

複数バス投機実行のためのレジスタセット管理方式

安島 雄一郎, 坂井修一, 田中 英彦

東京大学大学院 工学系研究科

本論文では大規模な複数バス実行に適したレジスタセット管理方式として分散フューチャファイル・モデルを提案する。本方式ではレジスタ値出力に複数サイクルを要する場合があります、この遅延がプロセッサ性能に与える影響をシミュレーションによって評価した。結果として部分フューチャファイル数4～8程度のクラスタ化によって性能低下を抑えられることを示した。

Register Management System for multi-path execution

Yuichiro AJIMA, Shuichi SAKAI, Hidehiko TANAKA

University of Tokyo, Graduate School of Engineering

We proposed the Distributed Future File Model; a register management system for multi-path execution. By using this model in multi-path execution processors, in particular case, it may spent multi-cycle delays before the value of register become available. We evaluated the decrease of performance caused by this penalty. The results showed that by clustering partial future files by number of 4 to 8, most amount of this penalty can be reduced.

1 はじめに

複数バスの投機的実行により、分岐予測失敗によるペナルティの削減と高い命令レベル並列性の取り出しが可能であることが指摘されている [2][7]。単一バスでは1つの実行状態のみを保持することで実行が可能であった。しかし、複数バス実行では分岐命令において実行状態を複製し、それぞれ別の実行バスに割り当てて並列に演算を進める必要がある。近年、これを可能にする複数バス実行機構に関する研究が盛んに行なわれている [4][5]。本研究では、複数実行状態のレジスタセットを部分フューチャファイル [8] によって管理する、分散フューチャファイル・モデルを提案する。また、部分フューチャファイルに分割することで生じるデータ転送に関する遅延が、プロセッサ性能に与える影響を評価する。

2 分散フューチャファイル

2.1 実行ステート

out-of-order 実行を行なうプロセッサでは、例外回復機能を備えたレジスタ管理が要求される。例外回復を行なうためには、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ー・ステートである。インオーダー・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。

2.2 部分フューチャファイル

完全なアーキテクチャ・ステートを保持するレジスタファイル(フューチャファイル)を

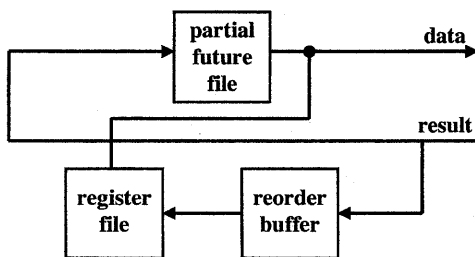


図 1: 部分フューチャファイル

用いた場合、回復動作においてインオーダー・ステートからアーキテクチャ・ステートへの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、完全なアーキテクチャ・ステートを保持するのではなく、新しい演算結果のみを保持し、インオーダー・ステートの結合でアーキテクチャ・ステートを作る方式(図1)が提案されている[1]。この場合フューチャファイルからはインオーダー・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ー・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。以降ではこの新しい演算結果のみを保持するレジスタファイルを部分フューチャファイルと呼ぶ。

3 分散フューチャファイル・モデル

複数バス実行で高い IPC を取り出すためには、広範囲な命令から実行可能命令を取り出す大規模な投機的実行が必要になる。しかし、既存の実行機構では投機的実行の大規模化は難しい。これは各制御ユニットが単一であるため命令解析範囲の拡大に伴い、オペランド出力、フォワーディング、演算器制御が複雑になるためである。これを克服するにはオペランド供給、フォワーディング、演算器制御の分散化が必要となる。本章では部分フューチャファイル(Partial Future File: PFF)を複数使用することで、複数バス実行可能でかつ大規模投機的実行に適した新しい例外回復・オペランド供給機構、分散フューチャファイル・モデルを提案する。

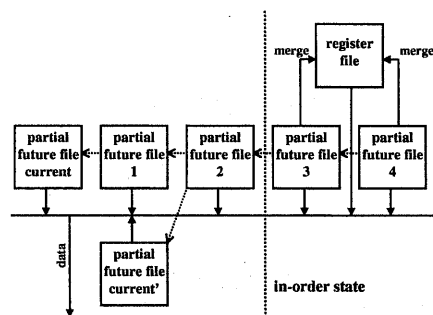


図 2: 分散フューチャファイルによるオペランド供給のモデル

分散フューチャファイルによるオペランド供給のモデルを図2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2の例では、部分フューチャファイル current の実行バス上の命令には、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' の実行バス上の命令には、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、その先の分岐バスに新たに割り当てられる。命令の演算結果はその分岐バ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、インオーダー・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。インオーダー・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バス単位でレジスタファイル更新値を持つため、リオーダーバッファのようにインオーダー・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バスまで実行を進めた後、その分岐バ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

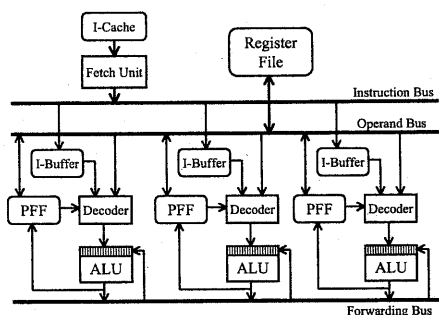


図 3: 分散フューチャファイルによるプロセッサの基本構成

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大規模な投機的実行を行なうプロセッサを構成する利点を考える。単純に従来の機構を大規模化する場合、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る。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がない。また、分岐命令によって実行状態が複数に分かれる場合にアーキテクチャ・ステートのコピーを必要としない。一方、演算結果を命令の順序によって複数の部分フューチャファイルに保持するため、レジスタ値出力の制御に複数サイクルかかる場合がある。

3.1 プロセッサ構成

前節で述べたモデルに従った基本的なプロセッサ構成を図3に示す。フェッチ・ユニットは命令バスを通じて各命令バッファに命令を送り、デコーダは命令バッファから命令を取りだし、近傍の部分フューチャファイルからオペランドを得る。近傍の部分フューチャファイルからオペランドが得られない場合は、オペランド・バスを通じ遠隔の部分フューチャファイルまたはレジスタファイルからオペランドを得る。その後、デコードした命令を命令ウィンドウに送る。演算器は命令ウィンドウを監視し、データが揃った命令から演算を行なう。演算結果は近傍の部分フューチャファイルとフォワーディング・

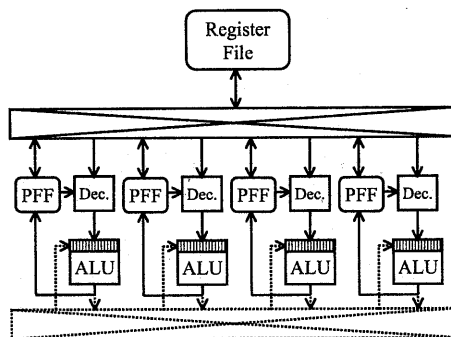


図 4: 分散フューチャファイルによるオペランド供給時のデータベース

バスに送られる。命令ウィンドウはフォワーディング・バスを監視し、演算結果をエントリに取り込む。

この構成では簡単のため各部分フューチャファイル命令バッファ、命令デコーダ、演算器を割り当てているが、PFF の結合(後述)などによりこれらのユニットは共有することが可能である。

図4に、分散フューチャファイルによるプロセッサ構成でのデータベースを示す。実線はオペランドのデータベース、破線はフォワーディングのデータベースである。なお、分散フューチャファイルではオペランドのデータベースを扱うため、フォワーディング・バスの構成にはこれとは別の機構を考慮する必要がある。図3では部分フューチャファイル間はバスで接続していたが、ここでは一般化してネットワーク接続としている。これは投機実行の規模が大きくなって実行中の分岐バス数が増えると、必要な部分フューチャファイルの数も増えるためバスが駆動できなくなるためである。大規模投機的実行においては部分フューチャファイル間のオペランド転送には階層化されたバスやクロスバ・スイッチなどの接続網が必要になると考えられる。

3.2 クラスタ化

分岐バスの平均の長さは predicate などによって分岐命令が除去されても数命令程度である [3][6]。このため、1つの分岐バス毎に部分フューチャファイルを割り当てた場合、頻繁に別の部分フューチャファイルからレジ

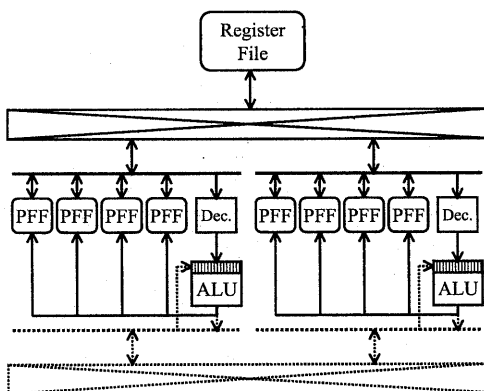


図 5: 4-PFF クラスタの例

スタ値を読み出すことになり、ペナルティが大きい。そこで部分フューチャファイルクラスタ化し、同クラスタ内では1サイクルで処理を行なう。クラスタの規模を上げることによりレジスタ値読み出しの遅延を削減できるが、ハードウェアの複雑さは上がる。4つの部分フューチャファイルによるクラスタの例を図5に示す。クラスタ内部では演算器やバスを共有できる。部分フューチャファイルをマッピングテーブルと物理レジスタファイルに分けて実装する場合、物理レジスタファイルを共有することも可能である。またクラスタが小規模であれば、部分フューチャファイルにチェックポイント回復機能を追加することで、クラスタ内部の部分フューチャファイルを1つに統合することが可能になる。また、クラスタに連続した分岐バスを割り当てる場合、1つのクラスタは単一バス実行と同様の動作になるため、設計が容易になる。

4 評価

分散フューチャファイルでは部分フューチャファイルを分散配置する。このため遠隔の部分フューチャファイルからオペランドを供給する時に、レイテンシが複数サイクルかかることが考えられる。このレイテンシがプロセッサ性能に与える影響を評価する。評価はトレースベースのシミュレータを作成して行った。シミュレート対象は命令セットをSPARC Version8、OSをSolaris 2.5.1、実行プログラムはSPECint95よりcompress, gcc, go, jpeg, li, perlとし、それぞれ

10,000,000 命令実行した。また、シミュレートした CPU の設定は7ステージ・パイプライン、8 命令同時デコード、分岐予測は2bit, 1024 エントリ, 4way set associative、ストアバッファ8本で、メモリは1次キャッシュ 64KB, 256bit/line, 4way set associative、2次キャッシュ 1MB, 256bit/line, direct map, 5 サイクル/アクセス、メインメモリ 50 サイクル/アクセスであった。

シミュレータは static tree heuristic[2] による複数バス実行制御を行ない、命令フェッチ、フォワーディングバスに関しては大規模化に伴うペナルティはないものとした。シミュレーションは表1に示す5種類の投機的実行規模について行ない、オペランド供給のレイテンシを変えてIPCの変化を評価した。

オペランド供給のレイテンシは部分フューチャファイルのクラスタ内では0とする。また、部分フューチャファイルはプログラムの命令順に割り当てられるとし、レイテンシは演算結果を持つ部分フューチャファイルとオペランドを要求する部分フューチャファイル間のクラスタ距離に比例するものとした。ここでは1クラスタにつき1サイクルとした。即ち演算結果を出力した命令からオペランドを要求する命令の分岐バス距離を部分フューチャファイル・クラスタの結合数で割ったものがレイテンシ・サイクル数となる。なお、同じ部分フューチャファイルで一度要求されたオペランドは部分フューチャファイルに保存され、2回目以降はレイテンシなしとする。

表 1: 投機実行の規模

主投機数	副投機数	PFF 数
7	0	8
15	1	17
31	3	38
47	7	76
63	15	184

4.1 結果

図6に compress でのシミュレーション結果を示す。図7,8,9,10,11はそれぞれ gcc, go, jpeg, li, perl の結果である。横軸は部分フューチャファイル・クラスタの結合数である。最小の1では親の分岐バスでの出力を参照する場合も1サイクルのレイテンシがあり、10分岐バス離れた演算結果をオペラン

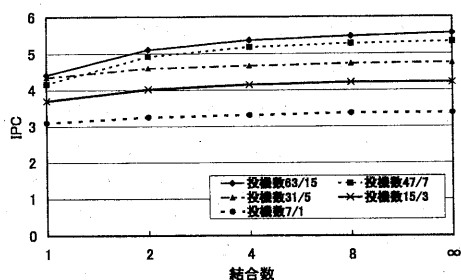


図 6: 結果: compress

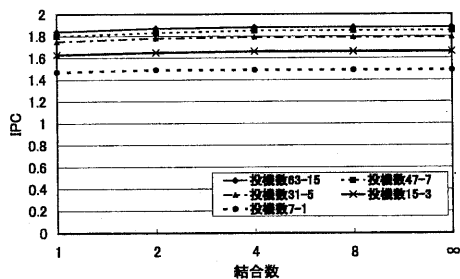


図 7: 結果: gcc

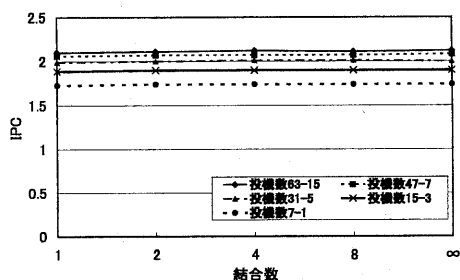


図 8: 結果: go

ドとする時は 10 サイクルのレイテンシが発生することになる。一方、無限大では他の部分フューチャファイルからオペランドを供給する時にもレイテンシが発生しない。これは即ち、全てのクラスタが 1 つのバスに接続されている等の構成である。

5 考察

compress, li, perl の大規模な投機の実行で性能が大きく向上するアプリケーションでは、部分フューチャファイル・クラスタ間のデータ転送レイテンシが大きい場合実行速度が低下することが分かる。特に結合数 1 では最大で compress の 20.1% と大幅に性能が

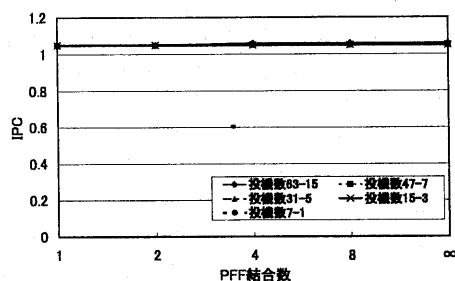


図 9: 結果: jpeg

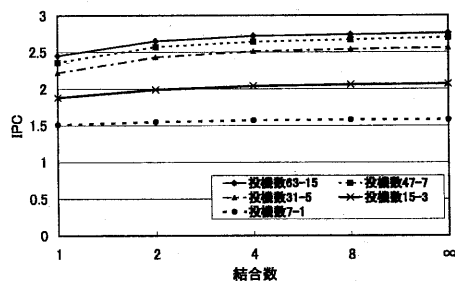


図 10: 結果: li

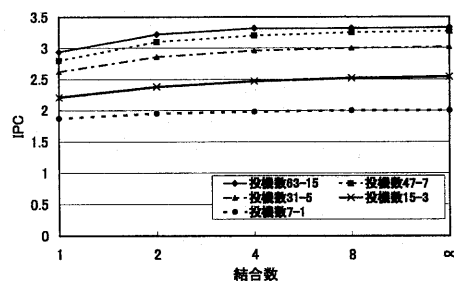


図 11: 結果: perl

低下している。しかし、結合数 4 では最大 3.5%、平均 1.0% 程度、結合数 8 では最大 1.5%、平均 0.5% 程度の性能低下となり、結合数が大きくなると性能低下は著しく減る。一方、gcc, go, jpeg といった大規模な投機の実行での性能向上が小さいアプリケーションでは、オペランド供給のレイテンシも性能に与える影響は少ない。

これより、大規模な投機の実行を行なう場合、部分フューチャファイル・クラスタの結合数を 4,8 程度にすればオペランド供給に関するレイテンシがプロセッサ性能を低下させる割合は低いことが分かった。結合数 4,8 程度の部分フューチャファイル・クラスタは、大規模な投機の実行全体を制御する一つのユニットに比べて、非常に複雑さが小さい。また、投機実行の規模を上げる際もクラスタを増やすだけで良いので比較的容易である。評価結果より、結合数 4,8 程度の部分フューチャファイル・クラスタは大規模投機の実行のオペランド供給機構として適していることが確認できた。

6 今後の課題

大規模に複数バスを実行するには本章で提案した分散フューチャファイル・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つは複数バ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、大規模に投機の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。オペランド供給は制御依存に従って分散化する機構が適していることを示したが、データ依存に従うフォワーディングについては異なる方式が必要になる。

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 [7]。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。今後の課題として、分散フューチャファイルによるオペランド供給機構と ALU-Net を組み合わせ、大規模に投機の実行を行なうプロセッサのデータバス

構成手法を確立することが挙げられる。

7 まとめ

本論文では大規模な複数バス実行に適したレジスタセット管理方式として分散フューチャファイル・モデルを提案した。また、データ転送による遅延がプロセッサ性能に与える影響を評価し、部分フューチャファイル数 4~8 程度のクラスタ化によって性能低下を抑えられることを示した。

本研究の一部は (株) 半導体理工学研究センターからの補助による。

参考文献

- [1] Mike Johnson: "Superscalar Microprocessor Design", Prentice-Hall (1991)
- [2] Augustus K. Uht and Vijay Sindagi: "Disjoint Eager Execution: An Optimal Form of Speculative Execution", MICRO-28, pp.313-325 (1995)
- [3] S. A. Mahlke and R. E. Hank and J. McCormick and D. I. August and W. W. Hwu: "A comparison of full and partial predicated execution support for ILP processors.", Proc. 22th Annual International Symposium on Computer Architecture, pp.138-150 (1995)
- [4] Tien Fu Chen: "Supporting Highly Speculative Execution via Adaptive Branch Trees.", The 4th International Symposium on High-Performance Computer Architecture, pp.185-194 (1998)
- [5] Artur Klauser, Abhijit Paithankar, Dirk Grunwald: "Selective Eager Execution on the PolyPath Architecture.", Proc. 25th Annual International Symposium on Computer Architecture, pp.250-259 (1998)
- [6] David I. August et al: "Integrated Predicated and Speculative Execution in the IMPACT EPIC Architecture.", Proc. 25th Annual International Symposium on Computer Architecture, pp.227-237 (1998)
- [7] 中村 友洋, 吉瀬 謙二, 辻 秀典, 安島 雄一郎, 田中 英彦: "大規模データバ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)
- [8] 安島 雄一郎, 中村 友洋, 吉瀬 謙二, 辻 秀典, 田中 英彦: "例外回復可能な複数バス実行機構の提案", 情報処理学会研究報告 98-ARC-129 (1998)