例外回復可能な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

複数パスを投機的に実行することにより、より高いIPC(Instructions Per Cycle)を得られることが指摘されている。しかし従来のマイクロ・プロセッサはハードウェア量の制限から単一パス実行を前提として設計されており、複数パス実行のために新しいアーキテクチャが必要となっている。本稿では既存の例外回復機構を複数パスの投機的実行に適用した場合に生じる問題について議論し、これを解決する新しい例外回復・オペランド供給機構のモデルを提案する。

Exception recovery mechanism for multi-path execution

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

It is pointed out in several papers that it is possible to gain higher IPC(Instructions Per Cycle) by speculative multi-path execution. However, because past micro-processors were designed for single-path execution, new architecture for speculative multi-path execution is required. In this paper, we discuss problems of applying existing exception recovery mechanism to multi-path execution, and propose a new exception recovery and operand supply model to solve the problems.

1 はじめに

大規模な投機的実行を行なう場合、得られる IPC は本質的に数十程度あるとされている[4][5][6]。一方、現実のプロセッサの IPC は2程度である。これは、種々の実装上の制限によって性能が抑えられているためである。しかしデバイス技術の進歩により、将来は大規模な投機的実行などの高度な、より高い IPC を得ることができる設計が必要になると予想される。本稿では大規模な投機的実行を行なうシステムに必要な例外回復機構について考察し、新しい例外回復機構の提案を行なう。

2 複数パス実行

プロセッサが分岐命令をフェッチした場合、その後に取り得る実行パスは複数存在する。演算によって分岐先が確定する前に実行することを投機的実行と呼ぶ。ここで、実行パスを1つだけ選ぶものを単一パス実行(または単一パスの投機的実行)、複数のパスを実行するものを複数パス実行(または複数パスの投機的実行)と呼ぶ。

単一パスの投機的実行では、正しい分岐を予測している限り分岐先の確定を待たずに実行を進めることができる。分岐予測に失敗した場合、誤った実行パスで行なった演算結果を破棄してプロセッサ

の状態を分岐命令時点まで巻き戻した後、本来の実行パスに実行を移す必要がある。この巻き戻しは out-of-order 実行における例外回復と同じ作業であり、共通の機構を用いることができる。このため、近年のプロセッサでは例外回復機構を利用した単一パスの投機的実行が行なわれている。しかし、分岐予測ミスによる性能低下が 10% ～ 20% 程度存在しており [7]、今後プロセッサの性能が向上すればこの割合はさらに大きくなる。分岐は現在のプロセッサ性能を制限する大きな要因の一つであり、このため近年のプロセッサでは単一パスで効率的に投機的実行できる規模に演算ロジックを制限し、代わりにキャッシュメモリを大きくしている。

一方、複数パス実行では分岐の影響をさらに小さいものにできる。分岐する確率が低いと予測される実行パスについても、分岐先が確定する前にある程度実行を進めることができるため、分岐予測が不正確となる場合でもプロセッサの性能に与える影響を小さくできる。また、多くの分岐命令を跨ぐ場合でも、複数パスを実行することにより正しい実行パスを投機的に実行できる確率を上げることができる。このため、大規模に投機的実行を行なうことによって、より多くの命令から実行可能な命令を検出することが可能になる。ここで図 1に命令ウィンドウのサイズを増やすことによるプロセッサの性能向上の可能性を示す [6]。これより、大規模な投機的実行を行ない、広い範囲から実行可能な命令を検出することによってより高い ILP を得られる可能性があることが分かる。

3 例外回復機構

3.1 実行ステート

out-of-order 実行を行なうプロセッサで例外回復を実現するには、プロセッサは 2 つの実行ステートを保持する必要がある。1 つは例外回復のためのインオーダ・ステートである。インオーダ・ステートは連続した完了命令だけによって作られたプロセッサの状態である。このステートは実行が確定した状態であるので、これ以前の状態を保存する必要はない。もう 1 つはオペランド供給を行なうアーキテクチャ・ステートである。このステートは命令の演算結果や、演算結果の格納先として予約されているエントリのタグを含んだ最新のステートである。

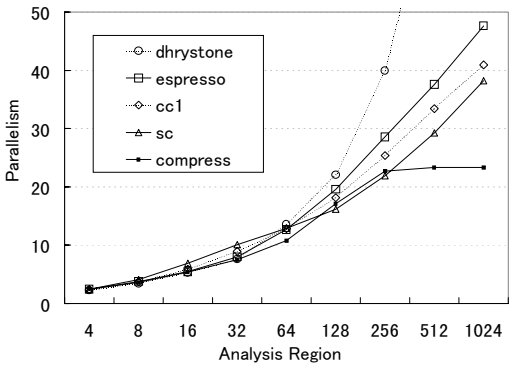

図 1: 命令ウィンドウサイズと性能向上

以上のステートによってインオーダ・ステートからのやり直しによる例外回復が可能であるが、リオーダバッファを用いた例外回復では先見ステートも導入される。これはインオーダ・ステート以降の全ての命令の演算結果、及び未完了の命令の場合はその演算結果が格納されるエントリのタグによるステートである。命令単位で演算結果を保持するため、同じレジスタへの代入が起きる場合でも古い値が保存されている。これにより、命令単位での例外回復を高速に行なうことができる。

3.2 既存の例外回復機構

3.2.1 チェックポイント回復

チェックポイント回復 [2] のモデルを図 2に示す。チェックポイント回復では分岐命令または任意の命令でアーキテクチャ・ステート (図中 current) のバックアップを取る。新しくバックアップがとられる度に古いバックアップは push され、最も古いバックアップはインオーダ・ステートになっている。図では、太い矢印はバックアップ全体のコピー操作、細い矢印はデータの流れを表している。例外回復は current をインオーダ・

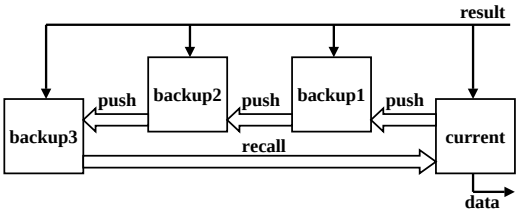

図 2: チェックポイント回復

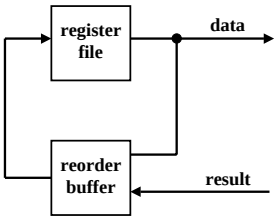

図 3: リオーダバッファ

ステートに戻した後、該当命令まで実行をやり直すことで行なう。バックアップがインオーダ・ステートに移行するために、バックアップが取られたあとも未完了命令の結果が戻ってくればバックアップ内容の更新を行なう。このため、全てのバックアップは命令の実行結果を受けとる必要がある。また、チェックポイント回復では 2 番目に古いバックアップがインオーダ・ステートになるまで最も古いバックアップを保持する必要がある。このため、バックアップ数が上限に達した場合 current のデコードを停止する。

3.2.2 リオーダバッファ

図 3に示すリオーダバッファによる例外回復 [1] では先見ステートをリオーダバッファ (FIFO) に、インオーダ・ステートをレジスタファイルに保持する。連想ハードウェアを用い、双方から命令オペランド供給を行なうことでデコーダ側からはアーキテクチャ・ステートが見える。リオーダバッファは先見ステートを保持しているため、該当命令以降のエントリを削除することで例外回復を行なう。やり直しを行なう必要がないため、命令単位での高速な回復動作が可能である。リオーダバッファではエントリが足りなくなるとデコードを止める。

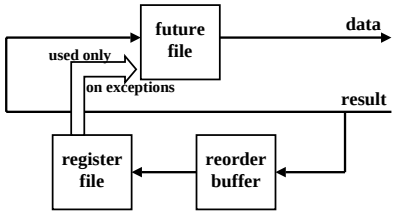

図 4: フューチャファイル

3.2.3 フューチャファイル

リオーダバッファによるオペランド供給には連想検索が必要である。これを解消するため、フューチャファイル [1] が追加する方法がある (図 4)。フューチャファイルはアーキテクチャ・ステートを保持し、オペランド供給の一切を行なう。レジスタファイル、リオーダバッファは例外回復にのみ使用する。例外回復ではまずインオーダ・ステートを該当命令まで進め、その後レジスタファイルの内容をフューチャファイルへコピーする。

3.2.4 部分フューチャファイル

フューチャファイルでは回復動作でレジスタファイルのコピーを必要とする。これを不要とするため、フューチャファイルに完全なアーキテクチャ・ステートを保持せず、レジスタファイルからもオペランド供給を行なう改良 (図 5) が提案されている [3]。この場合フューチャファイルからはインオーダ・ステートに含まれない演算結果や保留中のタグをオペランドとして供給し、レジスタファイルからはインオーダ・ステートからアーキテクチャ・ステートの間に更新されていないレジスタ値をオペランドとして供給する。この改良により、レジスタファイルのコピーと連想ハードウェアのどちらも不要になり、有効な出力の選択のみが必要となる。以降では改良されたフューチャファイルを部分フューチャファイルと呼ぶ。

3.3 大規模投機的実行への適用

本節では既存の例外回復機構を複数パスに適用し、さらに大規模に投機的実行する場合にどのような問題が生じるか議

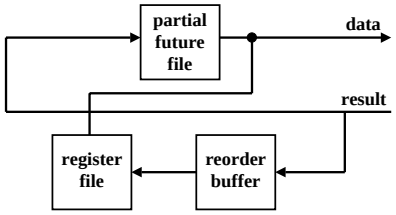

図 5: 部分フューチャファイル

論する。

3.3.1 チェックポイント回復

図 6にチェックポイント回復においてチェックポイントを実行パスに対応して多重化することにより、複数パス実行での例外回復を行なうモデルを示す。オペランドの供給は実行パス毎のアーキテクチャ・ステートから行なわる。演算結果は全てのバックアップに送られ、各バックアップは必要な結果を用いて内容を更新する。分岐命令が現れた場合、アーキテクチャ・ステートのコピーを取り実行パスを分ける。

この方式の欠点はステートのコピーが多いことである。特に、正しい実行パスである確率が低いと予測される実行パスにもアーキテクチャ・ステートのコピーを必要とする点は大きな欠点と言える。また、演算結果によって多数のバックアップ、アーキテクチャ・ステートを更新する必要がある。さらに、複数パスを大規模に実行した場合、1 サイクルに返される演算結果は単一パス実行に比べて多くなる。このため、演算結果のブロードキャスト、バックアップの更新に必要なハードウェア量が非常に大きくなることが予想される。

3.3.2 リオーダバッファ

まず、リオーダバッファの大規模化の効果について考察する。図 7に、分岐投機数による IPC の変化を示す。横軸は分岐パス数、縦軸は IPC である。ここで分岐パスとは実行中の命令列上で分岐命令によって区切られた 1 続きの命令列を指す。この図での分岐パス数は分岐投機数 +1 になる。実行は仮想的なプロセッサでシミュレートし、リオーダ・バッファ 32 エントリ、1 サイクル 8 命

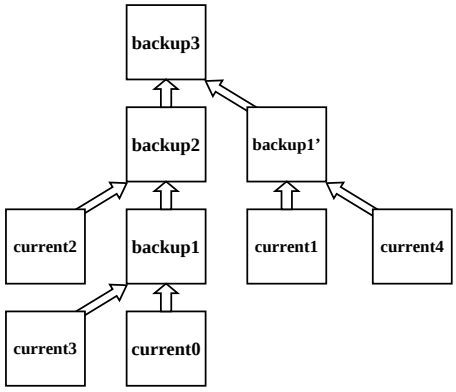

図 6: チェックポイント多重化

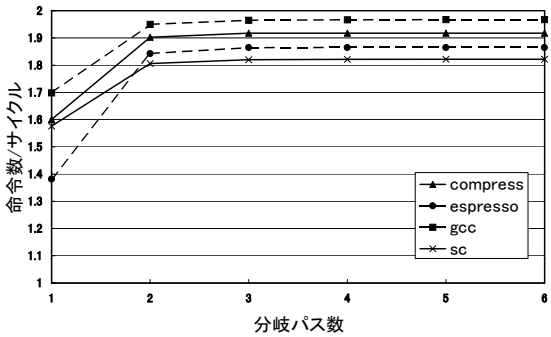

図 7: リオーダバッファ使用下の IPC

令デコード、全ての命令を 1 サイクルで実行、同時命令発行数無制限、分岐予測正解率 100% と設定した。使用したコードは SPARC V8、OS は SunOS4.1.4 で、SPEC92 に含まれる 4 つのプログラム compress,espresso,cc1,sc について 10,000,000 命令実行のシミュレーションを行った。なお、分岐命令は平均 5.15 命令に 1 つの頻度で現れた。

次に、例外回復にリオーダバッファ以外の機構を用いたと仮定し、アーキテクチャ・ステート、即ちレジスタの最新値と保留中のタグのみが制限されるとする。エントリ数は図 7と同じく 32 とする。この場合のシミュレーション結果を図 8に示す。

図 7では性能が飽和しているのに対し、図 8では分岐パス数を上げることによって性能が向上していることが確認できる。リオーダバッファによって命令の解析範囲が命令長で制限されるため、分岐パス数 (= 分岐投機数 +1) を増やして

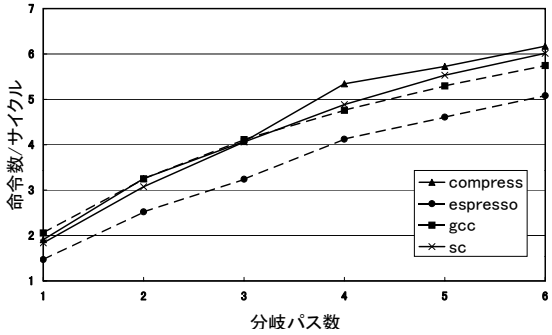

図 8: 命令解析範囲を制限しない場合

もある程度で性能は飽和する。また分岐命令の頻度から、リオーダバッファでは平均命令ランが用意されたエントリの半分程度に達すると性能が飽和しておりエントリ利用効率が悪く、ハードウェア規模に対して性能が得られにくいことが予想される。これより、リオーダバッファのように命令解析範囲を限定する機構は大規模な投機的実行においては性能を制限する要因となると考えられる。

リオーダバッファのみによる例外回復では連想ハードウェアが、部分フューチャファイルを用いた例外回復ではオペランドの選択が必要になる。これらのハードウェアはリオーダバッファの規模が大きくなるにつれて高速化が難しくなる点も、投機的実行の大規模化による性能向上を妨げる要因となる。また、リオーダバッファを用いた例外回復機構では演算結果がリオーダバッファに集中するため、IPC が向上した場合にリオーダバッファそのものの構成が複雑化し、プロセッサの高速化を妨げる要因となると考えられる。

また、単一パス実行では分岐命令における頻繁な回復動作が必要であるが、複数パス実行機構によってこれを不要にすることが可能である。よって、リオーダバッファの大きな利点である命令単位での高速な回復動作は必ずしも必要ではない。

4　新しい機構の提案

前章で議論した投機的実行大規模化の際に既存の例外回復機構に生じる短所を踏まえ、これらを克服する新しい例外回復・オペランド供給機構を提案する。

提案する機構のモデルを図 9に示す。ここで実線矢印はデータの流れを、破線矢印は命令実行順序の関係を表す。この機構ではオペランド供給は複数の部分フューチャファイルから行なわれる。図の例では、部分フューチャファイル current の実行パス上の命令には、current 及び部分フューチャファイル 1〜4、レジスタファイルからオペランドが供給される。また、部分フューチャファイル current' の実行パス上の命令には、current' 及び部分フューチャファイル 2〜4、レジスタファイルからオペランドが供給される。

部分フューチャファイルは分岐命令が現われる毎に、その先の分岐パスに新たに割り当てられる。命令の演算結果はその分岐パスに割り当てられた部分フューチャファイルのみを更新する。

また、インオーダ・ステートはレジスタファイルといくつかの部分フューチャファイルで保持する。インオーダ・ステートに達した部分フューチャファイルは、その内容を出力してレジスタファイルに更新する。レジスタファイルを更新し終った部分フューチャファイルは捨てられる。部分フューチャファイルは分岐パス単位でレジスタファイル更新値を持つため、リオーダバッファのようにインオーダ・ステートを更新するために命令毎の演算結果を保存する必要はない。

例外回復は該当分岐パスまで実行を進めた後、その分岐パスの実行をやり直すことで実現される。分岐は新たな部分フューチャファイルの割当てで処理されるためやり直しが生じるのは割り込み処理や例外処理の場合のみである。このためやり直しによる回復が必要となる頻度は低く、やり直しのオーバーヘッドが性能に与える影響は小さいと考えられる。

このモデルによって例外回復機構を構成する利点をまとめる。従来の機構の拡張ではいずれも、IPC 向上に伴って大量に発生する演算結果の処理が問題となっていた。このモデルでは、演算結果は割り当てられた部分フューチャファイルの

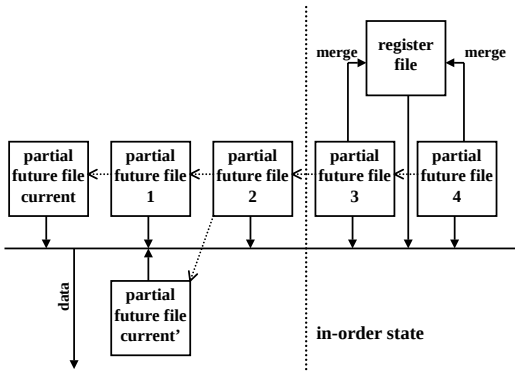

図 9: 提案する機構のモデル

みを更新するため、投機的実行の規模を上げても部分フューチャファイルそれぞれの規模を上げる必要はない。また、いくつかの機構で必要となっていたアーキテクチャ・ステートのコピーを必要としない点も挙げられる。

5 今後の課題

大規模に複数パスを実行するには本稿で提案したモデルのみでは不十分であり、未解決の問題が残されている。1つは複数パス実行導入に伴って、レジスタ及びそのタグの生存時間を制御する方法である。もう1つは、大規模に投機的実行することに伴ってフォワーディングによるオペランド供給の制御が複雑になる問題である。最後は、提案したモデルの実装に関係するが、適切なオペランドを出力する部分フューチャファイルの選択方法である。

特に、本稿で提案した機構のようにオペランド供給を複雑化した場合、オペランド供給経路及びフォワーディングの制御が非常に複雑になる。このため、従来のように演算器を共通のバスに接続する方法では設計が困難である。

VLDP プロジェクトでは多数の演算器をネットワーク的に接続したALU-Netに関する研究を行なっている [5]。ALU-Net ではフォワーディングを行なう命令の対をハードウェア的に近い演算器に割当て、直接データ転送を行なう。これにより、演算結果出力時に共通バスの制御を行なう頻度が激的に減らせることが期待できる。また、ALU-Net による演算器の分散構成は、部分フューチャファイルを細かく分ける本稿の提案と親和性が良いことが期待できる。今後の課題として、本稿で提案した機構と ALU-Net の関係について研究し、オペランド供給に関する制御の複雑さを軽減することが挙げられる。

本研究の一部は文部省科学研究費 (一般研究 (B) 課題番号 07458052「大規模データパスプロセッサの研究」) による。

参考文献

[1] James E.Smith: "Implementation of Precise Interrupts in Pipelined Processors.", *Proc.12th ISCA*, pp.36-44 (1985)

[2] Wen-mei W.Hwu and Yale N.Patt: "Checkpoint Repair for Out-of-order Execution Machines." *Proc.14th ISCA*, pp.18-26 (1987)

[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)

[5] 中村 友洋, 吉瀬 謙二, 辻 秀典, 安島 雄一郎, 田中 英彦: "大規模データパスプロセッサの構想", 情報処理学会研究報告 97-ARC-124, pp.13-18 (1997)

[6] 吉瀬 謙二, 中村 友洋, 辻 秀典, 安島 雄一郎, 田中 英彦: "多数演算器方式における演算器利用率の検討", 情報処理学会研究報告 97-ARC-125, pp.121-126 (1997)

[7] 安島 雄一郎, 中村 友洋, 吉瀬 謙二, 辻 秀典, 田中 英彦: "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", *JSPP'98* (1998 発表予定)