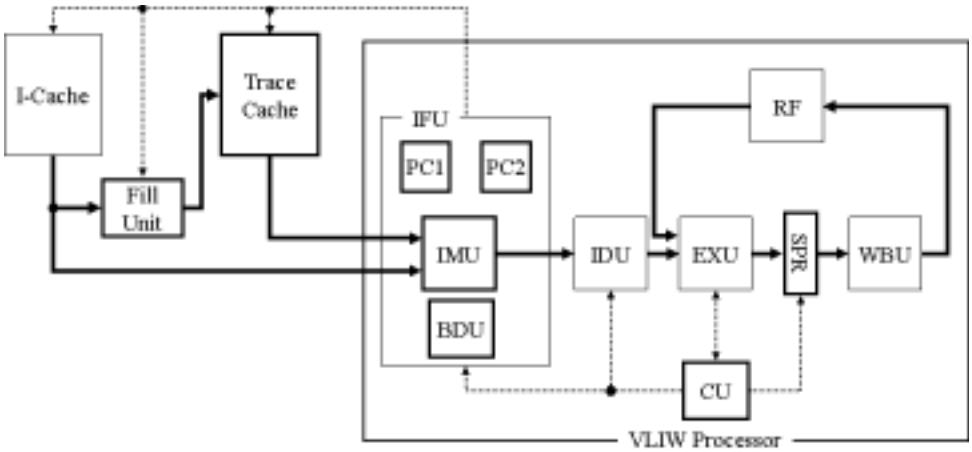

図 2: 提案トレースキャッシュの構成

の際には，トレースキャッシュは分岐予測カウンタの値をもとに分岐先を予測し，後述する命令合成ユニットに予測結果を出力する．

3.2 複数パス投機実行機構

複数パス投機実行を実現する機構は，命令フェッチユニット内に実装した分岐命令検出ユニットと，命令を合成する命令合成ユニット，分岐しなかったパスを無効化する無効化ユニット，投機実行中の VLIW 命令の実行結果を保証するための投機実行レジスタなどで構成する．

1．分岐命令検出ユニット（BDU）

分岐命令検出ユニットは，フェッチしてきた VLIW 命令を常に監視し分岐命令を検出する．また分岐命令が成立した場合，フィルユニットに対して分岐命令が成立したことを知らせる制御信号を出力する．この他，命令フェッチユニット内には，投機実行中に後続パスと分岐先パスのアドレスを保持するためにプログラムカウンタ (PC) を 2 つ実装する．

2．命令合成ユニット（IMU）

命令合成ユニット (IMU) は，投機実行中，トレースキャッシュから出力された分岐予測の結果をもとに，後続パスと分岐先パスの VLIW 命令を合成する．また IMU は，非優先パスの VLIW 命令を合成しきれなかった場合には，非優先パスの残りの命令を保持する．VLIW 命令を全て合成した場合には，発行が完了したことを知らせるための発行完了フラグを合成した VLIW 命令に付加する．なお，投機実行時以外は，命令キャッシュからフェッチした VLIW 命令をそのまま出力する．

3．無効化ユニット（CU）

無効化ユニットは，分岐命令の分岐先が確定した後に，分岐しなかった方のパスの命令を保持しているパイプラインレジスタとアドレスを保持している PC を無効化する．

4．投機実行レジスタ（SPR）

投機実行レジスタ (SPR) は，実行ステージの後段に配置し，投機実行時にバラバラに発行された非優先パスの VLIW 命令の実行結果を保持する．SPR は，バラバラに発行された非優先パスの VLIW 命令が全て実行されてから，実行結果を後続のステージに出力する．バラバラに発行されたかどうかは，IMU において付加された発行完了フラグの情報をもとに判断する．

以上に示したトレースキャッシュ機構と複数パス投機実行機構を VLIW プロセッサ内に実装することにより，トレースキャッシュを用いた複数パス投機実行手法を実現する．

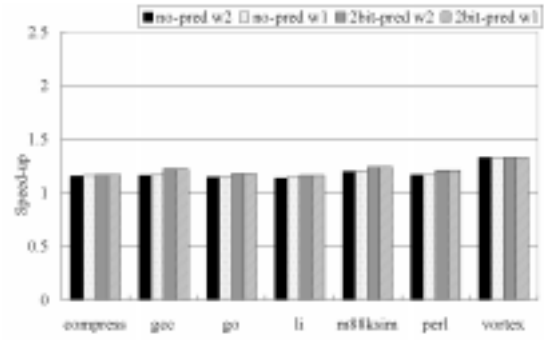

図 3: 各プログラムの IPC(遅延サイクル数 1)

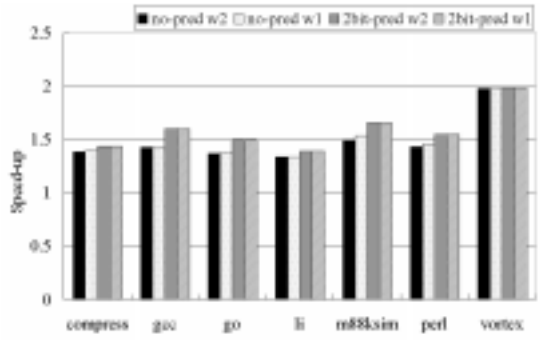

図 5: 各プログラムの IPC(遅延サイクル数 3)

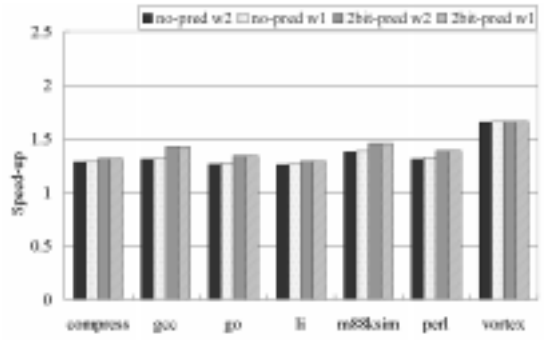

図 4: 各プログラムの IPC(遅延サイクル数 2)

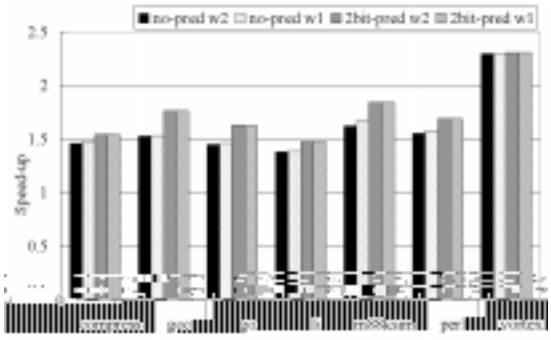

図 6: 各プログラムの IPC(遅延サイクル数 4)

4 性能評価

今回，SPEC95 ベンチマークのいくつかのプログラムに対して，提案手法の評価を行った．評価に用いた VLIW プロセッサは，MIPS 社の R3000^{TM}をベースにしており，128 ビットの VLIW 命令に最大 4 つの命令を指定することができる．ただし，1 つの VLIW 命令には分岐命令は 1 つまでしか指定できないものとした．

評価は，提案手法と遅延分岐方式について，投機実行のサイクル数 (遅延サイクル数) を 1 ～ 4 サイクルとしたときの IPC を評価した．なお，遅延サイクル数は，提案手法で保持するトレースの長さと発生するミスペナルティの最大値となっている．提案手法の評価では，64 エントリを保持できるダイレクトマッピング方式と 2Way セットアソシアティブ方式の 2 種類のトレースキャッシュに対して，投機実行時に 2 履歴分岐予測手法に基づいて優先パスを選択した場合と，分岐予測を行わずに常に後続パスを優先パスとした場合について評価した．ただし，2 履歴分岐予測手法を行う場合は，トレースキャッシュ内に，エントリ毎に 2 ビットの分岐予測カウンタが用意されているものとする．

評価はソフトウェアシミュレーションによって行い，各ベンチマークプログラムは GNU の GCC 2.7.2.3 を用いてコンパイルし，その結果をリストスケジューリング手法で再スケジューリングしたものを使用した．提案手法の効果を確認するため，命令キャッシュにはキャッシュミスは発生しないものと仮定した．

図 3 ～ 6 に，それぞれ遅延サイクル数が 1 ～ 4 段の場合の各プログラムの IPC を示す．各グラフの値は，遅延分岐方式の IPC を 1 とした場合の割合となっている．

図 3 ～ 6 から遅延分岐方式に比べて，提案手

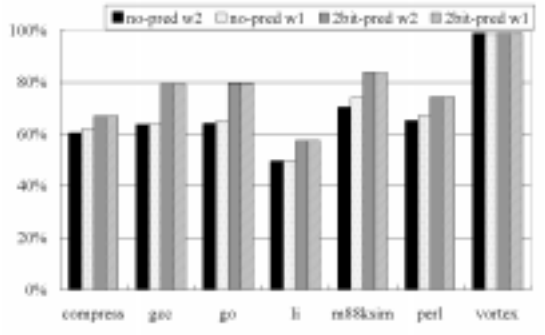

図 7: ペナルティ削減率

法を適用した場合，遅延サイクル数 1 では 15% ～ 35% ほど IPC が向上している．IPC の向上率は，遅延サイクル数が増加するにしたがって高くなっており，最大で約 2.3 倍に向上している．ただし，実際の IPC の値は，遅延サイクル数が少ない方が高くなっている．

続いて，投機実行時に分岐予測を行った場合と行わなかった場合で比較すると，全てプログラムにおいて分岐予測を行った場合の方が IPC の向上率が高くなっていることがわかる．両者の差は，遅延サイクル数が増加するにしたがって大きくなっている．

トレースキャッシュの構成の違いで比較してみると，分岐予測を行わない場合では，ダイレクトマッピング方式の方がやや IPC が高くなっている．一方，分岐予測を行った場合では，ほとんど違いは見られなかった．

続いて，提案手法を適用した場合のペナルティ削減率を求めた．ペナルティ削減率を図 7 に示す．なお，ここでのペナルティ削減率は，遅延サイクル数が 1 ～ 4 サイクルまでの平均値を表している． 図 7 より，分岐予測を行った場合，gcc, go, m88ksim, vortex 等で 80% 前後のペナルティ削減率を実現している．分岐予測を行わない場合には，vortex を除き，最大で約 70%のペナルティ削減率しか達成できなかった．

以上のことから，提案手法では簡単な分岐予測機構でも性能向上を実現できることがわかる．

5 おわりに

今回，VLIW プロセッサ上での複数パス投機実行手法を実現するために，VLIW プロセッサ向けのトレースキャッシュを提案した．また，提案手法を実現するための，トレースキャッシュ機構と複数パス投機実行機構をそれぞれ示した．提案手法の評価を行った結果，提案手法が VLIW プロセッサの性能向上に有効であることを示した．

提案手法では，トレースキャッシュ内の分岐先パスと後続パスの 2 つのパスだけを投機実行の対象としていた．今後の課題として，2 つ以上のパスを対象に提案手法を適用することについて検討していく予定である．また，実際に提案するトレースキャッシュを設計し，トレースキャッシュの回路面積やクリティカルパスなどを調査する予定である．

参考文献

[1] 中澤喜三郎，“計算機アーキテクチャと構成方式，” 朝倉書店, 1995.

[2] 富田眞治，“第 2 版 コンピュータアーキテクチャ，” 丸善, 2000.

[3] 阿部孝之，仲池卓也，小林広明，中村維男，“VLIW アーキテクチャのためのダイナミックブースティング機構，” 信学論 (D-I), Vol. J83-D-I, No. 1, pp. 171–183, 2000.

[4] E. Rotenberg, S. Bennett, and J. E. Smith, “Trace cache: a low latency approach to high bandwidth instruction fetching,”Proc. 29th Annual International Symposium on Microarchitecture, pp. 24–34, 1996.

[5] Q, Jacobson and J. E. Smith, “Trace Preconstruction,”Proc. 27th Annual International Symposium on Computer Architecture, pp. 37–46, 2000.

[6] “SimpleScalar Simulation Tools for Microprocessor and System Evaluation,” URL:http://www.simplescalar.org/.

[7] Gerry Kane 著，前川守 監訳，“mips RISC アーキテクチャ —R3000/R2000—,” 共立出版，1992.