スーパスカラ・アーキテクチャのための複数パス実行機構の提案

安島 雄一郎, 中村 友洋, 吉瀬 謙二, 辻 秀典, 田中 英彦
東京大学大学院 工学系研究科

スーパスカラ・アーキテクチャ向けの複数パス投機的実行機構を提案する。深いパイプラインを持ち複数の命令を同時に発行するスーパスカラ・プロセッサでは、分岐予測の失敗により大きなペナルティを受ける。しかし分岐先となる複数のパスを実行することにより、分岐予測ミスのペナルティを大幅に削減することができる。シミュレーションによる評価の結果、1 サイクルあたりの命令実行数が 3 〜 10% 程度向上することを確認した。

Speculative Multi-Path Execution Mechanism for Superscalar Architecture

Yuichiro Ajima, Tomohiro Nakamura, Kenji Kise, Hidenori Tsuji, Hidehiko Tanaka
University of Tokyo, Graduate School of Engineering

We propose a speculative multi-path execution mechanism for superscalar architecture. Branch miss-predictions cause serious penalties for a superscalar processor with deep pipeline and multiple instruction issue system. Speculative multi-path execution reduces the branch miss penalty and improve processor performance. Our evaluation results show that our mechanism improves processor performance by 3% to 10%.

1 はじめに

深いパイプラインを持ち、1 サイクルに複数の命令を発行する近年のスーパスカラプロセッサでは、分岐における命令供給の停止は大きな性能低下を招く。多くのスーパスカラ・プロセッサでは、分岐先の予測により投機的実行を行ない命令供給を停止しないようにしている。しかし、分岐予測が誤りであった場合、その分岐命令がフェッチされてから誤りが判明するまでの間にフェッチされた命令は全て無駄となる。分岐予測の正解率は 85% 〜 95% に達するものもあるが、それでも分岐予測ミスによるペナルティによってプロセッサは 10% 〜 20% 程度性能を低下させている。このため、分岐予測ミスに伴うペナルティそのものを削減する必要がある。

分岐命令において双方の分岐先のパスを投機的に実行し、正しい分岐先の結果のみを採用すれば分岐命令で命令供給停止によるペナルティは生じない。しかし、全ての分岐命令で双方のパスを実行するには複雑で大規模なハードウェアが必要になるため、マイクロプロセッサでは複数パスの実行は行なわれてこなかった。本論文では、スーパースカラプロセッサに拡張を行なうことによって複数パスを投機的に実行する機能を追加する手法を示し、分岐予測ミスのペナルティを削減できることを確認する。

以下、第2章ではスーパスカラ・アーキテクチャについて説明し、第3章では複数パス実行について述べる。第4章では具体的なアーキテクチャの提案を行ない、第5章でシミュレーションによって性能を評価する。最後に第6章では今後の課題、展望について述べ、第7章でまとめを行なう。

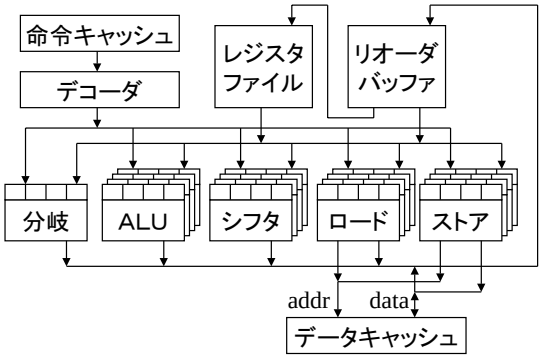

図 1: スーパスカラ・モデルのブロック図

2 スーパスカラ

2.1 スーパスカラの命令実行

スーパスカラ・アーキテクチャのモデルをブロック図で図 1に示し、スーパスカラでの命令実行の様子を概説する。命令はまず命令キャッシュからフェッチされ、デコーダで解釈される。その後、レジスタファイルまたはリオーダバッファから読み込まれたオペランドとともに、各実行ユニットに備えられたリザベーション・ステーションに送られる。リザベーション・ステーションで実行条件が揃うのを待ち合わせ実行条件が揃った命令は各実行ユニットで実行される。最後に演算結果をリオーダバッファまたはデータキャッシュに送って命令実行完了となる。

一般的なプロセッサではパイプライン構成によるスループットの向上が行なわれている。このモデルではステージ 1で命令フェッチ、ステージ 2でデコード、ステージ 3ではリザベーション・ステーションで実行条件が揃うのを待ち合わせ、ステージ 4,5 で実行、完了となる。

2.2 out-of-order 実行

スーパスカラプロセッサは、命令の順序に基づく実行順序の制限を緩和し、演算結果に影響がなければ逆の順序や同時に命令を実行できる out-of-order 実行機構を持っている。

単純に命令の順序を逆転して実行する場合では、先に実行した命令の演算結果を直ちにレジスタに書き込んでしまうと、後から実行する命令に誤ったオペランドを供給してしまう場合がある。これを防ぐため、スーパスカラ・プロセッサでは演算結果をすぐにレジスタファイルに書き込まずに一旦リオーダバッファに書き込む。リオーダバッファは元の命令順序に従って演算結果が揃うまで演算結果を保持し、命令順にレジスタファイルに書き込む。デコーダがオペランドとしてレジスタファイルに書き込む前の演算結果を要求した場合、リオーダバッファはレジスタファイルの代わりにオペランドの供給を行なう。

また、他の命令の演算結果を必要とするデータ依存関係が存在する場合、デコードを止めるとプロセッサの性能が大きく下がる。このため、リオーダバッファは演算結果がまだ書き込まれていない場合、デコードされた命令に演算結果が格納される予定のリオーダバッファ・エントリの番号をオペランドの代わりに渡す。その後必要な演算結果が出力され次第、リザベーション・ステーションで待機している命令にオペランドを供給する。

以上に説明したリオーダバッファとリザベーション・ステーションの動作により、スーパスカラ・プロセッサでは効率的な out-of-order 実行を実現している。

2.3 投機的実行と例外回復処理

前節で述べたように、スーパスカラ・プロセッサでは out-of-order 実行でプロセッサ性能を向上させている。しかし、out-of-order 実行の効率を上げるにはデータ依存関係の解消した命令が多数必要になる。このため、プロセッサは将来実行すべき命令を予め多数デコードしておく必要がある。しかし、プログラム実行中に出現する命令の約 2 割は分岐命令であることが知られている。このためプロセッサが将来実行すべき命令をフェッチしようとした場合、頻繁に含まれる分岐命令によって次にフェッチする命令が決まらなくなる。そこで、分岐命令の分岐先を予測し、投機的に命令フェッチ、実行を進める。これを投機的実行と呼ぶ。

分岐予測の成功率は、分岐先が 2 種類以下の分岐命令で 90% 以上である。この分岐命令は分岐命令実行の大部分を占める。レジスタ間接分岐を合わせた全体の分岐予測では 80% 〜 90% 程度である。ここで分岐予測が失敗であった場合、予測を行なった分岐命令以降

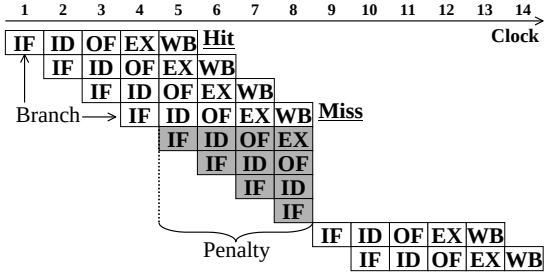

図 2: 単一パス実行でのパイプライン動作

にフェッチした命令を無効化し、正しい実行状態を回復しなければならない。スーパスカラ・プロセッサでは演算結果をリオーダバッファに格納しているため、この回復処理はリオーダバッファのエントリを削除するだけで良い。また、実際のプロセッサでは外部から見た動作をスカラ・プロセッサと同じにするため、例外処理のためにプログラムの任意の命令で実行を中断できることが要求される。このため、out-of-order 実行も一種の投機的実行と言える。

以上に述べた、特定の命令まで実行した状態を回復する処理を例外回復処理と呼び、プロセッサに要求される機能の一つである。

2.4 分岐予測失敗によるペナルティ

図 2 に投機的実行を行なうプロセッサのパイプライン動作を示す。横方向はクロックの進行を表し、縦方向はプロセッサのフェッチする命令の進行を表している。各命令はクロックの進行にともなってパイプラインステージが進み、WB ステージを経て完了する。ここでは簡単のため、各ステージで実行が停止することはないものとする。以下、クロック n にフェッチされた命令を命令 n と表現する。図 2では、命令 1 と命令 4 は分岐命令である。ここではフェッチ時に分岐予測によって投機的に分岐先を決定し、次のサイクルでは予測した分岐先の命令をフェッチする。命令 1 の分岐方向はクロック 5 で決定し、予測は正解であった。この場合、後続の命令すなわち命令 2,3,4 は有効であり、毎サイクル命令が完了する。一方、命令 4 の分岐命令ではクロック 8 で分岐予測が失敗となった。分岐予測失敗の場合、後続の命令すなわち命令 5 〜 8 は無効となり、クロック 9 から改めて正しい分岐先の命令がフェッチされる。命令 5 〜 8 が無効となるため、クロック 9 〜 12 では有効な命令が完了しない。この例では分岐予測失敗によって 4 命令分のペナルティが生じていることが分かる。

ここで分岐予測失敗のペナルティがプロセッサの性能に与える影響を試算する。分岐予測失敗は 10 回に 1 度の頻度でランダムに起こるとし、分岐命令は全実行命令の約 1/5 を占めるとする。これらより、分岐予測失敗は

およそ 50 命令に 1 度の割合で発生することが分かる。一方分岐予測失敗によるペナルティは、典型的なパイプラインを持つプロセッサでは 5 サイクル程度であり、スーパスカラ・プロセッサでは 1 サイクルあたりの命令実行数 (IPC) は平均 1 〜 2 である。よって、ペナルティは平均 5 〜 10 命令に相当する。以上より、プロセッサは分岐予測失敗によっておよそ 50 命令に 1 度、5 〜 10 命令相当のペナルティを受けていることになる。これは 10 〜 20% の性能低下である。今後アーキテクチャの進歩により IPC が向上すればペナルティに相当する命令数が大きくなり、全実行時間に占めるペナルティの割合はさらに増大する。このため、分岐予測失敗による性能低下を削減することは重要な課題となっている。

以上の試算方法から、この性能低下を押えるには 3 つの取り組み方があることが分かる。まず分岐予測の失敗率を下げること、次に分岐命令の頻度を小さくすること、そしてペナルティを少なくすることである。このうち分岐予測成功率は既に 90% 前後に達しているため、大幅に改善することは難しい。また、分岐命令の頻度もプログラムの構造上、大幅に減らすことはできない。そこで、分岐予測失敗によるペナルティ自体の削減が必要となる。次章からは分岐予測失敗のペナルティを削減する方法として複数パス実行について説明する。

3 複数パス実行

まず、用語の定義を行なう。以下では従来の投機的実行を単に投機的実行または単一パス実行と呼び、複数パスの投機的実行を複数パス実行と呼ぶ。

分岐予測失敗によるペナルティは、正しい分岐先が実行されていないことに起因する。複数パス実行は分岐先を一つだけ選ぶのではなく、分岐する可能性のある複数の分岐先を実行するものである。よって複数パス実行で分岐予測失敗によるペナルティが削減できることは明らかである。しかし分岐先にも分岐命令は存在するため、無制限に実行するパスを増やすと必要な計算資源が爆発的に増大する。以下では複数パス実行のモデルと制御アルゴリズムを述べる。

3.1 複数パス実行のモデル

複数パスの実行モデルには、まず単純に全ての分岐先を実行するものがある。これを Eager Execution (以下 EE) モデルと呼ぶ。

図 3に、EE モデルで投機的に実行する制御の流れをグラフで示す。グラフの枝は、実行中または実行結果保持中で、プロセッサ内に留まっている命令を表している。制御順序の性質から、流れは木構造に表現される。矢印は分岐パスを表している。分岐パスとは、

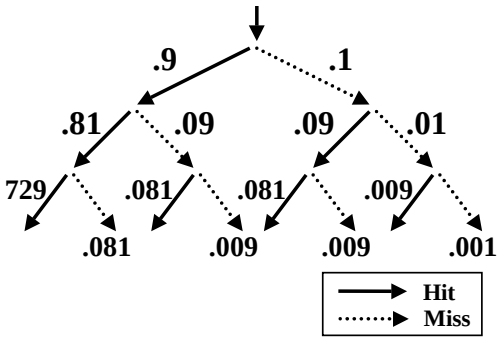

図 3: EE モデルにおける制御の流れ

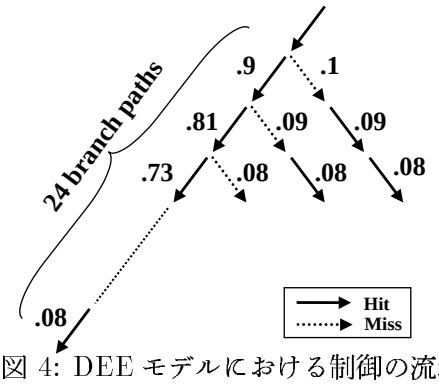

図 4: DEE モデルにおける制御の流れ

分岐命令によって区切られた単一の制御流のことである。実線矢印は、直前の分岐命令での分岐予測が成功 (Hit) となる方向の分岐パス、破線矢印は分岐予測が失敗 (Miss) となる方向の分岐パスである。矢印の側に記した数字は、そのパスに制御が移る統計的な確率を示している。ここでは理解を容易にするため、左側 90%、右側 10% に固定してある。すなわち、図 3において分岐予測は常に左側を予測するものとする。

EE では投機的実行が成功である確率が極めて低い分岐パスを他の分岐パスと区別せず投機的に実行する。また全ての分岐先を実行するため、投機段数が増えるとプロセッサ内部に保持する分岐パス数が爆発する。このため、得られる性能に対して必要なハードウェア量が非常に大きいことが予想できる。

これを改善するため、パスに制御が移る確率の大きいものから計算資源を割り当てる Disjoint Eager Execution (以下 DEE) モデルが Uht, Sindagi によって提案されている [4]。DEE による制御の流れを図 4に示す。木構造や矢印の意味は図 3と同じである。分岐予測の成功率は 85% 〜 95% 程度であるため、分岐予測が失敗となる側の制御流は、成功となる側に比べ制御が移る確率が非常に小さくなる。このため、分岐予測が連続して成功する制御流に多量の計算資源を割り当て、分岐予測失敗を含む制御流には少量だけ計算資源を割り当てる。図 4の例では、2 回分岐

予測が失敗する制御流には計算資源を割り当てていない。

3.2 制御アルゴリズム

ここで DEE モデルによる実行を複数の制御流の組合せで近似する。まず、in-order ステートから全ての分岐パスが分岐予測成功の方向である長い制御流を主制御流と定義する。次に、主制御流に含まれる分岐命令から始まる制御流で、先頭以外の分岐パスが全て分岐予測成功である短い制御流を副制御流(群)と定義する。DEE モデルでは in-order ステートから制御流の先頭までの間に 2 回分岐予測失敗の方向を含むことは少ないため、主制御流と副制御流群の組合せで十分に良く DEE モデルが近似できる。各制御流の中に含む分岐に注目すると、主制御流、副制御流とも分岐予測成功の方向のみ選択している。ここでさらに全ての分岐命令で分岐予測成功の方向に進む確率が一定であると仮定すれば、主制御流と副制御流の長さを分岐を越える数(以下、分岐投機数)で静的に決定することができる。

これにより、分岐方向の確率を求めなくても、分岐予測と分岐投機数だけで DEE モデルによる制御を近似的に実現できる。図 4の例で、それぞれの制御流の分岐投機数を数える。これは図 4での分岐パスの節の数に等しいので、主制御流の分岐投機数は 24、副制御流の分岐投機数は in-order ステートに近いものから順に 2, 1, 0 となる。このように、統計的計算に基づいて制御流毎の分岐投機数を静的に決定し、制御を行なうアルゴリズムを static tree heuristic という [4]。

また実際の計算機では、IBM370/168 が副制御流を 1 つ、IBM3033 は副制御流を 2 つ持っている [1]。これらも近似的に DEE を行なっていると言える。

なお、以降では全体の制御流の数を制御数、副制御流の数を副制御数と呼ぶ。図 4の例では制御数 4、副制御数 3 となる。

3.3 スーパスカラへの適用

本節では、スーパスカラ・アーキテクチャで複数パス実行を行なう場合、効率の良い制御アルゴリズムについて定量的に考察する。

ある分岐パスを投機的に実行することで期待できる性能への寄与は、分岐パスが正しい実行の制御に含まれていた場合にプロセッサの性能を向上させる割合と、その分岐パスを実行する確率との積で求まる。

まず、分岐パスを実行する確率について考察する。現在の分岐予測方法は、一つの分岐命令の分岐方向に偏りが存在する性質を利用したものが主流である。しかし、複数パス実行制御に使用する分岐予測方法では連続して失敗しないことが求められる。単一パス実行では 1 回の予測失敗はペナルティに直結するため、初回の予測成功・失敗のみが意味を持つ。しかし、複数パス実行では確率が低いと予測されたパスの実行も行なうため、予測失敗の方向で出現する分岐命令についても分岐方向の性質を考慮する必要がある。分岐方向の統計的な性質に着目すると、単一の分岐命令には分岐予測が連続して当たる傾向があることが知られている。よって、コード実行中に続いて現れる分岐命令の間にも分岐予測成功・不成功の相関があることが予想される。ここで、分岐予測が失敗した分岐命令の次に実行された分岐命令について、分岐予測の成功率を図 5に示す。なお、シミュレーションは表 1に示す対象について行なった。

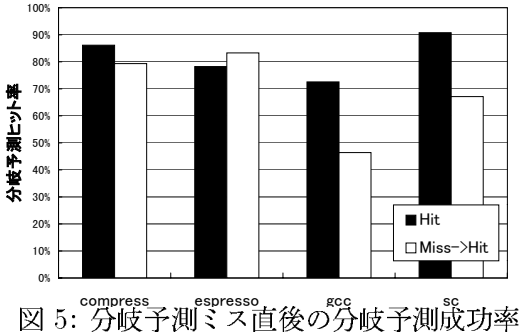

図 5: 分岐予測ミス直後の分岐予測成功率

黒で示した「Hit」は、通常の分岐予測成功率すなわち全ての分岐命令の分岐予測成功率を示している。白で示した「Miss → Hit」は、直前の分岐命令における分岐予測が失敗であった分岐命令についての分岐予測成功率を示している。compress, espresso では、通常の分岐予測成功率と失敗後の分岐予測成功率はほぼ同じであった。一方、cc1, sc では失敗後は分岐予測性効率が大幅に下がり、特に cc1 では成功率が 50% を下回っている。

これらより、プログラムによって違いはあるが、複数パス実行で分岐予測失敗となる方向のパスを実行する場合、分岐方向の性質が変わる場合があることが分かる。

3.4 規模の小さい複数パス実行

リオーダバッファはオペランドを供給するため、規模を大きくすることは難しい。また

表 1: シミュレート対象

項目	
CPU	SPARC Version8
OS	SunOS4.1.4
使用プログラム	compress,espresso,cc1,sc
実行命令数	10,000,000 命令
分岐履歴	512entry
	full associative

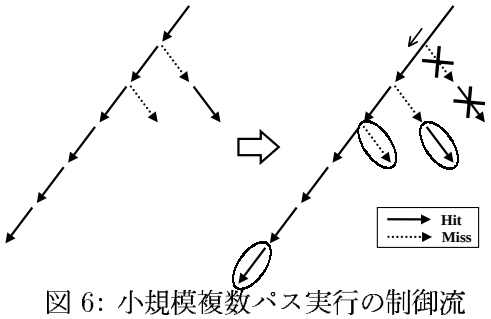

図 6: 小規模複数パス実行の制御流

分岐予測の確率を考慮すると、スーパスカラに static tree heuristic による複数パス実行を導入した場合、副制御流は短く、本数も少ないものが適していることが予想される。本節では、このような規模の小さな複数パス実行での命令フェッチの振舞いについて述べる。

小規模な複数パス実行を行なっている場合にプロセッサ内部に保持される命令の制御流について、定常的な状態を示す (図 6左)。矢印の意味は図 3と同じである。この例では、主制御流の分岐投機数を 6、副制御流の分岐投機数は 2 としている。分岐予測に関する性質が全ての分岐命令で同じと仮定すると、図 6のように副制御流の長さをもっとも in-order ステートに近い部分で長くなるように制御する場合の効率が最良である。分岐予測成功の方向に分岐が確定した場合、命令フェッチは図 6に示すように進行する。確定した分岐命令からの副制御流の実行結果は無効であるため削除される。in-order ステートに 2 番目に近い分岐命令から新たな副制御流を起こし、主制御流、各副制御流の先頭で命令フェッチを行なう。

4 スーパスカラ向け複数パス実行機構の提案

本章ではリオーダバッファを多重化することにより、スーパスカラ・プロセッサに複数パス実行機構を導入する手法を提案する。

4.1 アーキテクチャ

図 7に提案するアーキテクチャのモデルを示す。従来のスーパスカラに追加する機構はリオーダバッファの多重化、リザベーション・ステーションへのエントリ削除機能の追加、ストアバッファの多重化である。主制御流と副制御流はほとんどの資源を共有しており基本的な構成に変更は必要ない。

リオーダバッファは各制御流に属する命令の演算結果を格納できるように拡張する。リザベーション・ステーションは、無効になった分岐パスに含まれる命令のエントリを削除できるように拡張される。また、ストアバッファも無効になった分岐パスのストアを削除できるよう、制御流に対応して多重化される。次節以降、各追加機構の詳細な動作と例外処理へ移行する手順について述べる。

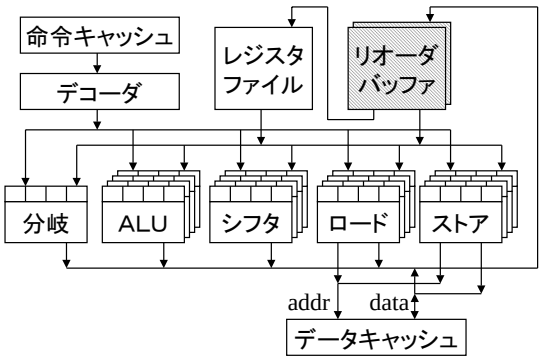

図 7: 複数パス実行の可能なスーパースカラ・アーキテクチャの構造

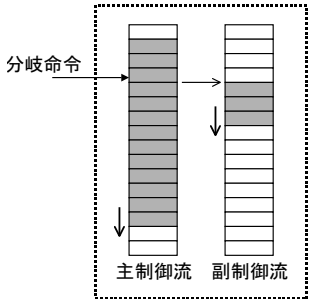

図 8: 多重化したリオーダバッファ

4.2 リオーダバッファの多重化

リオーダバッファは命令の演算結果をコードの順序を保ったエントリに格納する。計算結果は古いものから順にレジスタファイルに送られるため、レジスタファイルには確定した状態 (in-order ステート) が格納され、リオーダバッファには最新の状態 (アーキテクチャ・ステート) が格納される。このリオーダバッファにより、スーパスカラ・プロセッサは out-of-order の実行において、任意の箇所で停止して例外処理を行なうことができる。

提案する機構ではリオーダバッファを制御流の数だけ用意し、制御流ごとに対応したリオーダバッファを使用する (図 8)。分岐予測失敗が確定した場合主制御流が移動するため、全てのリオーダバッファは主制御流としても副制御流としても使用でき、なおかつ切替に複数サイクル必要としないものが必要である。現実的な実装方法としては、ハードウェア的に複数のリオーダバッファを用意する方法、論理的にのみ独立のエントリ空間を用いハードウェア的には共有のエントリ空間を利用し、ハードウェア的にエントリ空間の射像を行なうことでエントリを行なう方法がある。副制御流では使用するエントリ数が少ないため、後者の方法はハードウェア量の点から有利である。

また、制御流が無効となった場合、その制御流に含まれる命令が各実行ユニットのリザ

ベーション・ステーションのエントリに残っている場合がある。このため、無効化されたリオーダバッファはリザベーション・ステーションに制御流が無効になったことを通知する。これによって無効になることが判明した命令を実行することによる性能低下を避ける。また、ストアバッファに対しても同様の通知を行なう。

4.3 各ユニットの機能追加

4.3.1 リザベーション・ステーション

リザベーション・ステーションは各実行ユニットに設けられており、実行すべき命令とそのオペランドまたはオペランドのタグ、そして演算結果を書き込むリオーダバッファのタグをエントリに格納する。エントリに格納された命令はオペランドが揃い、演算資源に空きができたら実行され、結果はリオーダバッファに送られる。

複数パス実行では、分岐命令の結果によって特定の制御流だけが無効になる。この時、無効になった制御流に含まれる命令だけをリザベーション・ステーションのエントリから削除する必要がある。提案するアーキテクチャのリザベーション・ステーションでは、リオーダバッファから無効化の通知を受けとると、まず削除すべき命令を検出する。これは無効化を通知したリオーダバッファに演算結果を送る命令である。検出された命令はリザベーション・ステーションのエントリから速やかに削除される。

4.3.2 ストアユニット

キャッシュに格納されるデータを in-order ステートに保つため、ストア命令はストアユニット内のストアバッファに保持される。保持されたストア命令は、in-order 実行が保証されるまでメモリへの出力を行なわない。リオーダバッファがオペランド供給を行なうのと同様に、ストアバッファもストア命令を追い越したロード命令に対してデータの出力を行なう。このストアユニットも制御流に対応して多重化し、制御流別にストア命令の保持、追い越しを行なったロード命令へのデータ供給を行なう。

4.3.3 例外処理

リオーダバッファを拡張して用いるため、任意の命令で処理を中断することができる。具体的には、中断する命令以降のエントリをリオーダバッファから削除し、命令フェッチを停止する。中断する命令までの処理が完了するまで待ち、完了後リオーダバッファ無効化の通知を各ユニットに行なった後に例外処理へ移行する。

4.4 動作と各ユニットへの負荷の考察

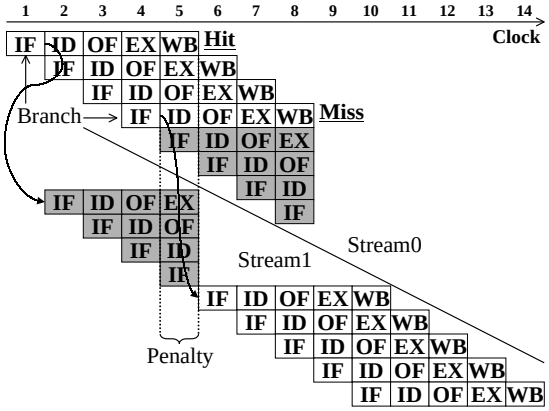

図 9: 複数パス実行でのパイプライン動作

ここで、複数パス実行を導入したプロセッサがペナルティを削減する様子を、図 2と同様のパイプライン動作のモデルで図 9に示す。この例では、複数パス実行の副制御数は 1、副制御流で実行できる分岐パスは 1 に制限されている。命令制御流が同時に 2 つ存在するため、以下では Steam0 でクロック n にフェッチされた命令を命令 n(0)、Stream1 でクロック n にフェッチされた命令を命令 n(1) と表す。図 2と同様、命令 1(0) は分岐予測が成功する分岐命令、命令 4(0) は分岐予測が失敗する分岐命令である。

図 9では、クロック 2 で分岐命令の分岐先が両方フェッチされる。命令 2(0) は分岐予測で示された分岐先の命令で、命令 2(1) はもう一方の分岐先の命令である。命令 1(0) は分岐予測成功であるから Stream0 の動作は図 2の例と同様で、ペナルティは生じない。Stream1 では命令 1(0) の分岐方向が判明するクロック 5 で、無効になった命令 2 〜 5(1) が削除される。一方、命令 4(0) では分岐予測が失敗し、主制御流は Stream1 に移る。ここで Stream0 において命令 5 〜 8(0) が無効になるのは図 2と同様である。しかし Stream1 では、分岐命令 1(0) からの分岐パスが無効であることが分かったクロック 5 の次のクロック 6 から、分岐命令 4(0) からの分岐パスの実行をはじめている。このため、有効な命令 6 〜 8(1) がクロック 10 〜 12 で完了する。この場合、ペナルティは Stream0 の命令 4(0) が完了してから Stream1 の命令 6(1) が完了するまでの 1 サイクルのみに減ったことが分かる。

ここで、図 9を例として、複数パス実行が各ユニットに与える負荷を考察する。ここでは、命令 4(0) の分岐先が複数実行されているクロック 6 〜 8 に注目する。まずインストラクション・フェッチに注目すると、フェッチされたのは命令 6 〜 8(0),6 〜 8(1) の 6 命令で、単一パス実行の 3 命令の 2 倍となっている。次にデコードに注目すると、単一パス実行の命令 5 〜 7(0) に加え、命令 6 〜 7(1) が増加して約 1.7 倍の負荷増であ

る。オペランドフェッチを行なった、すなわち ID ステージから OF へ移った命令は命令 4 〜 6(0),6(1) であり単一パス実行の約 1.3 倍となっている。実行、完了に関しては単一パス実行と変わらないことが分かる。これより、副制御流が短い場合副制御流は次々に無効化されて新たに起こされるため、実行、完了などの深いパイプラインステージにあるユニットへの負荷は比較的高くならないことが分かる。この例では 3 命令おきに分岐命令が置かれたため各ユニットへの負荷のかかり方の違いが極端に現れたが、実際の分岐命令の頻度は平均 5 命令に 1 つであるので副制御流の分岐投機数 0 の場合でも深いパイプラインステージのユニットに多少負荷がかかる。

5 評価

提案した機構による性能向上を評価するため、スーパスカラ・プロセッサのシミュレータに複数パス実行機能を実装し、プログラム実行のシミュレーションを行なった。

5.1 評価環境

シミュレータの設定概要を表 2 に示す。5 ステージのパイプラインを備え、OF ステージから WB ステージまでは out-of-order に実行される。リオーダバッファは制御流毎に 32 エントリで、命令は制御流毎に 4 命令同時デコードされる。分岐履歴エントリは 512 個用意し、分岐方向の状態数は 2bit である。命令メモリのアクセス時間は 1 サイクル固定とし、データメモリにはキャッシュとアクセス時間を設定した。アクセス時間の設定値は、表の想定環境による。シミュート対象は表 1 に示した通りで、シミュレータは、スカラ・プロセッサのシミュレータで出力したトレースデータをベースに駆動される。

実装した複数パス実行のアルゴリズムには static tree heuristic を用いている。このアルゴリズムでは、もっとも in-order ステートに近い副制御流では、副制御流の分岐投機数は副制御数 -1 になる、すなわち副制御数に等しい分岐パスを実行するように制御される。1000 万命令の実行サイクル数から IPC を計算し、主制御流の分岐投機数と副制御流の数を変えて性能の変化を評価した。

表 2: シミュレーションパラメタ

プロセッサモデル	
パイプライン	IF/ID/OF/EX/WB
	OF 〜 WB: out-of-order
命令レイテンシ	各ステージ 1 サイクル
リオーダバッファ	32 エントリ
スーパスカラ度	4 命令同時デコード
分岐履歴	512 エントリ
	full associative
ストアバッファ	8 エントリ
命令メモリ	
アクセス時間	1 サイクル
データメモリ	
1 次キャッシュ	16KB,256bit/line
	4way,LRU,2 ポート
2 次キャッシュ	256K,256bit/line
	4way,LRU
	3 サイクル / アクセス
メインメモリ	25 サイクル / アクセス
(想定環境)	
CPU サイクル	333MHz
外部バスサイクル	66MHz
2 次キャッシュ	オンチップ 9ns 未満
メインメモリ	5-1-1-1 アクセス

5.2 結果

図 10 に compress での IPC の変化を示す。横軸に主制御流の分岐投機数、縦軸 IPC を取り、副制御数ごとに折れ線グラフで示してある。副制御数 0 は単一パス実行に等しい。同様に、図 11 は espresso、図 12 は gcc、図 13 は sc での結果である。

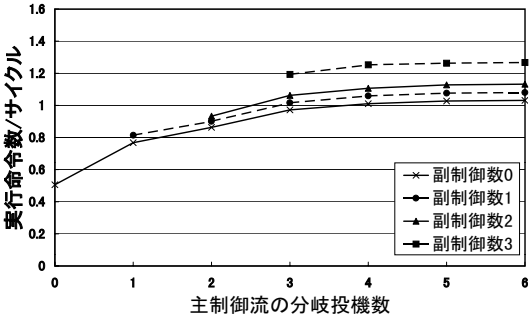

図 10: 結果: compress

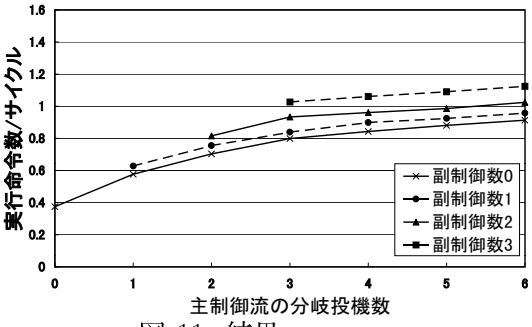

図 11: 結果: espresso

5.3 考察

図 10〜 13 より、どの設定でも副制御数 1 では単一パス実行より IPC で 0.03 〜 0.1、割

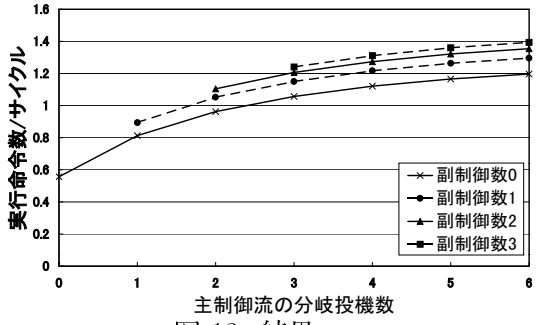

図 12: 結果: gcc

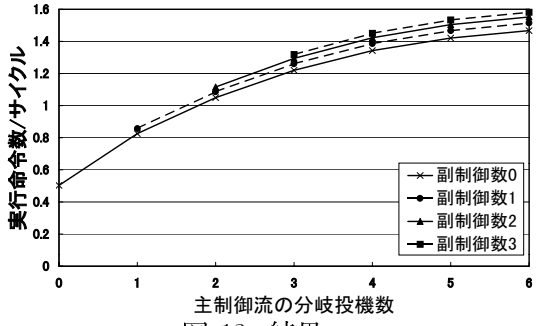

図 13: 結果: sc

合で 3% ～ 10% 程度性能向上していることが確認できる。同様に副制御数 2 では 10% ～ 15%、副制御数 3 では 15% ～ 25% 程度性能が向上している。図 5で分岐ミスの直後の分岐予測成功率が低かった gcc, sc では、副制御数 2,3 での性能向上が低くなっていることが分かる。

制御数の増加とともに副制御流での分岐投機数が増えるため、実装に必要なハードウェア量は増大する。副制御流の分岐投機数が上がると、深いパイプラインステージのユニットへの負荷も制御数倍に近付く。

副制御数 1 の実装では比較的小さなハードウェア量で 3% ～ 10% の性能向上を実現しているのに対し、副制御数 2,3 では非常に大きなハードウェア量を用いて 10% ～ 25% 程度の性能向上である。これは、一般的には投入したハードウェアに対して十分な性能向上を得ているとは言えない。

6 今後の課題、展望

評価ではフェッチャ、デコーダ、各実行ユニットへの負荷は考慮しなかったが、実際のハードウェアでは特定のユニットに負荷が集中する場面が考えられる。これに対して、主制御流と副制御流の資源利用の間に優先度を設けるなどの対処を検討する必要がある。また副制御数 1 では副制御流の命令は完了する割合が小さくなると考えられるため、各ユニットに対する負荷のかかり方が従来と変わると予測される。今後はこれらの効果、すなわち各ユニットへの負荷を定量的に評価する必要がある。

リオーダバッファはオペランド供給を集中的に行なうため、大規模化が難しい。しかし、文献 [4] は複数パスの投機的実行を大規模化することによってより高い命令レベルの並列性が得られることを示しており、投機実行の大規模化が必要とされている。このため、大規模な複数パスの投機的実行で例外回復を実現するための新たな例外回復機構の実現が課題となっている。

また、図 5 で示したように、現在提案されている分岐予測は複数パスの実行を前提にしたものではない。副制御数 1、すなわち副制御流では分岐投機を行なわない実装においては必要ではないが、一般的には複数パス実行での性能を意識した分岐予測が求められる。複数パス実行を意識した分岐予測方法も今後の課題として挙げられる。

7 まとめ

本研究では、統計的性質よりスーパスカラ・プロセッサには static tree heuristic による副制御流の分岐投機数の小さい複数パス実行方式が適していると考察した。また、複数パス実行をスーパスカラ・アーキテクチャの上に実現する方法の提案を行なった。さらに、提案したアーキテクチャの性能に関しスーパスカラ・プロセッサのシミュレータを用いて評価を行なった。結論として、副制御数 1 の複数パス実行を行なうことで 3 ～ 10% の性能向上を得られることが分かった。

参考文献

[1] Johnny K. F. Lee and Alan Jay Smith: "Branch Prediction Strategies and Branch Target Buffer Design", *IEEE Computer* Vol.17, pp.6-22 (1984)

[2] James E. Smith: "Implementation of Precise Interrupts in Pipelined Processors.", *Proc. 12th Annual International Symposium on Computer Architecture*, pp.36-44 (1985)

[3] Mike Johnson: "*Superscalar Microprocessor Design*", Prentice-Hall (1991)

[4] Augustus K. Uht and Vijay Sindagi: "Disjoint Eager Execution: An Optimal Form of Speculative Execution", *MICRO-28*, pp.313-325 (1995)