計算機アーキテクチャ　123－10
（１９９７．　３．　７）

# 複数分岐での投機的実行の有効性

児島 彰　　弘中 哲夫　　高山 毅　　藤野 清次

広島市立大学 情報科学部 情報工学科

〒731-31 広島市 安佐南区 大塚東 3 丁目 4-1

*E-mail: {kojima, hironaka, takayama, fujino}@csys.ce.hiroshima-cu.ac.jp*

最近のパイプライン型プロセッサでは、条件分岐でのパイプラインの乱れによる速度低下を抑制するために、分岐予測とそれに使った投機的実行が行われている。ただし、予測失敗時にはペナルティが大きいので、分岐予測の精度向上が望まれる。これまでの分岐予測では、単独の条件分岐に対して過去に分岐したか、どうかの履歴を取り、その情報に基づいて、分岐するかしかないかを予測している。本研究では、さらに分岐予測の精度を高めるため、連続して実行される複数の条件分岐の分岐パターンに対して履歴をとり、この履歴からの分岐予測を従来の手法に組み合わせる方法を提案する。また、この手法による分岐予測の有効性を実際のプログラムを用いて評価する。

# Branch Prediction based on Histories of Multi-Branch Execution Patterns for Speculative Execution

Akira KOJIMA　Tetsuo HIRONAKA
Tsuyoshi TAKAYAMA　Seiji FUJINO

Department of Computer Engineering,
Faculty of Information Sciences,
Hiroshima City University,
3-4-1 Ozuka-Higashi, Asa-Minami-Ku,
Hiroshima City, 731-31 Japan

*E-mail: {kojima, hironaka, takayama, fujino}@csys.ce.hiroshima-cu.ac.jp*

Superscalar processors and superpipeline processors use branch prediction and speculative execution to avoid suffering pipeline bubbles at conditional branches. The costs of recovering pipelines when occurring prediction mishits are large in high-level pipelined processors using speculative execution. Therefor, very accurate dynamic branch prediction is strongly required in such processors. Today's high performance processors use branch prediction based on history of single branch. In this paper, to explore more accuracy of branch prediction, we propose dynamic branch prediction based on histories of multi-branch execution patterns, and show the results of the performance evaluation with execution traces of real programs.

# 1 はじめに

スーパスカラ・プロセッサやスーパパイプライン・プロセッサなど、命令レベルでの並列処理を利用するパイプライン型のプロセッサでは、パイプラインの段数を増加させることによって高速化を図られている。しかし、単純なパイプライン化を行っただけでは、プログラム中に比較的頻繁に現れる条件分岐などにおいて、分岐の判断の条件が確定するまで待たなくてはならず、すなわち、パイプラインを停止させるというパイプラインの乱れが発生して、結果として速度が低下してしまう [1]。そこで、最近のプロセッサでは、条件分岐でのパイプラインの乱れを抑制するために、分岐の判断の条件が確定する前に、条件分岐の分岐予測を行い、その予測に基いた投機的実行を行うことにより、パイプラインが停止することをなるべく少なくするように改良されている。ところが、高度にパイプライン化されたプロセッサでは、投機的実行は予測が的中しなければ、その度に正しいところまで状態を戻す処理が必要になり、逆に予測失敗のペナルティが大きくなってくるので、高い分岐予測の精度が強く要求される。一般にパイプラインの段数が多く高速なプロセッサほど、予測失敗時の状態復帰の処理が複雑になり、復帰にかかる時間は長くなるので、分岐予測精度の向上への要求は極めて大きい。

条件分岐で分岐するか、分岐しないかの振舞いは、過去のその条件分岐での振舞いと似た結果になることが多い。そのため、分岐予測では、この性質を利用し、条件分岐での振舞いを履歴として記録し、これに基づいて予測を行うのが一般的である [2]。これまでのに実現されている分岐予測では、単独の条件分岐ごと過去に分岐したか、分岐しなかったかの履歴を取り、その情報に基づいて、分岐するかしかないかを予測していた。本研究では、さらに分岐予測の精度を高めるため、連続する複数分岐の分岐パターンに対して履歴を取り、この履歴から分岐予測し、さらに従来手法と組み合わせる方法を提案する。

本稿では、まず、2章で従来の分岐予測手法について述べる。3章では、複数分岐の分岐パターンに対して取った履歴に基づく分岐予測手法について述べる。4章では、実際のプログラムでの実行トレースから、各手法の予測精度を評価する。最後に5章で本稿をまとめる。

# 2 従来の分岐予測方式

## 2.1 1ビットの履歴情報を使った分岐予測

従来より分岐予測に使われているのは、単独の分岐で、過去にどの様に分岐したかの履歴を取り、その情報から、分岐予測を行う方法である [1][2]。最も単純な方法としては、毎回条件分岐の実行の度に各条件分岐の分岐方向を1ビットのデータとして1回分だけ記録しておき、次回に同じ条件分岐にきたときは、前回と同じ分岐方向に実行されると予測する方法である。しかし、1ビットのデータで1回分だけ履歴を使う方法では、小さな例外によるに変動に弱く、予測確率は高くない。この方法では、ほとんど分岐方向が一方向であるような高い確率の分岐予測が可能な条件分岐で、1回の例外的な条件分岐で、2回のミスヒットを起こしてしまう。例えば、ループを形成する10回中9回は分岐成立するような条件分岐の場合を考えると、1回の条件不成立で、まず不成立による1回のミスヒットを起こし、そして、多くの成立が続く状態に戻ることより、もう1回のミスヒットを起こし、合計2回のミスヒットを起こしてしまう。この結果、予測的中率は、90%でなく2回のミスヒットにより80%になってしまう。

最近のプロセッサで多く採用されているのは、1回分だけでなく複数回分の分岐履歴を記録し、それを元に分岐予測する方法で、小さな例外によるに変動にも強く、比較的高い予測精度を達成している。

## 2.2 2ビットの履歴情報を使った分岐予測

MIPS R1000, DEC Alpha21164, Sun Ultra SPARC, IBM Motorola PC620 などでは、2ビットの履歴情報を使った分岐予測が採用されている [5][7]。各分岐に対しての過去の履歴から、

- 高い確率で分岐成立 (stronly taken)
- 低い確率で分岐成立 (likely taken)
- 低い確率で分岐不成立 (likely not taken)
- 高い確率で分岐不成立 (stronly not taken)

の確率を表す4つの状態を割り当て、状態を2ビットで表現し、これに基づいて分岐予測を行う。予測ビットの状態が前の2つならば、分岐すると予測し、後の2つならば、分岐しないと予測する。実際の分岐条件が確定したとき、分岐するか、分岐しな

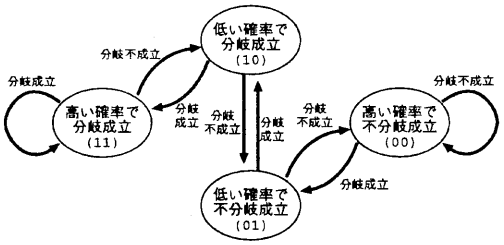



図 1: 2 ビット履歴 アルゴリズム (a)

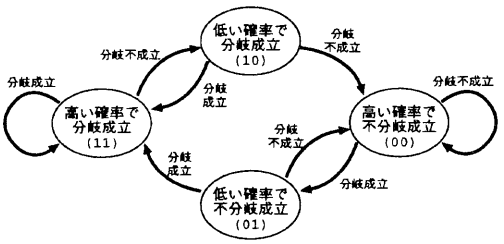



図 2: 2 ビット履歴 アルゴリズム (b)

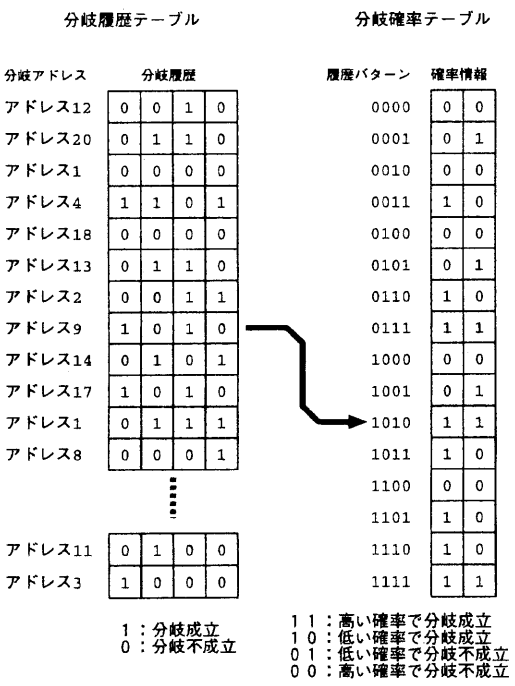



図 3: Yeh のアルゴリズム

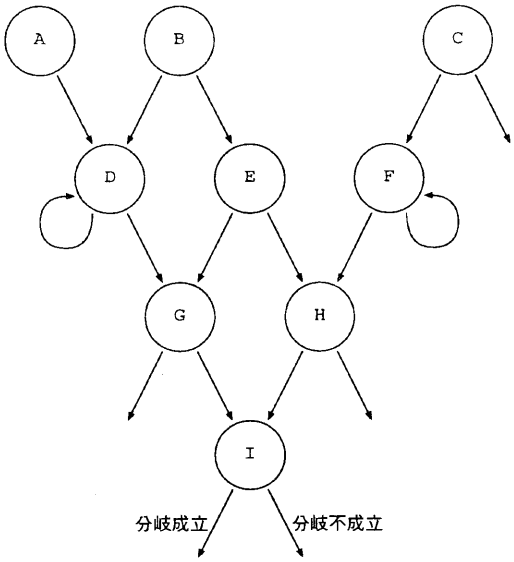
いかで、各分岐に対する 2 ビットの予測ビットを状態遷移させる方式が取られている。状態の遷移の方法は、図 1、図 2のようになっている。「低い確率で分岐成立」「低い確率で分岐不成立」の状態からの状態遷移のアルゴリズムは、方式により異なり、R1000,Alpha21164,PC620 などでは、図 1のアルゴリズム (a) が採用され、Ultra SPARC では、図 2のアルゴリズム (b) が採用されている。

この方式では、分岐確率が安定した条件分岐に対して高い予測が得られる。1 ビットの履歴を使う方法では、ミスヒットの大きかった、小さな例外がある場合の予測も比較的正しく予測できることが多い。しかし、2 ビットの履歴情報では、偶数、奇数ごとで、交互に分岐成立、分岐不成立が入れ替わるような時々出現する可能性のある特定の分岐パターンで、著しく予測が低下する場合もある。

## 2.3 Yeh のアルゴリズムによる分岐予測

Intel Pentium Pro などの場合、さらに、2 レベルの履歴を使う Yeh のアルゴリズム [3][4] で予測確率を上げている [6]。Yeh のアルゴリズムでは、各分岐の過去の数回分の実行履歴を格納する分岐履歴テーブルと、分岐履歴のパターンに分岐予測確率を対応づける分岐確率テーブルの 2 段階の履歴データを使って分岐予測を行う。その様子を図 3に示す。分岐の予測は、その分岐での過去数回分の履歴のパターンから、分岐確率テーブルから対応する分岐確率を取り出し、「高い確率で分岐成立」「低い確率で分岐成立」の状態ならば、分岐が成立すると予測する。実際の分岐条件が確定すると、分岐履歴テーブルと、分岐確率テーブルの両方を更新する。分岐確率テーブルの更新は、分岐予測に使ったエントリを予想確率の状態と見なし、先に述べた 2 ビットの履歴を使う方法と同じアルゴリズムで更新する。すなわち、テーブルの各エントリの状態変数を、成立したか、成立しなかったかで、図 1、または、図 2のアルゴリズムの通りに状態遷移、更新させる。

Yeh のアルゴリズムを使う方法では、特定のパターンを分岐確率テーブルに学習させる効果があり、先に述べた 2 ビットの履歴情報を使う方法では苦手とした、偶数、奇数ごとで、交互に分岐成立、分岐不成立が入れ替わるような時々出現する可能性のある特定の分岐パターンでも、学習により高い精度で予測できる。




分岐パターン

```
A - D - G - I
D - D - G - I
B - D - G - I
B - E - G - I
B - E - H - I
C - F - H - I
F - F - H - I
```

図 4: 分岐パターン

## 3 複数分岐の履歴

従来の分岐予測の手法は主に単独の分岐に対しての履歴で、分岐予測を行っていた。実際のプログラムの実行トレースを調査し、実行過程での一連の複数の条件分岐に着目すると、ある特定の分岐の前に通過してきた分岐パターンによって、その条件分岐が実行されるか、実行されないかに、確率的な偏りがあることがわかった。特に特定の分岐パターンで決まった分岐方向になることが、出現頻度の高い条件分岐の中のいくつかに確認された。つまり、一連の複数の条件分岐で形成される分岐パターンの履歴をとることで、条件分岐に対する分岐予測ができることが分かる。分岐パターンは、図4に示すように、予測したい条件分岐の前に行われた条件分岐の流れのパターンである。図4では、条件分岐 I の分岐予測をするために、条件分岐 I に到達する分岐パターンを取り出している。このそれぞれの分岐パターンに対して、履歴を取る操作を行い、その情報を元に分岐予測する。

単独の分岐のみに注目する方法では、小さなループを形成する条件分岐などの場合は、高い精度の予測が可能であるが、中規模以上のループの繰り返し部に含まれる条件分岐などに対しては、予測精度が悪くなる可能性がある。分岐パターンの振舞いを調べることで、ループの繰り返し部に含まれるような条件分岐でも、予測精度を向上できる場合があることが調査から分かった。さらに、分岐パターンに対して、先に述べたような、従来の手法で用いた履歴情報を与え、従来の手法と組み合わせると、より高い精度の分岐予測が可能性がある。分岐パターンには特定のパターンに確率的な偏りがあることが確認されているので、分岐パターンに1回分の履歴で、前回と同じ振舞いを条件分岐がすると単純に予測しても、ある程度の予測精度は達成できるが、前述の従来の手法と組み合わせると、より効果的であると考えられる。以下に組合せた手法の例を説明する。

### 2bit の履歴情報を使う方法

先に述べた2bit の履歴情報を与える方法を、特定の条件分岐に到達する各分岐パターンに適用する。各分岐パターンごとに、2bit の履歴情報を与え、先の4つの状態を割り当てる。特定の分岐パターンが出現したとき、それに対する 2bit の履歴情報による状態を調べ、「高い確率で分岐成立」「低い確率で分岐成立」の2つの状態ならば、分岐すると予測する。実際の分岐の条件が確定後は、履歴の状態を図1、図2のアルゴリズムに従って状態遷移させる。

### Yeh のアルゴリズムを使う方法

先に述べた Yeh のアルゴリズムを分岐パターンに適用する。特定の条件分岐に到達する各分岐パターンごとに、数回分のの履歴情報を保持する履歴テーブルと、履歴のパターンに対する予測確率を対応づける確率テーブルを用意する。分岐予測は、実行してきた分岐パターンに対する履歴情報を取り出し、さらにその履歴パターンに対する予測確率を確率テーブルから取り出し、「高い確率で分岐成立」「低い確率で分岐成立」の状態ならば、分岐が成立すると予測する。実際の分岐条件が確定すると、履歴テーブルと、確率テーブルの両方を更新する。確率テーブルの更新は、先に述べた2ビットの履歴を使う方法と同じアルゴリズムで更新する。

表 1: JPEG 圧縮の分岐予測精度

| | 2bits(a) | 2bits(b) | Yeh(4a) |
|---|---|---|---|
| 単独分岐 | 88.19 % | 87.97 % | 87.10 % |
| 2 分岐パターン | 88.23 % | 88.00 % | 87.16 % |
| 3 分岐パターン | 88.83 % | 88.54 % | 87.81 % |
| 4 分岐パターン | 89.19 % | 88.98 % | 88.16 % |
| 5 分岐パターン | 89.34 % | 89.13 % | 88.34 % |
| 6 分岐パターン | 89.39 % | 89.21 % | 88.41 % |
| 10 分岐パターン | 90.41 % | 90.23 % | 89.54 % |

表 2: JPEG 展開の分岐予測精度

| | 2bits(a) | 2bits(b) | Yeh(4a) |
|---|---|---|---|
| 単独分岐 | 93.31 % | 93.19 % | 92.81 % |
| 2 分岐パターン | 93.42 % | 93.19 % | 92.89 % |
| 3 分岐パターン | 93.68 % | 93.54 % | 93.10 % |
| 4 分岐パターン | 93.64 % | 93.47 % | 93.19 % |
| 5 分岐パターン | 93.72 % | 93.58 % | 93.50 % |
| 6 分岐パターン | 93.91 % | 93.80 % | 93.55 % |
| 10 分岐パターン | 94.47 % | 94.34 % | 93.92 % |

# 4 評価

各手法での分岐予測精度の評価を行うために、画像ファイルを JPEG 圧縮/展開するプログラムを用いた。プログラムは、Sun SPARC Solaris 2.X 用にコンパイルし、227 × 149 24-bit-color の自然画の画像を実際に圧縮/展開させ、その際の全ての実行トレースを採取し、条件分岐の分岐状況について調べた。

2 ビットの履歴情報を使い、図 1のアルゴリズムを使う場合と図 2のアルゴリズムを使う場合、4 回の分岐履歴と図 1の状態遷移を使った Yeh のアルゴリズムを使う場合を調べた。履歴情報は、単独分岐と、2,3,4,5,6,10 の分岐間の分岐パターンに対しての情報を採用した場合について調べた。その結果を表 1、表 2に示す。

画像の JPEG 圧縮プログラムでは、単独分岐に対する履歴情報のみを使う従来手法に比べて、分岐パターンに対する履歴を使う手法は、2 ビット状態履歴を使う方法、Yeh のアルゴリズムを使った方法の両方共で、5 分岐の分岐パターンの履歴を使った場合は、予測精度の向上が 1%程度あり、10 分岐の分岐パターンの履歴を使った場合は、予測精度の向上が 2%程度あった。

一方、画像の JPEG 展開プログラムでは、単独分岐に対する履歴情報のみを使う従来手法でも、予測精度は 93%程度とかなり高い。分岐パターンに対する履歴を使う手法では、5 分岐の分岐パターンの履歴を使った場合は、予測精度の向上が 0.5%程度、10 分岐の分岐パターンの履歴を使った場合は、予測精度の向上が 1%程度であり、圧縮操作に比べて精度向上は少なかった。展開操作では、圧縮されたデータを補間する処理で、規則性のある処理が多くなり、その結果、単純な方法でも分岐予測が当たり易くなっていると考えられる。また、多く規則性が期待できない一般の処理での分岐予測は、JPEG 展開処理の結果に近いと考えられる。

今回の評価で使用したプログラムでは、2 ビットの履歴情報を使う方法では、うまく扱えないような特定の分岐パターンは、多くは出現していないと考えられ、2 ビットの履歴情報を使う場合と、Yeh のアルゴリズムを使った場合とで、それほど大きな違いはなかった。しかし、いずれ方法の場合も分岐パターンへの履歴に適用すると、程度の差はあるものの速度向上があることが確認された。

表には記載していないが、単独の分岐に対して 1 ビットの履歴で分岐予測行った場合、JPEG 圧縮で 60%程度、JPEG 展開で 75%程度の予測精度であった。分岐パターンに対して 1 ビットの履歴で分岐予測行った場合には、1〜2%程度の性能向上があった。しかし、これだけでは、いずれの場合も予測精度が低く、実用的な方法とはいえない。

高度にパイプライン化されたプロセッサでは、分岐予測が失敗したときのペナルティが大きい。Pentium Pro では、12 段にパイプライン化されており、パイプラインが円滑に動作している時は、1 サイクルで平均 3 命令を実行するが、分岐予測が失敗したときには、正しい命令から実行を再開するのに、10〜15 サイクルかかる。今回、評価に使用したプログラムでの実行トレース中の条件分岐の出現頻度は、約 8.6%であった。今回の方式で、予測精度が 88%から 89%に向上したとして、キャッシュミスなどの影響を無視し、条件分岐の予測失敗以外ではパイプラインが円滑に動作していると仮定し、この数値を Pentium Pro にあてはめると、約 2.0〜2.7%程度の速度向上が見込まれる。

将来、パイプラインの段数は 12 段からは劇的に増えるとは予想しにくいが、VLIW の様な方式で、

実行ユニットがさらに増えたプロセッサが出現することは予想できる。将来のプロセッサで、1サイクルで平均10命令を実行するとし、分岐予測が失敗したときには、正しい命令から実行を再開するのに、10〜15サイクルかかるとし、今回の方式で、予測精度が88%から89%に向上したと仮定すると、このとき、約4.3〜5.3%程度の速度向上が見込まれる。また、将来はより大きい規模の投機的実行が行われると予想できるが、予測失敗時の修復もより大きなコストがかかるので、予測精度の向上による速度向上は、より大きなものになると考えられる。

## 5 おわりに

本研究では、単独の条件分岐の履歴情報を使う従来の手法を、複数の条件分岐からなる分岐パターンに対しての履歴情報に適用する分岐予測の方法について提案した。実際のプログラムで、命令レベルの実行トレース中の全ての条件分岐を分析することで、この手法の有効性を評価した。その結果、単独分岐の履歴のみを使う手法より、分岐パターンに対しての履歴を使う方が予測精度が向上することが確認された。

単独分岐の履歴と分岐パターンに対する履歴を組み合わせて使う方式については、改良の余地がある。また、実際にハードウェアとして実装する際は、履歴保持のためのバッファは有限であるので、今後は、バッファ容量と予測性能の特性を評価し、本方式を用いた場合のハードウェア量、実装コストと、性能向上の関係を詳細に調査する必要がある。

## 謝辞

本研究を進めるにあたり、ご協力いただきました、広島市立大学 情報科学部 情報機械システム工学科 竹迫良範 君に、謹んで感謝いたします。

## 参考文献

[1] Hennessy, J. L. and Patterson, D. A. (富田眞治, 村上和彰, 新實治男 訳): *Computer Architecture : A Quantitative Approach*, Morgan Kaufmann Publishers,Inc.,1990.

[2] Lee,J.K. and Smith,A.J : "Branch Prediction Strategies and Branch Target Buffer Design", IEEE Computer, vol.17, no.1, pp.6–22, Jan. 1984.

[3] Yeh,T. and Patt,T. : "A Comparison of Dynamic Branch Predictors that Use Two Levels of Branch History", Proceedings of the 20th International Symposium on Computer Architecture, pp.257–266, May 1993.

[4] Yeh,T. and Patt,T. : "Branch History Table Indexing to Prevent Pipeline Bubles Wide-Issue Superscalar Processors", Proceedings of the 26th Annual International Symposium on Microarchitecture, pp.164–175, Dec. 1993.

[5] Marc Tremblay, Dale Greenley and Kevin Normoyle: "The Design of the Microarchitecture of UltraSPARC-I", Proceedings of the IEEE, vol.83, no.12, pp.1653–1662, Dec. 1995.

[6] 秋庭正之, 山本和已: "12段のパイプラインで高速化図った Pentium Pro プロセッサ", 日経エレクトロニクス, 1995.12.4 (no.650), pp.137-151, 1995.

[7] "次世代マイクロプロセッサ", 日経エレクトロニクス, 1995.1.30 (no.627), pp.67-150, 1995.