プロファイルを用いた値予測命令削減手法

飯塚大介[†]　小沢年弘[††]
坂井修一[†]　田中英彦[†]

値予測は、真の依存関係を越えた並列度を抽出する手法である。通常、ハードウェアによる値予測は予測成功率の高い命令について予測を行なう。これらの予測命令中には、並列度を上げるものとそうでないものが存在するため、予測命令中には並列度の向上に寄与しない命令が多く含まれることになる。本論文では、プロファイルを用いて、そのような冗長な予測を静的に削減することで、総予測命令数を削減する手法を提案する。この手法を適用した結果、最大で総予測命令数を半分に削減し、予測命令を削減しない場合とほぼ同等の速度向上を得ることができた。

Decreasing Value Predicted Instructions Using Profile

Daisuke Iizuka,[†] Toshihiro Ozawa,[††] Shuichi Sakai[†]
and Hidehiko Tanaka[†]

Value prediction is a new technique to improve parallelism by breaking true data dependences. Most of proposed architectures with value prediction support, are designed to selectively predict instructions with high hit prediction rate. However, not all of these instructions contribute to more parallelism. This paper describes a new technique to eliminate redundant value predictions using profile data. Evaluation results show that we can eliminate number of predictions up to half of normal number of predictions, with only a little performance degradation.

1. はじめに

アウトオブオーダ実行を行なうスーパースカラプロセッサは、プログラムから命令レベル並列性を動的に抽出することで、実行速度の向上を図っている。しかし、真の依存関係にある命令が存在するため、従来の手法では並列度の抽出には限界がある。近年、この真の依存関係を解決するための投機実行として、値予測が考案されている[3)~5)]。

値予測は投機実行のうちの一つであり、予測した結果が正しければ速度向上を得ることができるが、誤っていた場合はその結果を使用した全ての命令が生成した値を破棄し、予測開始時点から再実行する必要がある。従って、予測のミス率を下げることが値予測による性能向上を得るための条件の一つとなる。また、値を生成する全命令が値予測を行なう候補となるため、ミス率を下げつつ、多くの命令について値予測を行なうことでも性能向上を計ることができる。

しかし、予測を行なう命令の中には予測による性能向上効果のない命令も含まれるため、このような命令についても予測を行なうと、予測命令数が増大し、効果のない予測命令のために余分なハードウェア量が必要となってしまう。また、値履歴テーブルの競合が発生し、かえって予測正解率が低下することも考えられる。

本研究では、そのような効果のない、あるいは少ない値予測命令をプロファイルを用いて静的に選択して総予測命令数を削減することで、実行時に予測命令を削減しなかった場合と比較して同等の速度向上を得るための手法を提案する。これにより、値予測に必要なハードウェア量も少なくて済むため、結果として効率良く値予測を行なうことができる。

† 東京大学大学院 工学系研究科
Graduate school of Engineering, The University of Tokyo
†† 富士通研究所
Fujitsu Laboratry

2. 値予測機構

2.1 値予測による速度向上

ある命令 B が、命令 A の実行結果に依存する場合、通常は命令 A の結果が出なければ命令 B を実行することができない。しかし、A の実行前に A が生成する値を予測することができれば、図 1のように通常よりも 1 サイクル早く実行することができる。 このように、値予測は真の依存関係を解消することで速度向上を計っている。

2.2 値予測の方式

値予測は種々の方式が提案されている。本節では値予測方式のうち、主用なものと本研究と関連のあるものについて紹介する。

2.2.1 Last-value 予測機構

Last-value 予測機構は、最も近い過去に得られた値

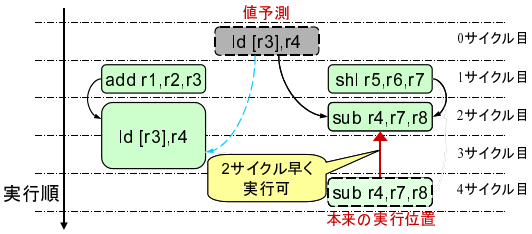

図 1 値予測による性能向上

を予測値とする予測機構である。[3),4)]。

予測正解率を高めるため、文献[3)]では、予測ミスの回数削減のために、動的に変化するカウンタからなる分類テーブル (Classification Table, CT) を用いる値予測機構を提案している。

2.2.2 ストライド値予測機構

ストライド値予測機構は、最も近い過去の 2 回の差分 $Stride$ と、最も近い過去に得られた値 $Value$ から、今回の値を $Value + Stride$ として予測する予測機構である[5)]。ストライド値予測機構で良く当たるものとして、誘導変数がある。今、二重ループの内側の誘導変数の予測を行なうとした場合、内側のループを実行中は誘導変数はある期間一定の割合で増減しているが、内側のループの実行が終了して外側のループに処理が移り、再び内側のループに処理が戻って来た場合、誘導変数は初期化されるので予測ミスを起こしてしまう。これを防ぐため、文献[5)]では $Init$,$Transient$,$Steady$ という 3 状態を用いて予測ミスを削減する手法を提案している。

2.2.3 ソフトウェアによる値予測支援

値予測に関して、ソフトウェアによる支援を行なう手法が提案されている。

文献[2)]では、このような手法として、プロファイルを用いて、あらかじめ予測成功率の高い命令についてのみ予測を行なう手法を提案している。この手法によると、予測をおこなうかどうか判定する閾値を 60% 以上に設定することで、ハードウェアのみよりも低いミス率を達成できるとしている。文献[2)]も本論文で提案する手法もプロファイルを用いているが、前者は予測成功率の高い命令を予測対象とするのに対し、後者は値予測による性能向上の得られない命令を予測対象から除外するという点が異なっている。

また、文献[6)]では、コンパイル時に値予測のための特別な命令を挿入し、予測はハードウェアで行なうが、予測ミスが起きた時の巻き戻しの処理はソフトウェアで行なう手法を提案している。

2.2.4 ハードウェアによる予測命令の選択

文献[1)]では、ハードウェアにより予測命令を選択する手法を提案している。この手法は、ある予測命令が命令ウィンドウ中にある場合、その命令が実際に実行されるまでのレイテンシがある一定の値以上である場合にのみ予測を行なうというものである。また、その予測命令によって生成された値を使用する命令に関しても考慮して予測命令を選択している。本論文でも文献[1)]と同様に予測命令を選択しているが、本論文は静的に予測命令を選択するために、文献[1)]と比較して予測機構に必要なハードウェア量が少なくてすむという利点がある。

3. 従来の予測機構の問題点

本章では、従来の値予測機構の問題点について述べる。なお、以降「静的な命令」とはコンパイラによって生成された命令のことを指し、「動的な命令」とは、実際に実行されたそれぞれの命令を指すものとする。

3.1 値予測により得られる速度向上を大きくすることによる問題

今、静的なある命令について値予測を行なうこととする。この命令の予測正解率を p とし、値予測を行なうことで g サイクル早くデータ依存関係が解決し、予測が失敗したことが判明してから予測開始時点に巻き戻すまでに m サイクルかかったとする。すると、この命令の値予測を行なうことで、この命令により生成されるデータは、式 1で表される期待値の分だけ、本来よりも早く生成されることになる。

$$\text{speedup} = pg - (1-p)(g+m) \qquad (1)$$

従って、全ての値予測を行なう動的な命令についてこの合計をとった、$\sum$speedup を大きくすることが、性能向上のための条件の一つとなる。以下、この値を大きくすることにより生じる問題について説明する。

3.1.1 予測機構の複雑化

式 1からわかるように、予測正解率 p を上げると speedup は向上する。従って、予測ミスを減らすために、2 レベルストライド値予測のように分岐履歴も考慮した予測機構などが提案されている[7)]。しかし、そのような予測機構を用いた場合、確かに予測率を上げることはできる。しかし予測機構が複雑になる場合、ハードウェアの実装上の制約から、実際にプロセッサに搭載することが困難になることが考えられる。

3.1.2 予測命令数の増大

ある静的な命令について、当該命令を実行する際に、大部分の実行に対しては値予測を行なうことにより得られる利得 g が 0 となるが、一部分の実行に際して利得 g が 0 より大きくなることがある場合、予測正解率 p が低くても、式 1の値が正になる場合がある。そのような命令についても値予測を行なえば $\sum$speedup は増大する。

しかしながら、値予測は分岐予測やアドレス予測とは異なり、値を生成する全命令が対象となるため、そのままでは予測命令数が膨大になってしまい、予測機構に必要なハードウェア量が増大する。

また、予測命令数が増大すると、1 サイクルに予測する命令数も増大する。しかし、値履歴テーブルの大きさは有限であるため、履歴テーブルの競合により速度向上効果が得られなくなる可能性が高くなる。

3.1.3 予測による速度向上が得られない命令の存在

分岐予測は分岐命令が対象となり、予測が当たればパイプラインがストールしないため速度向上が得られる。

一方、アドレス予測はロード・ストア命令が対象となり、投機的に実行したロード命令によって生成された値を使用する命令が存在すれば速度向上が得られる。

値予測はアドレス予測と同様に、予測値を使用する命令が存在すれば速度向上を得られる。しかし、値を生成する全命令が予測対象となるために、予測した値を使用する命令が存在しない確率はアドレス予測よりも大きくなる。従って、予測値が使用されない値予測は予測を行

表 1　値予測を行なうプログラム例

C 言語		アセンブラに変換
r3 = r1 + r2;		add r1,r2,r3
r4 = *r3;	→	ld [r3],r4 /* Last-value で予測 */
r7 = r5 << r6;		shl r5,r6,r7
r8 = r4 - r7;		sub r4,r7,r8

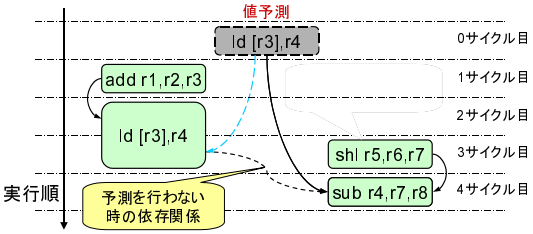

図 2　効果のない値予測

なう意味がなく、予測ミスを起こした際に速度低下を招いてしまうという問題も生じてしまう。また、ある静的な予測対象命令が非常に多く実行されるのに、生成された値が使用される場合は非常に少ない場合にも同様のことが言える場合がある。

よって、値予測を行なう際には予測正解率の高い命令を選択するだけでなく、実際に予測による速度向上効果のある命令についても選択を行なうことが重要となる。このような命令を選択することで、総予測命令数を削減しながら削減前と同等の速度向上を得ることができる。

また、このようにして予測命令数を削減すれば、第 3.1.1節や第 3.1.2節で生じる問題も解決できる。そのため、予測命令数を削減することは、実際のハードウェアに値予測機構を実装する場合にも重要となる。

3.2　値予測による効果のない命令

本節では、予測による速度向上効果がない、あるいは少ない命令にはどのようなものがあるかについて述べる。これらの命令を除外することで、前節で述べたように総予測命令数を削減することができる。

3.2.1　予測による速度向上効果がない命令

今、表 1のようなプログラムについて考える。ロード命令のレイテンシは 2 で、それ以外の命令のレイテンシは 1 であるとする。命令ユニットは無限にあるとする。予測を行なわない場合、命令間のデータ依存により 4 番目の命令は図 1の本来の実行位置で実行されるものとする。ここで、図中の灰色の四角は当該命令が実行中であることを表しており、命令間の矢印はデータ依存関係を示している。

今、プログラム中の 2 番目にあるロード命令が生成する値を値予測するものとし、図 1のように実行され、正しく予測が行なわれたものとする。これによって、ロード命令によって生成される値は、値予測により 2 サイクル早く取り出され、その値を使用する 4 番目の命令は 2 サイクル早く実行できるようになる。これは、図 1の場合は、 2 番目のロード命令の実行が終了する以前に 3 番目の命令の実行が終了しているため、ボトルネックとなっている 2 番目のロード命令の値予測を行なうことで 4 番目の命令の実行を早めることができるのである。

しかし、 3 番目の命令の実行が 2 サイクル後ろにずれた場合は、図 2のように実行される。この場合には、ロード命令により生成される値は図 1と同様に 2 サイクル早く取り出されるが、その値を使用する 4 番目の命令は通常実行と変わらないタイミングで実行される。これは、図 2で値予測を行なわない場合は、 2 番目のロード命令と 3 番目の命令の双方がボトルネックとなっているため、データ依存の片方だけが値予測行なうことで解決されても、もう片方は通常に実行された後に解消されるため、結局 4 番目の命令は通常と同様のタイミングで実行される。従ってこの場合、 4 番目の命令は値予測によって速度向上を得ることができない。

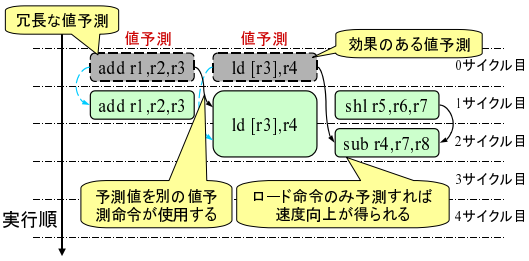

図 3　冗長な値予測の削除

このように、ある命令の実行に対してボトルネックとなっている命令を値予測する場合には、予測によってデータ依存が通常よりも早く解決することにより、後に続く命令の実行を早めることができる。反面、ボトルネックとなっていない命令は値予測を行なっても速度向上を得ることができない。また、ボトルネックとなっていない命令は、予測正解率が 100% でない場合は、予測ミスによるペナルティーも受けてしまうため、速度低下の要因となる。

3.2.2　予測による速度向上効果が少ない命令

ある静的な命令について考える。当該命令が実行時に複数回実行されるとすると、実行されたそれぞれの場合において、図 1のように値予測による効果が得られる場合と、図 2のように予測効果が得られない得られない場合の両方の場合が存在することがある。

このような場合に、当該命令の実行うち、大部分が予測による速度向上が得られないものであり、速度向上が得られるのがわずかである場合、その命令を値予測することにより得られる速度向上は少ないことになる。

従って、このような値予測を削除すれば予測命令を大幅に減らすことが可能となると思われる。

3.2.3　冗長な値予測

ある値予測を行なう命令で生成される値を、別の値予測を行なう命令が使用する場合、前者の命令を値予測することによって速度向上が得られないことがある。例えば、図 3のように、加算命令とロード命令の値予測を行なう場合を考える。前者の加算命令で生成される値を後者の値予測を行なうロード命令のみが使用する場合、依存関係の一番下にある減算命令はロード命令についてのみ予測を行なえば速度向上を計ることができるため、前者の加算命令の値予測は冗長となる。

従って、このような冗長な値予測命令を削除することによって、速度向上率を落さずに予測命令を削減することができる。

4. プロファイルを用いた値予測命令削減手法

第 3.2節で説明したように、予測を行なう命令中には速度向上効果の得られない、あるいは少ない命令が多々含まれる。従って、予測候補となる命令からこれら値予測による速度向上効果のない命令を除外し、速度向上効果のある命令を選択することで、総予測命令数を削減しつつ削減前と同等の速度向上を計ることができる。

このように予測する命令を削減させることにより、値予測に必要なハードウェア量を少なくできるため効率の良い値予測を行なうことができることになる。

本章では、そのように予測命令を削減するための命令選定アルゴリズムとなる条件を 3 つ提示する。実際には、ここで提示した 3 つの条件をいくつか組み合わせて予測命令を選定することで、総予測命令数を削減する。

4.1 変数の定義

本節では、命令選択アルゴリズムで使用する命令を選択するアルゴリズムを説明するために必要な変数の定義を行なう。以下でおこなう変数の定義のための説明図を図 4に示す。

なお、以下の変数の定義中に出てくるレイテンシは、動的な命令それぞれについて、値予測を開始した時点を基準として数えるものとする。

VPDataReadyLatency
: 値予測を行なう命令が予測を行なってから、実際に実行して正しい値が生成されるまでのレイテンシ。

VPExecTimes
: ある静的な予測命令が値予測を行なった回数。

VPDestLatency
: 正しい値が予測された時点から、予測値を最初に使用する命令が実行されるまでのレイテンシ。予測値を使用する命令が存在しなかった場合は 0 とする。

VPDestExistTimes
: 静的な予測命令について、正しく予測された際に予測値を使用する命令が存在した回数。

VPGain
: 正しく予測された時点から、予測値を最初に使用するまでのレイテンシと、予測命令が実際に実行され、正しく計算された値が生成されるまでのレイテンシの差。ただし VPGain < 0 となる場合や、予測値を使用する命令が存在しない場合は 0 とする。

VPPenalty
: 値予測が外れた場合において、予測を開始する直前の状態に巻き戻すまでのレイテンシ。正しく予測された場合は 0 とする。

4.2 予測率 99% の命令

予測ミスによるペナルティーを抑えるために、プロファイルを用いて予測正解率が 99% 以上となる命令をあらかじめ選択しておくこととする。この方式で選択された命令を用いて値予測を行なう方式を、「予測率 99%」方式と名付け、以降これを予測命令のベースとする。このベースの中からさらに、プロファイル情報をもとに後述する 3 つの条件をそれぞれ適用して予測命令を削減し、動的な予測命令数と性能向上率の比較を行なうこととする。

4.3 予測命令選択条件

本節では予測命令を選択する 3 つの条件を提示する。

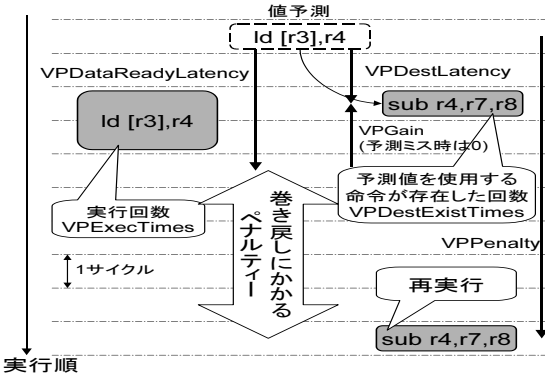

図 4 変数説明図

4.3.1 条件 1: 利益のある命令を選択

ある値予測を行なう命令に対して、予測値を最初に使用する命令が存在している場合を考える。その値予測命令が予測値を使用する命令の実行に対してボトルネックとなっている場合は、予測を行なうことで予測値を使用する命令の実行を早めることができる。

このとき、予測値を使用する命令は、値予測を行なわなかった場合と比較して、 VPGain の分だけ早く実行されることになる。

そのため、同一 PC をもつ値予測命令に関して、値予測を行なった結果予測値を最初に使用する命令の実行が早くなるサイクル数を合計すると

$$\mathrm{TotalGain} = \sum \mathrm{VPGain} - \sum \mathrm{VPPenalty} \quad (2)$$

となる。よって、

$$\mathrm{TotalGain} > 0 \quad (3)$$

を満たす、静的な値予測命令について予測を行なえば、予測値を使用する最初の命令の全ての実行を早めることができることになる。

式 3を満たすような静的な値予測命令を選択するという条件を条件 1 とする。条件 1 を満たす静的な値予測命令は、予測による利益が全くない命令や、値予測のミスによるペナルティーの大きな命令を除外したものとなる。

4.3.2 条件 2: 効果の少ない命令を除外

値予測を行なっても意味のない命令は、前述の条件 1 で選択されない命令の他にも存在する。例えば、ある静的命令を実行する場合を考える。すると、ある時には、その命令を実行することにより

$$\mathrm{VPDestLatency} > \mathrm{VPDataReadyLatency} \quad (4)$$

となるが、別のある時には予測値を使用する命令が存在しないことがある。後者の場合は VPGain が 0 となるため、値予測を行なう意味がない。

しかし、大部分の実行においては VPGain が 0 となるが、部分的に VPGain > 0 となる場合が存在するため、式 3を満たす静的命令が存在する。

このような静的命令については、予測を行なうことで速度向上を計ることができるが、予測する命令数と比較して速度向上率が低いので効率は悪くなる。

そこで、このような予測による速度向上効果の少ない命令を除外することを条件 2 とする。

具体的には、以下の式 5を満たす静的命令については

表 2　予測命令選択方式

	予測率 99%	条件 1 効果有選択	条件 2 効果少除外	条件 3 冗長命令除外
方式 A	○	○		
方式 B	○		○	
方式 A'	○	○		○
方式 B'	○		○	○

表 3　ベンチマークの詳細

ベンチマーク	入力パラメータ	総実行命令数
compress	30000 q 2131	125,304,374
go	9 9	159,817,549
li	train.lsp	264,507,762
m88ksim	train.ctl.in	154,496,314

値予測を行なわないようにする。

$$\frac{\sum \mathrm{VPDataReadyLatency}}{\mathrm{VPExecTimes}} < \frac{\sum \mathrm{VPDestLatency}}{\mathrm{VPDestExistTimes}} \quad (5)$$

式 5の左辺は、ある値予測を行なう静的命令を実行する際に、予測を開始してから実際に命令を実行して結果が生成されるまでのレイテンシの平均を表す。

右辺は、予測を開始してから、それぞれの予測値を最初に使用する命令までのレイテンシの平均を表す。

この式により、ある値予測を行なう静的命令を実行した際に、大部分が VPDestLatency > VPDataReadyLatency となり、$\sum$VPGain が少ない場合には式 5を満たさないために除外される。これにより条件 1 よりも予測命令数を減らすことができる。

4.3.3　条件 3: 予測値を別の値予測命令が使用する

ある命令を値予測することによって生成された予測値を最初に使用する命令が値予測を行なう命令であった場合、前者の命令について予測をおこなわなくても、後者のみ予測を行なうことで最終的な速度向上効果が得られる場合がある。したがって、条件 1,2 において、予測値を最初に使用する命令として、値予測命令を選ばないようにすることで、さらに予測命令数を削減することができる。これを条件 3 とする。

5.　提案手法の評価

本章では、第 3章で提案した、値予測する命令を選別する際の条件を組み合わせて命令選択方式を作成し、それを適用した際の性能向上率と、値予測する命令の削減割合について評価し、考察を行なう。

5.1　評 価 環 境

5.1.1　予測命令選択方式

実際に命令を選択する方式として、予測率 99% を基準として、表 2のような条件を組み合わせた 4 方法により命令を選択した。

5.1.2　ベンチマークプログラム

評価に用いたベンチマークプログラムは、整数系の演算のベンチマークである SPECint95 の中から、comprss、go、li、m88ksim を選んで使用した。ベンチマークの実行命令数が、1 億～ 2 億命令になるように入力パラメータを調整した。ベンチマーク毎の入力セットと実行命令数を表 3に示す。

5.1.3　コンパイラ

SPECint95 のベンチマークプログラムをコンパイルするために、富士通研究所作成の newcc コンパイラを使用した[8]。これは、SPARC version 7 用のコードを出力するコンパイラであり、多レベルの中間言語を使用しているため、最適化を行なう際に、それに適した中間言語の上で処理を行なうことができるという特徴を持つ。

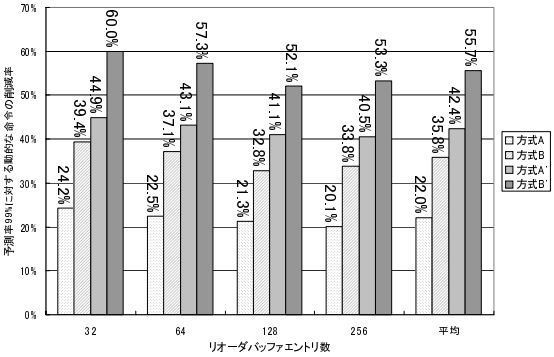

図 5　削減命令数の比率 (全体の平均)

5.1.4　プロセッサモデル

値予測の評価環境として、以下のような構成のプロセッサモデルを使用した。

- ロード命令のレイテンシは 2 サイクル、他の命令は 1 サイクル。
- リオーダバッファのエントリ数以外のハードウェア資源は無限にある。
- 命令セットは、 newcc が出力するコードである SPARC version 7 を用いる。
- 分岐予測の予測精度は 100% である。
- ロード・ストアのアドレスは、パイプラインの早い段階で正確に予測され、依存関係のないロード・ストアはアウトオブオーダ実行が可能である。
- 値予測機構として、 Last-value 予測機構と、ストライド予測機構を使用する。
- 予測ミスが起きた場合は、予測開始状態に戻すのに 5 サイクルかかる。

5.2　結　　果

5.2.1　削減命令数の比較

方式 A、 B、 A'、 B' において、予測率 99% 方式と比較して動的な予測命令数がどの程度削減されたのかを図 5に示す。

図からわかるように、リオーダバッファエントリ数が変化しても、それぞれの方式による削減命令の割合はほぼ等しくなっている。平均すると、方式 A で 22%、方式 B' では 56% と半分以上の動的な予測命令を削減していることがわかる。

5.2.2　同時予測命令数の比較

予測率 99% 方式と、方式 A、 B、 A'、 B' において、1 サイクルに同時に値予測を行なう命令数の平均がどのように変化したかについてを、図 6に示す。

図からわかるように、予測率 99% 方式と比較して、方式 A は 0.5 ～ 1.4 命令、方式 B' は 1.4 ～ 4.0 命令程度減少している。また、リオーダバッファエントリ数が 32 のときの方式 B' では予測命令数の平均が 1 を下回っている。つまり、平均して 1 サイクルに 0.9 命令の値予測を行なうことになるため、予測に必要なハードウェア機構が少なくてすむことになる。また、値履歴テーブル

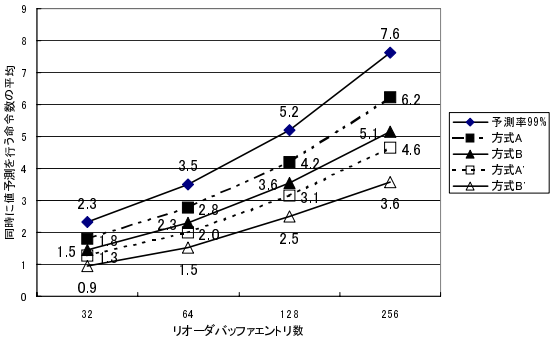

図 6 平均同時予測回数 (全体の平均)

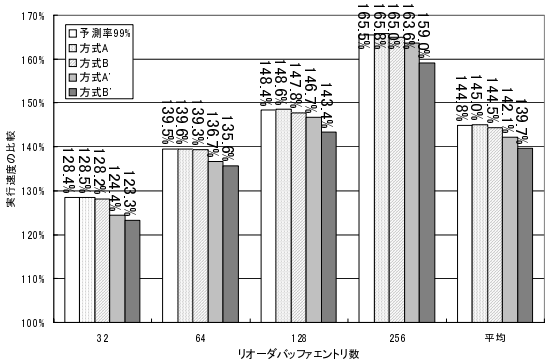

図 7 実行速度の比率 (全体の平均)

の競合が起きにくいという利点もある。

5.2.3 速度向上率の比較

それぞれの値予測命令選択方式を用いた時に、値予測を用いない場合と比較した場合の速度の比率を図 7に示す。図の 100% は、全く予測を行なわなかった時の実行速度となっている。

図から、方式 A では予測率 99% のものよりも若干速度が向上していることがわかる。また、方式 B' では予測率 99% のものよりも最大で 6% の程度速度低下になっているが、予測率 99% と比較しても遜色ない速度向上率になっている。

方式 B' では予測命令数を平均で 56% 削減しているため、予測命令数を削減することにより得られるメリットを考えるとこの速度低下は容認できるものであると思われる。

5.3 考　　察

予測率 99% の方式と比較して、方式 B' による速度向上率が若干劣るのは以下のような理由による。

条件 3 では、予測値を使用する命令が値予測命令である場合は、後者の命令が存在しないものとして扱っていた。従って、後者の値予測による性能向上が得られない場合に、双方とも値予測を行なわないように設定されてしまうことがある。このような場合には、前者の値予測を行なうように方式 B' を改良すれば、後者の予測を行なわない予測命令の実行を早めることができる。

このような条件を探すためには、グローバルにデータ依存関係を考慮しなければならない。特に、値予測を行なうとデータの依存関係が解決するのが早くなるため、データ依存関係が変化する。動的にこのような条件を判別するのは難しいため、コンパイラ等によって静的に依存関係を解析して予測命令を決定することが考えられる。これにより、方式 B' のように予測命令数を大幅に削減しつつ、方式 A のように予測率 99% の方式と同等の速度向上率を得ることが可能になると思われる。

6. ま と め

本研究では、従来の値予測を行なう際の問題点を指摘した。予測率の高い命令の中には性能向上に寄与しない命令も存在するため、そのような命令を予測対象から外し、総予測命令数を削減する手法を提案した。評価を行なった結果、予測率 99% の方式と比較して、方式 A を適用した場合は予測命令数を 22% 削減でき、予測率 99% よりも若干上回る 45% の速度向上が得られた。また、方式 B' を適用した場合は予測命令数の半分以上である 56% が削減可能となり、 40% の速度向上率が得られた。さらに、コンパイラ等によって静的にデータ依存関係を解析しながら予測命令数を選択することで方式 B' のように大幅に予測命令数を削減しながら、方式 A のように予測率 99% と同等の速度向上を得ることができる可能性があることを示した。

謝　　辞

本研究をすすめるにあたり、富士通研究所の木村康則氏から御助言を頂きました。深く感謝いたします。

参 考 文 献

1) Brad Calder, Gleen Reinman, and Dean M. Tullsen. Selective Value Prediction. *ISCA*, 1999.
2) Freddy Gabbay and Avi Mendelson. Can Program profiling support value prediction? *MICRO-30*, pp. 270–280, December 1997.
3) Mikko H. Lipasti and John Paul Shen. Exceeding the dataflow limit via value prediction. In *MICRO-29*, pp. 226–237, December 1996.
4) Mikko H. Lipasti, Christopher B. Wilkerson, and John Paul Shen. Value locality and load value prediction. In *ASPLOS VII*, pp. 138–147, Octover 1996.
5) Kai Wand and Manoj Franklin. Highly accurate data value prediction using hybrid predictors. *MICRO-30*, pp. 281–290, December 1997.
6) Chao ying Fu, Matthew D. Jennings, Sergei Y. Larin, and Thomas M. Conte. Highly accurate data value prediction using hybrid predictors. *ASPLOS VIII*, pp. 281–290, December 1998.
7) 吉瀬謙二, 坂井修一, 田中英彦. マルチレベル・ストライド値予測機構による命令レベル並列性の向上. *JSPP*, pp. 119–126, Jun 1999.
8) 飯塚大介, 小沢年弘, 坂井修一, 田中英彦. C コンパイラにおけるループ最適化の検討. 情報処理学会研究会報告 99-HPC-77, pp. 65–70, Aug 1999.