5N-07

複数パスの投機的実行を考慮した例外回復機構

安島 雄一郎, 中村 友洋, 吉瀬 謙二, 辻 秀典, 田中 英彦
{aji,hiro,kis,hide,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学大学院 工学系研究科 *

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]。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 例外回復機構

2.1 実行ステート

アウト・オブ・オーダ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 つはインオーダ・ステートで、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。インオーダ・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。もう一方はアーキテクチャ・ステートで、次にデコードする命令にオペランドを供給できる最新の状態である。このステートは実行が完了した命令の演算結果や、命令実行が完了してないために保留中のレジスタ更新値 (通常はタグ) を含む。

また、本質的に必要なステートではないが、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。これはデコードされた全ての連続命令列によって作られた状態で、保留中のレジスタ更新値を含む。例外処理のために命令単位で演算結果を保留するため、同じレジスタへの代入でも値の上書きが起こらない。

2.2 既存の例外回復機構

チェックポイント回復 [2] では分岐命令または任意の命令でアーキテクチャ・ステートのバックアップを取る。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い、回復はバックアップからの再実行による。この方式ではチェックポイント数の制限によりデコードを止める。

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し、どちらからも命令オペランドを供給する。2 つを連想ハードウェアで結合することにより、デコーダ側からはアーキテクチャ・ステートが見える。例外回復はリオーダ・バッファのエントリ削除により行なう。また、エントリが足りなくなるとデコードを止める。リオーダ・バッファからの演算供給は連想検索が必要になるため、これにフューチャ・ファイル [1] を追加する場合がある。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。しかし例外回復時にレジスタ・ファイルの内容をフューチャ・ファイルへコピーする必要があり、分岐予測ミスの回復には向かない。そこで、フューチャ・ファイルの拡張 [3] が提案されている。この場合フューチャ・ファイルには完全なアーキテクチャ・ステートを保持せず、レジスタ・ファイルとの結合でアーキテクチャ・ステートを実現する。結合には連想は必要ではなく、選択のみ行なえばよい。これにより例外回復時のコピーは必要でなくなる。

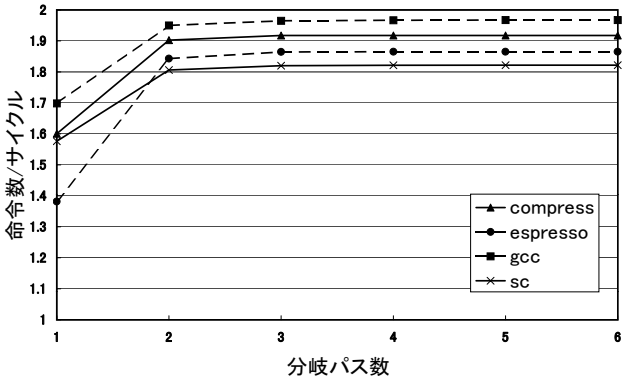

図 1: リオーダ・バッファによる制限下での IPC

3 大規模投機的実行による性能向上

文献 [4][5] などの研究により、本質的にはコードの解析から IPC を数十程度取り出し得ることが指摘されている。しかし、現実のプロセッサの IPC は 2 程度に過ぎない。これは実際のプロセッサでは各ユニットの実装により多くの面から性能が制限されるためである。ここで、投機的実行の性能に直接的な影響を与える例外回復機構、特にリオーダ・バッファについて、IPC に与える影響について解析を行なった。

図 1に、分岐投機数による IPC の変化を示す。横軸は分岐パス数 (= 分岐投機数 +1)、縦軸は IPC である。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。

図 1より、リオーダ・バッファによる例外回復を行なうシステムでは大規模な投機的実行を行っても IPC の向上は得られないことが分かる。次にリオーダ・バッファ以外の例外回復機構を用いたと仮定し、命令ウィンドウ 32 エントリ、実行が完了した命令はエントリを解放するとした場合のシミュレーション結果を図 2に示す。これより、実行が完了した命令を命令ウィンドウから解放できる機構では、大規模な投機的実行によって IPC の向上が得られる可能性があることが分かる。

実際には分岐予測ミスによるペナルティ、キャッシュ

---

* "Exception Recovery Scheme for Multi-Path Executable System", Yuichiro Ajima, Tomohiro Nakamura, Kenji Kise, Hidenori Tsuji, Hidehiko Tanaka
University of Tokyo, Graduate School of Engineering, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan

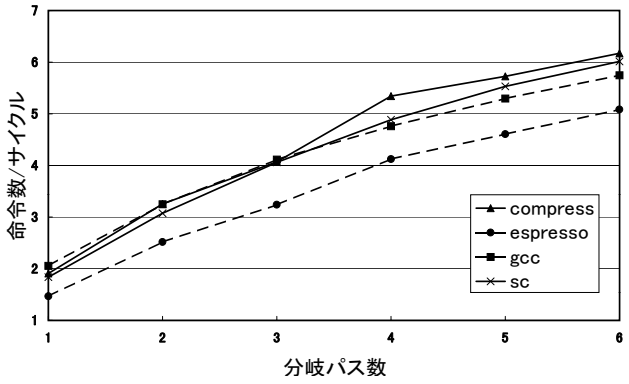

図 2: リオーダ・バッファによる制限を外した場合の IPC

ミスによる長いレイテンシなどにより、IPC は図 2ほど大きく伸びないと考えられる。しかし、リオーダ・バッファによる例外回復機構を用いる限り大規模な投機的実行によって IPC が向上することはないことが分かる。大規模な投機的実行によって IPC を向上させるにはリオーダ・バッファ以外の例外回復機構が必要であるといえる。

4 新しい例外回復機構の提案

本節では大規模な複数パスの投機的実行を行なう際に必要となる新しい例外回復機構を提案する。

図 3に基本的な動作を示す。レジスタ・ファイルはインオーダ・ステートを保持し、デコード中の各分岐パスにはフューチャ・ファイルを割り当てる。デコードが分岐命令に達するとその分岐パスのフューチャ・ファイルをチェックポイントとして保留し、分岐を投機的に越えてデコードを進める場合は分岐パスに新たなフューチャ・ファイルを割り当てる。また、一定命令数分岐が現れなかった場合は通常の命令にチェックポイントを設ける。例外回復はチェックポイントからのやり直しによって行うが、分岐は新たなフューチャ・ファイルの割当て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。このためやり直しの頻度は低く、オーバーヘッドは問題にならない。

ここで、デコード中の分岐パスのフューチャ・ファイル、チェックポイントに保留中のフューチャ・ファイル、及びレジスタ・ファイルをまとめてファイルと呼ぶ。新たにフューチャ・ファイルを割り当てる際、初期値をコピーする必要があると性能を著しく損なうことが予想される。そこでフューチャ・ファイルは文献 [3] と同様の拡張を行なう。割り当てられた直後のフューチャ・ファイルは空とし、アーキテクチャ・ステートはレジスタ・ファイル、祖先のチェックポイント、及び該当フューチャ・ファイルの結合で実現する。すなわち供給されるオペランドは、有効な値のうち最も若いファイルから出力されたものとなる。なお、演算結果は必要とするファイルに同時書き込まれる。また、保留されているフューチャ・ファイルがインオーダ・ステートに達する (すなわちレジスタ・ファイルと内容が同一になる) とその分岐パスは制御木から解放される。

このモデルでは全てのオペランド供給でレジスタ・ファイルがアクセスされることになり、レジスタ・ファイルやインオーダ・ステートに近いチェックポイントへの負荷が高い。このため、同一サイクルで全ての祖先のフューチャ・ファイルからオペランドを出力し選択を行なうのではなく、目的のオペランドを含むファイルだけ

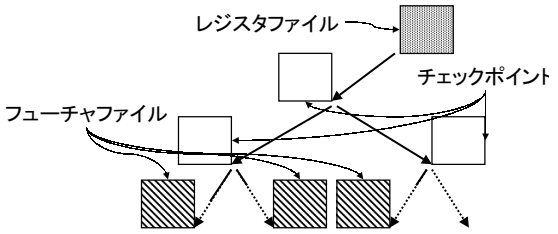

図 3: 複数パス実行での例外回復機構

にアクセスする方法が考えられる。単純な実装方法としては、若いファイルから順に調べ、有効な演算結果を保持するファイルが見つかるまで繰り返す方法がある。しかしこれではデコードが終了するまでに多くのサイクルがかかり、検索中に演算結果が返ってきた場合の処理が難しい。このため、各ファイル内に演算結果が保持されているか空であるかを集中的に管理するテーブルを用意し、1 サイクルでアクセスするファイルを特定する。ただし実際のアクセスは次のサイクルになるため、最新の演算結果を参照できない可能性がある。このためフォワーディングを必ず行ない、値が供給されないタグが発生しないようする。またこの拡張を行なった場合、インオーダ・ステートの更新のためにレジスタ・ファイルにフューチャ・ファイルをマージするハードウェアを追加する必要がある。

5 今後の課題

本稿で提案した例外回復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という副作用がある。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複雑になり過ぎ実装は困難になる。

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、直接データ転送することができる。今後の課題として、例外回復機構と ALU-Net の相互作用について研究し、さらに複数パス実行を効率的に行なうことのできる例外回復機構を開発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)