値予測の軽量効率化方式の提案と評価

飯塚大介[†,☆] バルリ ニコ・デムス[††]
坂井修一[††] 田中英彦[††]

値予測はデータ依存関係を投機的に解消することで並列度を抽出する手法である．これまでに予測精度を高めるための種々の方式が提案されているが，そのために必要なハードウェア量とレイテンシが大きく実装が困難なため，値予測は現実のものとなっていない．なかでも予測履歴テーブルは大きなハードウェアを必要とするためこの問題が顕著である．テーブルのタグや予測値のビット数，エントリ数を制限してハードウェア量を削減すればハードウエア量などの問題はなくなるが，これにともなう容量性ミスによる性能向上率の低下が問題となる．本論文では，値予測を現実のプロセッサに適用するための技術として，静的に命令を分類し，予測候補から除外する命令を選択することで総予測命令数を削減し，値予測履歴テーブルのエントリ数を削減しても性能が低下しないですむ方式を提案し，初期的な評価を行う．

A Light-weight Efficient Value Prediction Mechanism

Daisuke Iizuka,[†,☆] Niko Demus Barli,[††] Shuichi Sakai[††]
and Hidehiko Tanaka[††]

Value prediction is a technique to extract parallelism by predicting values and speculatively resolving true data dependencies among instructions. Various value prediction mechanisms have been proposed in the past and many of them employ sophisticated methods to achieve higher prediction accuracy. However, these methods are still not considered sufficiently feasible since they require large amount of hardware and impose high access latency, especially for Value History Table (VHT). This paper proposes a mechanism to reduce the requirements for VHT entries, by statically classifying instructions that are unlikely useful for prediction and exclude them from prediction targets. Using the mechanism, it is shown that we can achieve similar prediction benefits with smaller size of VHT.

1. はじめに

アウトオブオーダ実行を行うスーパ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，プログラム中にはデータ依存（真の依存）が存在するため，従来の並列度の抽出技法だけでは性能向上に限界がある．近年，この真の依存関係を解決するための投機手法として，命令が生成する値を予測する値予測が考案されている[6]．値予測は，予測精度が高く予測可能な命令が多いほど速度向上が得られる．値予測で予測可能な命令数を増やし，予測精度を向上させるために，種々の複雑な予測機構が提案されている[7]．また，値履歴テーブル（Value History Table，VHT）のサイズを大きくすると VHT にヒットしやすくなるため，予測可能命令数を増加させることができる．このような方法によって高精度で予測を行うことができるが，予測機構を実装するためのハードウェア量も膨大なものになり，また予測値を取り出すまでのレイテンシも大きくなる．本論文では，値予測の実装可能性を高めるために，一番単純な予測機構である LastValue 予測のみを用い，予測対象外となる命令を静的に選択しておくことで少ない VHT のエントリ数によるデメリットを軽減し，これによって小ハードウェアかつ低レイテンシで動作する値予測方式を提案し，評価する．

† 東京大学大学院工学系研究科
Graduate School of Engineering, The University of Tokyo
†† 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
☆ 現在，日立製作所中央研究所
Presently with Hitachi, Ltd., Central Research Laboratory

2. 値予測機構のアクセスレイテンシ

本章では，近未来のプロセッサに値予測機構を実装する場合を考える．具体的には，2001 年の SIA のロードマップ[3] より，2004 年に製作されると考えられる

90nm プロセス，4 GHz で動作する 4way のスーパスカラプロセッサに値予測機構を搭載する場合について考える．

2.1 値予測機構

値予測は命令の演算結果を演算前に予測することで，真のデータ依存関係を超えて投機的に命令を実行し実行速度を向上させる手法である．値予測の機構としては，最も近い過去に得られた値を予測値とする Last-Value 予測機構[6]，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 から今回の計算値を Value+Stride として予測するストライド値予測機構[12]，過去の連続した有限個の計算値を保存し，同一のコンテキストが繰り返されると仮定して予測を行うコンテキスト・ベース値予測機構[8] などがある．いずれの予測機構でも，過去の演算結果を VHT に保存し，次回同一 PC の命令をフェッチした際に，VHT から予測値を取得し予測を行う．予測値が本来の実行値と異なっていた場合（予測ミス時）は，それに依存した命令をすべて再実行する必要があり，実行時間にペナルティーが生じる．そのため，VHT の各エントリに確信度として飽和カウンタを設け，予測値がヒット/ミスした際にカウンタを増加/減少させ，ある閾値を超えた場合にのみ予測を行うのが普通である[6]．

2.2 値予測機構のアクセスレイテンシ

文献 1) では，プログラムを実行した際に，ある命令が発行されてから，その命令が生成した値を最初に使用しようとする命令が現れるまでのサイクル数を統計的に求めている．その結果，ロード命令の 78〜94%，整数命令の 73〜99%により生成される値については，生成されてから 1 サイクル以内に使用されていることが分かった．値予測を用いてこの依存関係を断ち切り速度向上を得るには，前者の命令の実行完了前に予測値を取り出さなければならない．同じ文献 1) によると整数命令の 13〜51%がリネームしてから 3 サイクル以内に実行を完了している．値予測機構から予測値を取り出すまでのアクセスレイテンシを小さくししないと予測値を取り出す前に実行が完了してしまう．

アクセスレイテンシを小さくするには，低レイテンシな予測方式を用いて，エントリ数，ポート数，連想度，タグビット幅を小さくすればよい．そのようにすると容量性ミスが発生しやすくなる．アクセスレイテンシを小さくしたまま容量性ミスを削減するためには予測成功率の低い命令の実行結果を VHT に反映させないようにすることが考えられる．後者については 3 章で述べ，2 章では前者のアクセスレイテンシを小さくする方法について考える．

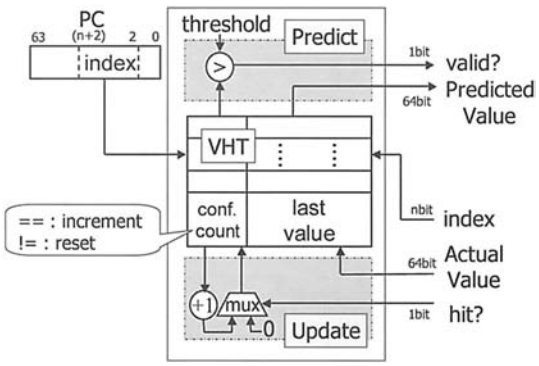

図 1 LastValue 予測機構
Fig. 1 LastValue predictor.

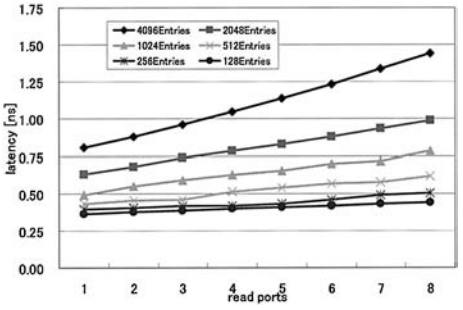

図 2 読み込みポート数とエントリ数による VHT アクセスレイテンシ（wports=4）
Fig. 2 VHT access latency (varying read ports and entries).

2.3 LastValue 予測機構のアクセスレイテンシ

一番単純な予測機構である**図 1** のような LastValue 予測機構を考える．読み込みポート数は同時に予測を行う命令の数だけ必要となる．リタイア時に VHT の更新を行うとすると，LastValue の書き込みポート数はリタイア幅と同じだけ必要となる．リタイア幅を 4 として，この予測機構の VHT のエントリ数と同時予測命令数（読み込みポート数）を変化させたときの VHT アクセスレイテンシを**図 2** に示す．これは CACTI3.0[10] を用いて計算した．計算に用いたパラメータは，1 エントリが 8 バイト，タグなしでダイレクトマップ，最速になるようにサブバンクに分割，ポートはすべて読み書きが独立，90 nm のプロセス，電圧 1.0 V とした．なお，確信度の値に関しては読み込みポート数が同時予測命令数とリタイア幅の合計分だけ必要となるが，4 ビット程度であれば予測機構全体へのレイテンシへの影響は小さいと考えられるため無視できるものとする．

図 1 の予測機構では，命令とともに予測値をリザベーションステーション（RS）に格納し，予測結果の

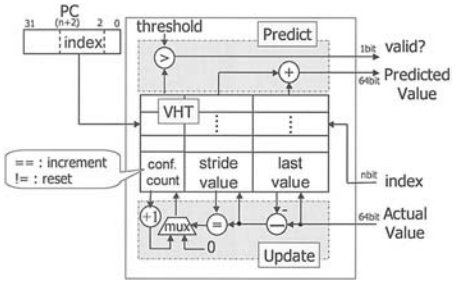

図 3 ストライド値予測機構
Fig. 3 Stride value predictor.

検証に利用するものとする．

2.4 ストライド値予測機構のレイテンシ

LastValue 予測機構の次に単純であるストライド値予測機構を**図 3** に示す．図 3 のストライド予測機構では，VHT 更新時にストライド値を計算するために，読み込みポート数が同時予測命令数に加えてリタイア幅の分だけ余計に必要となる．リタイア幅 4，同時予測命令数が 4 の場合は 8 ポートになる．図 2 より，同時予測命令数，リタイア幅が同じ条件の下では LastValue 予測機構よりもレイテンシの面で不利になる．

読み込みポート数の増加を抑えるために，予測値と LastValue をともに RS に入れておき，命令の演算終了後ただちに検証とストライドの計算を行うようにすれば，図 1 と同様に読み込みポート数を同時予測命令数と等しくできる．しかし 2 つの 64bit の値を RS 内に保持しなければならないために，1 つの予測値を RS に格納するだけで済む 2.3 節の方式よりも RS のサイズが大きくなる．近年の高速に動作するプロセッサでは値の代わりに物理レジスタ番号を RS に格納する方式を採用しており，2 つの 64bit の値を RS で保持すると，ソースレジスタの値を 2 つ格納する従来型の RS よりも規模が大きくなる．よって 2.3 節の場合よりも高速動作が困難になる．

また，パイプライン長が長い場合，ある命令がリタイアされる前に同一 PC の命令がフェッチされる可能性が高くなる．当該命令が誘導変数の場合，VHT が最新の状態に更新されないままストライド値予測を行うため，予測ミスが発生する．これを防ぐには分岐予測と同様にフェッチ時に投機的に VHT を更新することが考えられるが，書き込みポート数が同時予測命令数の分だけ余分に必要となりレイテンシが増加する．さらに，ストライド値予測機構では，予測値を得るための加算器のレイテンシ，ストライドを計算するための減算器のレイテンシ，VHT の各エントリにストライド値を保持するためのレイテンシが余分にかかり，LastValue 予測機構と比較すると不利になる．ハイブリッド予測機構やコンテキストベース予測機構はストライド値予測機構よりも複雑なため，さらにアクセスレイテンシが必要となる．

エントリ数が 128 の LastValue 予測機構はレジスタファイルとほぼ同じ大きさになる（1 kB 程度）ためハードウェア量，レイテンシが小さく実装しやすい．よって本論文では LastValue 予測機構のみを用いるものとする．

3. 静的な予測命令選択法の提案

2 章では，低アクセスレイテンシにするため，エントリ数を小さく，タグを使用しない VHT について述べた．このような VHT では容量性ミスが起きやすい．そこで本章では静的に命令の種類やデータフローを解析し，予測対象命令を限定することで容量性ミスが生じる可能性を低下させ，VHT を単純化した際に生じる性能低下を抑える方式を提案する．実行バイナリ中の各命令には予測を行うかどうか判断するため 1 ビットの付加情報がついているものとし，静的解析結果によりこのビットの値を確定する．

3.1 命令の静的分類

静的にプログラム中のデータフローを解析し，命令を以下に示すクラスに分類する．クラス分けは以下に述べる順に適用され，一度特定のクラスにクラス分けされた命令を他のクラスに再分類することはないものとする．

BRONLY 予測値が条件分岐命令またはレジスタ間接分岐命令にのみ使用される命令

RETADDR サブルーチン呼び出し後のリターンアドレスを生成する命令

GPSET グローバルポインタのセット命令

INDUC 誘導変数

MULSRC 予測命令が使用する値のうち少なくとも 1 つが，複数の命令により生成される命令

プログラムを実行した際に，値を生成する全実行命令中で，それぞれのクラスに所属する命令の占める割合を**図 4** に示す．図 4 で，同一 PC を持つものを 1 つとして数えたもの（静的命令）を**図 5** に示す．また，VHT エントリが無限大の場合に，プロファイルを用いて LastValue 値予測で 99%以上ヒットする命令を調べ，それらをクラス分けした．値を生成する全実行命令に対して 99%以上ヒットする命令がどの程度の割合を占めるかを**図 6** に示す．図 6 で 99%以上ヒットする命令のうち，前述したどのクラスにも属さないものは OTHERVPHIT として示してある．命令を静的

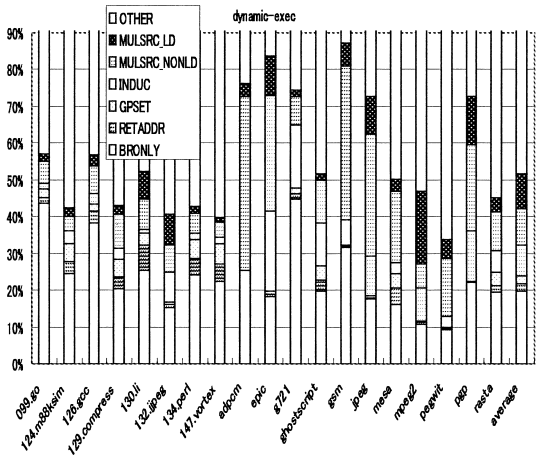

図 4 値を生成する実行命令の分類
Fig. 4 Classification of value-generated instructions (dynamic).

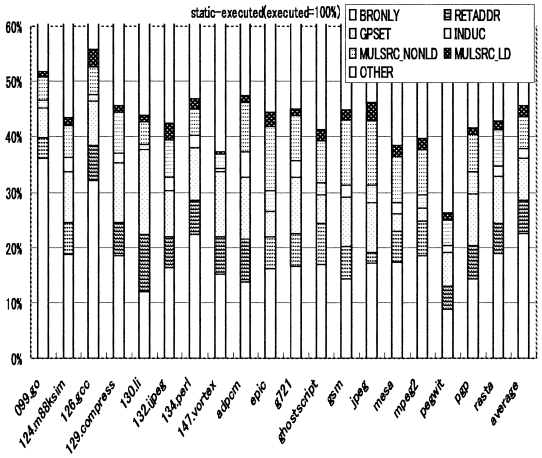

図 5 実行された静的命令の分類
Fig. 5 Classification of executed instructions (static).

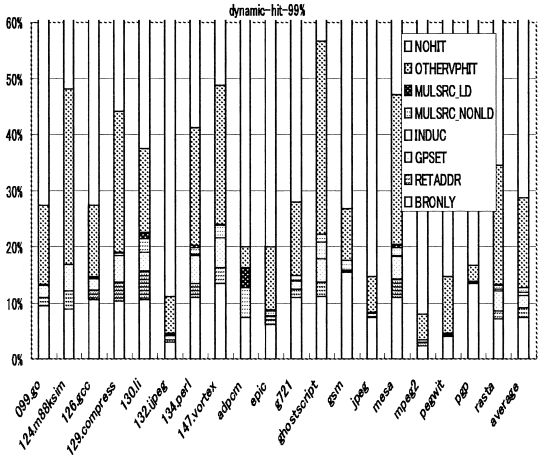

図 6 99% LastValue 予測ヒット実行命令の分類
Fig. 6 Classification of 99% LastValue hit instructions.

```
A  addq  $1,1,$2
B  stq   $3,0($2)
C  and   $2,0xff,$4
D  cmple $4,2,$5
E  bne   $5,L1
```

図 7 BRONLY コード例
Fig. 7 Example code of BRONLY.

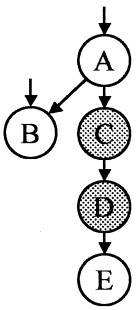

図 8 図 7 のデータフローグラフ
Fig. 8 Data flow graph of Fig. 7.

にこれらのクラスに分類した際に，特定のクラスの命令について LastValue 予測を行わないようにすることで，VHT の容量性ミスによる性能向上率の低下を抑制する．以下にこれらのクラスの詳細と，予測しないようにする理由を示す．

3.2 BRONLY：分岐命令にのみ使用される値

図 7 のコードを検討する．そのデータフローグラフは**図 8** のようになる．このコードをパイプライン処理し処理速度を向上させるには，高精度の分岐予測機構が必要となる．条件分岐は 1 ビットの値予測機構に相当し，予測精度の高い種々の機構が提案，実装されている[9]．

図 8 で，条件分岐命令 E の分岐予測が高精度でヒットする場合，プロセッサの発行幅が広く，ユニット数が多ければ命令 C，D の値予測は行わなくても性能は変わらないと予想される．よって，命令 C，D のように，命令の演算結果が条件分岐命令にのみ使用されている場合，当該命令は値予測を行わないようにする．C 言語の switch〜case や関数ポインタを使用した関数呼び出しによって生成されるレジスタ間接分岐命令に関しても同様のことがいえると予想される．したがって，レジスタ間接分岐命令にのみ使用される値を生成する命令は値予測対象から外す．

3.3 GPSET：グローバルポインタ

64 ビットプロセッサ上では，64 ビットの定数をセットする場合に汎用レジスタの 1 つをグローバルポインタ（GP）として使用する場合が多い．GP を使用せず

```
A  ldah t1,-1(gp)
B  ldq  t12,26880(t1)
C  ldq  t12,-32192(gp)
D  jsr  ra,(t12),12018fd10 <subroutine>
E  ldah gp,8190(ra)
F  lda  gp,-16988(gp)
```

図 9 呼び出し元関数例
Fig. 9 Example code of caller function.

に 64 ビット定数をセットする場合，複数の定数セット命令やシフト命令を必要とするが，メモリ上に 64 ビット定数格納領域を設け，GP がこのアドレスの近傍を指すようにすれば，ロード 1 命令だけで値をセットすることができるからである．関数のアドレスや文字列へのポインタや，C 言語の global や static な変数にアクセスする際のポインタを生成する際にも GP は使用される．**図 9** では，命令 A により global 変数へのポインタを取得し，命令 C により subroutine 関数のアドレスを取得している．

このように，GP は定数やアドレスが固定されているポインタを生成する命令の元となる場合が多い．GP 自身は同じコンパイル単位ではつねに同じアドレスを指している．そのため，GP を生成する命令は LastValue 値予測のヒット率が高い．しかし，前述したように GP の値を使用する命令自身は定数や固定したアドレスを生成する命令であるので，これらも LastValue 値予測のヒット率が高い命令である．したがって，GP の値を使用する命令を予測すれば，GP 自身を生成する命令を予測しなくても性能低下は起きないと考えられる．

3.4 RETADDR：リターンアドレス

リターンアドレスはサブルーチン呼び出し命令（図 9 の命令 D）や，呼び出し元に戻る直前にスタック上に退避してあったリターンアドレスを読み込む（**図 10** の命令 K）際に生成または取得される．そして，呼び出し先関数でスタック上に退避されるとき（図 10 の命令 J），リターン命令で使用される（図 10 の命令 L）．さらに，呼び出し元関数で GP を再セットする際にも使用される（図 9 の命令 E，F☆）．つまり，リターンアドレスは，最終的にはリターン命令と GP セット命令に使用される．リターン命令に使用される場合を考えると，3.2 節で説明した分岐命令のときと同様，リターンアドレス予測のヒット率が高ければリターンアドレス自身を値予測する必要はない．GP に使用される場合を考えると，3.3 節で説明したように，GP の値によって生成される値を予測するのであれば，GP を生成するのに必要なリターンアドレス自身を値予測する必要はない．したがって，リターンアドレスを生成する命令は値予測を行わないようにする．

☆ Compaq Alpha の場合

```
   <subroutine>:
G  ldah gp,8190(t12)
H  lda  gp,-16496(gp)
I  lda  sp,-112(sp)
J  stq  ra,0(sp)
   ...
K  ldq  ra,0(sp)
L  ret  zero,(ra),0x1
```

図 10 呼び出し先関数例
Fig. 10 Example code of callee function.

```
A  bne   $1,L1
B  mov   $2,$4
C  br  L2
   L1:
D  mov   $3,$4
   L2:
E  subq  $4,1,$5
F  and   $5,0xff,$6
```

図 11 MULSRC コード例
Fig. 11 Example code of MULSRC.

3.5 INDUC：誘導変数

誘導変数（Induction Variable）はループ中で一定の値が加減される変数のことである．誘導変数はストライド値予測機構を使用すれば高精度で予測することができるが，LastValue 予測機構を用いて高精度で予測することは難しい．図 4 からプログラム中では多くの誘導変数生成命令が実行されていることが知られるが，図 6 から分かるように LastValue でヒット率が高い誘導変数は非常に少ない．ここでは，できるだけ単純な予測機構（LastValue 予測機構）によって高い性能をあげることをねらうため，誘導変数は予測対象から外すこととする．

3.6 MULSRC：命令が使用する 1 つの値が複数の命令によって生成される場合

図 11 のコードを検討する．**図 12** に，このコードの制御フロー（破線）とデータフロー（実線）を示す．図 12 において，命令 E が使用する値は命令 B または命令 D が生成する値のどちらかである．どちらの値を使用するのかは条件分岐の分岐先によって決定される．もし分岐先が一定でなく，かつ命令 B と命令 D

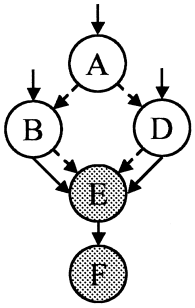

図 12 図 11 の制御・データフローグラフ
Fig. 12 Control/Data flow graph of Fig. 11.

が生成する値が異なっている場合，命令 E の入力値が前回と異なる場合が多くなると考えられる．実際に命令 B と命令 D が同じ値を生成することが静的に分かるのであれば，コンパイル時に最適化により A より前または E より下の 1 命令に集約されるため，命令 B と命令 D は異なる値をとることが多いと考えられる．命令 F に関しても同様のことがいえる．命令 E，F のように，ある命令が使用する 1 つの値が，複数の命令によって生成される可能性のある命令を MULSRC と命名する．

3.6.1 MULSRC がロード命令以外の場合

ロード命令以外で MULSRC に属する命令を MULSRC_NONLD と命名する．MULSRC_NONLD な命令が LastValue 予測機構でヒットしやすいかどうかはプログラムのコンテキストに依存する．しかし，図 4，図 6 から分かるように，MULSRC_NONLD に属する命令は，実行される命令としては 4.3%〜40%と多いものの，LastValue 予測機構でヒットする命令数は実行される命令数と比較すると最大の adpcm で 4%と少ない．これは，MULSRC_NONLD の命令の入力値が毎回異なる確率が高いため，結果として当該命令が生成する値も毎回異なる値になる確率が高いためであると考えられる．そのため，これらの命令を予測しないようにすることは VHT の汚染を防ぐには有効であると考えられる．

3.6.2 MULSRC がロード命令の場合

ロード命令で MULSRC に属する命令を MULSRC_LD と命名する．図 12 でノード E がロード命令の場合，決まった入力値に対して毎回決まった出力をするとは限らない．入力値から計算されるアドレスに格納されている値が前回から変更されている可能性があるからである．逆に，入力値が異なっても，出力される値が等しくなる場合も考えられる．アドレスが異なっていても格納されている値が等しい場合もあるからである．このように，ロード命令で，かつ複数の命令が生成した値を使用する可能性のある命令を MULSRC_LD に所属するものとする．MULSRC_LD な命令に関しても MULSRC_NONLD と同様に，実行命令よりも LastValue 予測機構でヒットする命令数が少ないものが多い．そのため，このクラスに属する命令も値予測を行わない候補として使用する価値があると思われる．ロード命令は実行レイテンシが大きいため値予測がヒットした際の利得は大きい．そのため，MULSRC_NONLD よりも MULSRC_LD の方は値予測除外候補としては優先度を低くする．

3.7 静的解析法

各命令のクラス分けは，静的リンクされたバイナリを逆アセンブルして制御フロー，データフローを解析することにより行う．これによりソースコードが手に入らないバイナリや，libc の関数内などについても解析を行うことができる．本解析ではソースコードはいっさい用いていない．解析は関数単位で行う．解析の流れは以下のとおりである．

(1) バイナリを逆アセンブルする．
(2) table jump（C 言語の switch〜case 文に相当）の飛び先を jump table を読んで決定する．
(3) 制御フローを解析し，基本ブロックに分割する．
(4) 基本ブロックごとにレジスタの Use-Def を求める．
(5) Reaching Definition [17] 解析をし，各命令間のレジスタの Use-Def を調べる．これにより関数内のレジスタを介したデータフローグラフが完成する．
(6) jmp 命令や条件分岐命令から Use-Def chain をたどり，分岐命令にのみ値が使用されている命令を探し BRONLY とする．BRONLY に属した命令からも同様のことを行う．これを収束するまで繰り返す．
(7) 自分が生成した値を自分自身で使用している場合，それを INDUC とする．
(8) GPSET，RETADDR を探す．
(9) MULSRC を探す．MULSRC を使っている命令も MULSRC にする．

3.8 予測除外命令選択法

本節では，3.2〜3.6 節で説明した命令のうち，どの命令を選択的に予測対象から外すのかについて述べる．まず，BRONLY，RETADDR，GPSET，INDUC に所属する命令は，予測対象から外す．この予測除外命令選択法を **BRONLY+**とする．さらに，BRONLY+に加えて MULSRC_NONLD を予測対象から外したものを **MULSRC-**とする．MULSRC-に加えて MUL-

表 1 予測除外命令選択法
Table 1 Choosing method of prediction exclusion instruction.

BRONLY+	GPSET INDUC BRONLY RETADDR
MULSRC-	GPSET INDUC BRONLY RETADDR MULSRC_NONLD
MULSRC+	GPSET INDUC BRONLY RETADDR MULSRC_NONLD MULSRC_LD

SRC_LD をさらに予測対象から外したものを **MULSRC+**とする．以上の 3 つの予測除外命令選択法を**表 1** に示す．

図 4，5 を見ることで，これらの選択法により予測対象外となる動的/静的命令の比率を知ることができる．なお，図 5 は実行バイナリ全体ではなく，実行バイナリのうち実際に実行された部分を 100%として比率を求めている．

以降，これまでに述べた提案手法を評価する方法について説明する．

3.9 プロセッサモデル

図 13 にベースラインとなるプロセッサのパイプラインを示す．各パラメータを**表 2** に示す．90 nm 4 GHz ではレジスタのアクセスレイテンシが 2 サイクルになる．

値予測の動作は以下のとおりである．フェッチした 4 命令のそれぞれについて，静的に追加された追加ビットを調べる．追加ビットにより予測対象外となっている命令については予測を行わず，VHT の汚染防止のためリタイア時に VHT への更新も行わないものとする．

予測対象となっていた場合は LastValue 予測機構にアクセスし，2 サイクル経過すると確信度と予測値が出力される．確信度が閾値以上であればリネームユニットからレジスタ情報を受け取り，レジスタファイルの該当エントリに予測値を書き込む．図 13 の下にあるレジスタファイルは上にあるものと実体は同じであり，予測値を書き込むタイミングを示すために下にも書いてある．2 サイクル後に書き込みが完了し，予測値は通常の値と同様に使用される．予測値をレジスタに書き込むのと同時に，予測対象命令が入っているリザベーションステーションにも予測値を書き込む．当該命令がディスパッチされると同時に予測値をパイプライン上に流し，実際の値が生成された際にすぐに比較できるようにする．予測値が異なっていたら分岐予測ミス発生時と同様，リタイア時にパイプラインをフラッシュし，状態を巻き戻す．リタイア時に実行結果を VHT に更新する．

図 2 より，クロック周波数 4 GHz の場合，VHT エントリが 128，256 のときには 2 サイクルで予測値が取り出せ，図 13 のパイプライン構成に合うためこれらのエントリ数で評価を行う．512 以上のエントリ数では VHT からレジスタに格納した予測値を取り出したときに正しい実行結果が出ている可能性があるため本論文では評価を行わない．

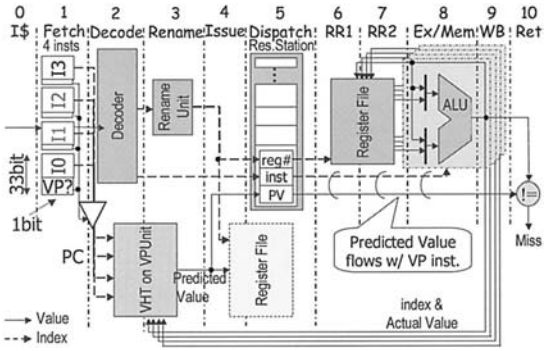

図 13 ベースラインプロセッサのパイプライン
Fig. 13 Pipeline of baseline processor.

表 2 プロセッサモデル
Table 2 Processor model.

ISA	Compaq Alpha compatible
Inst. Latency	Load 1 (for addr. calc), Other 1
Phy. Reg	64bit × 80
ROB	64 entries
Pipeline	11 stages
Fetch Decode Issue Retire	Max 4 insts/cycle
Retire width	4
Res. Station	20 entries
LSQ	20 entries
EU	ALU x 4, LSU x 2
BTB	1024 entries, 2 way
Branch Pred.	gshare, 4 k entries
Ret. Addr. Stack	20 entries
Icache	Block 16 B, 1 k entries, 2 way, latency 2
DcacheL1	Block 64 B, 256 entries, 2 way, latency 2
DcacheL2	infinite size, latency 6
VHT	No tag, read 8 ports, write 4 ports, 5bit counter, 128/256 entries

3.10 評価環境

評価には，トレースベースのシミュレータを使用した．ベンチマークは SPECint95 の 8 つのプログラムと，Mediabench[4] の 11 個のプログラムを使用した．ベンチマークは Alpha21264 のコードを出力する GNU GCC（バージョン 2.95.2）に，最適化オプション-O3 を付けてコンパイルし，ライブラリを静的リンクしたものを使用した．あらかじめこの実行ファイルについて静的にデータフロー解析を行い，予測対象外となる命令を探し，追加ビットに反映する．SPECint95

に関しては**表 3** にあるように，100〜200 M 命令程度で終了するように実行時の入力パラメータを調整した．Mediabench は全命令実行した．

表 3 ベンチマークの詳細
Table 3 Detail of benchmarks.

ベンチマーク	内容	実行命令数
099.go	囲碁	143 M
124.m88ksim	プロセッサシミュレータ	147 M
126.gcc	C コンパイラ	142 M
129.compress	ファイル圧縮	127 M
130.li	lisp 処理系	217 M
132.ijpeg	画像不可逆圧縮，展開	127 M
134.perl	スクリプト言語	129 M
147.vortex	データベース	120 M
adpcm	音声可逆圧縮，展開	12 M
epic	画像可逆圧縮，展開	73 M
g721	音声不可逆圧縮，展開	593 M
ghostscript	postscript エンジン	1017 M
gsm	音声不可逆圧縮，展開	287 M
jpeg	画像不可逆圧縮，展開	287 M
mesa	3D 描画ライブラリ	225 M
mpeg2	動画不可逆圧縮，展開	1542 M
pegwit	暗号，復号	67 M
pgp	暗号，復号	126 M
rasta	音声認識	28 M

4. 評　　価

本章では，提案方式の値予測ヒット率，ヒット命令数，および速度向上率について評価を行う．3 章で提案した予測命令選択法に基づき，あらかじめ静的に予測対象外命令を選択し追加ビットを修正しておく．

4.1 値予測ヒット率とヒット命令数

VHTEntry=128 の場合において，本論文の手法を適用した際の値予測ヒット率の変化を**図 14** に，値予測でヒットした動的な命令数を**図 15** に示す．図 15 は対数軸になっている．normalvp が通常の値予測である．

19 中 14 のプログラムでヒット率が向上した．図 14 より，BRONLY+，MULSRC-，MULSRC+と予測命令を制限する度にヒット率が上がるプログラムがある（go，m88ksim，gcc，compress，epic，mpeg2，pegwit）．ijpeg，vortex，jpeg，pgp では予測を行わなかった場合よりもすべての場合においてヒット率が低下しているが，それ以外のものは BRONLY+，MULSRC-，MULSRC+のいずれかを適用することで適用しなかった場合よりもヒット率が上昇している．

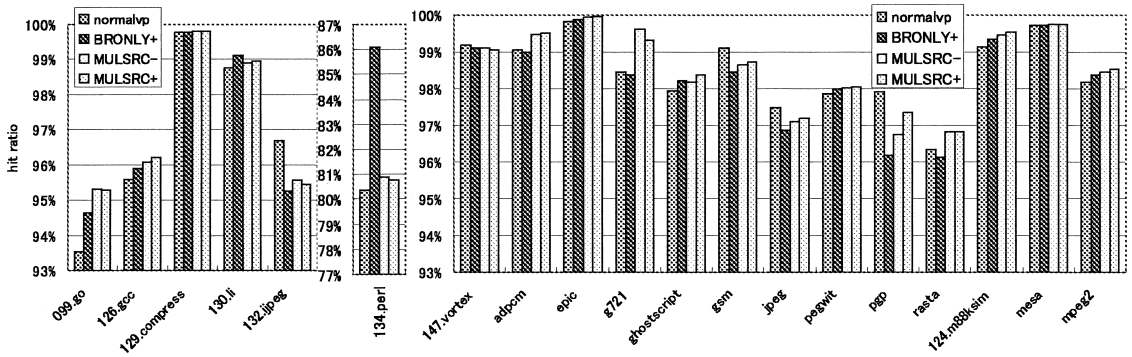

図 14 値予測ヒット率（VHTEntry=128）
Fig. 14 Value prediction hit rate (VHTEntry = 128).

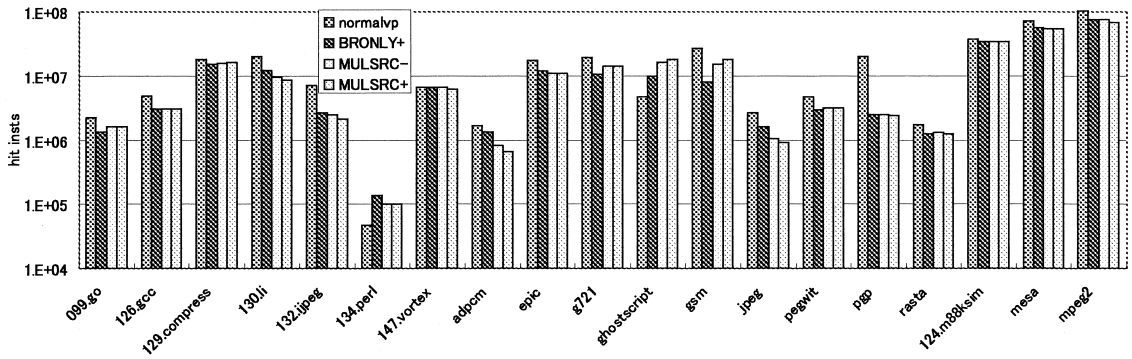

図 15 値予測ヒット命令数（VHTEntry=128）
Fig. 15 Value prediction hit instructions (VHTEntry = 128).

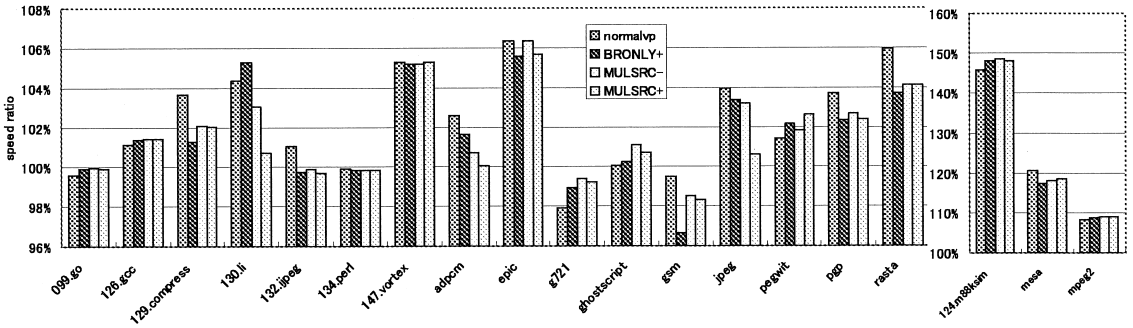

図 16 速度向上率（VHTEntry=128）
Fig. 16 Speedup ratio (VHTEntry =128).

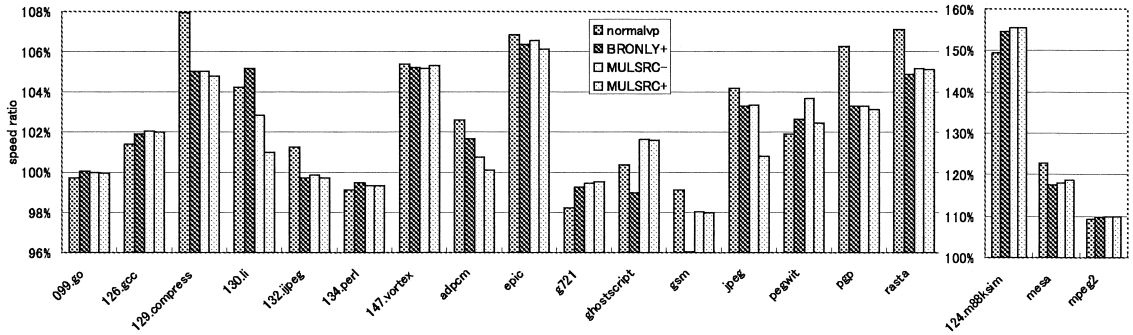

図 17 速度向上率（VHTEntry=256）
Fig. 17 Speedup ratio (VHTEntry = 256).

図 15 より，予測命令を制限することで，perl, ghostscript 以外のものは予測ヒット命令数が減少している．ghostscript で BRONLY+，MULSRC-, MULSRC+と予測命令を制限するたびにヒット命令数が増大しているのは，本来は高精度で予測可能であるが確信度カウンタが閾値に達する前に別の命令の内容で上書きされていたものが，命令制限により上書きが生じなくなり閾値に達し，予測を行うようになったためと考えられる．

図 14 より，perl はヒット率が悪い．perl はエントリ数を 1024 以上にすると予測ヒット率，ヒット命令数とも大きく増加するため，本論文の VHT サイズでは容量性ミスが頻発していると考えられる．

4.2 速度向上率

速度向上率を**図 16** と**図 17** に示す．100%が予測を行わないときの実行速度であり，これより高ければ実行速度が向上していることになる．

全体として，値予測を行った場合は行わない場合と比較すると −3.9%から 55.5%の性能向上が得られることが分かる．いずれかの命令選択法を適用することで，go，gcc，li，g721，ghostscript，pegwit，m88ksim，mpeg2 では normalvp よりも性能向上率が最大 6%向上する．ijpeg，perl（VHTEntry=128）は，値予測を行わない場合よりも性能が低下するが，その程度はわずか 0.1%である．gsm は normalvp でエントリ数が無限大のときでも性能向上率が 0.2%しかないことが分かっている．そのため，命令選択によりヒット率とヒット命令数が低下した影響により最大 3.9%の速度低下がおきたと考えられる．しかし MULSRC+では 2%に抑えることができた．残りのプログラムでは normalvp と比較すると劣るが，値予測を行わない場合よりは高速に実行できている．

li の BRONLY+と go，gcc，g721，pegwit は命令選択によりヒット命令数は減少しているがヒット率は向上し normalvp より性能が向上している．これは予測命令を選択することで高ヒット命令のうちいくつかが予測対象外になったが，容量性ミスにより予測ミスを発生させる命令がそれよりも多く予測対象外になったためと考えられる．

以上から，VHT を簡単化してハードウェア量・レイテンシを小さくした際に，通常の値予測を行った場合と比較しても性能が向上する場合があること，予測を行わなかったときよりも性能が向上するものが多いこと，予測を行わない場合と比較して通常実行時より

も性能低下が起きる場合でも，ただ 1 つの例外である gsm を除いては速度低下がごくわずかであることから本方式が有利であることが示された．

5. 関連研究

予測値を取り出すレイテンシを隠蔽する値予測方式として，VHT を用いずに値予測を行う方式[1),5)] が提案されている．文献 1) では，VHT の代わりに Prediction Value Cache（PVC）を用意し，トレースキャッシュ上の命令をフェッチするときと同時に PVC に格納されている予測値をフェッチする．その値を予測に使用すると同時に Prediction Trace Queue（PTQ）にも入れる．PTQ では命令がパイプライン中を流れていくのに合わせて格納値をシフトし，リタイア時の予測値更新に使用する．PVC を用いることで，そのトレースに合った予測値が取得でき，PTQ を使用することで PVC の読み込みポート数を 1 に抑えることができ，低レイテンシで動作可能となる．

文献 11) では当該命令が上書きする論理レジスタに入っている値を予測値とすることで VHT を用いずに値予測を行う手法を提案している．論理レジスタの空きエントリを VHT として利用する形になるため，別に VHT を使用する場合と比較するとエントリ数が少ないという問題がある．

VHT の 1 エントリに必要なハードウェア量を削減することでも低レイテンシを実現できる．文献 14) では，VHT のタグビット幅を削減する手法を，文献 15) では，予測値を 0/1 に限定することでハードウェア量を削減手法を提案している．本論文ではタグは使用せず，予測値は 64 ビットの値を予測しているが，高クロックで動作するように VHT のエントリ数を削減しているためハードウェア量が少なく，かつ低レイテンシで動作するようになっている．文献 2) では動的に予測命令を選択する方式を提案している．クリティカルパスの先頭にある命令を動的に検出し，当該命令の実行結果のみ VHT の更新を行う方式を提案している．また，予測命令や予測値を使用する命令の種類により確信度の閾値を変える方式も提案している．文献 13) では静的に，あるいはプロファイルを用いて，LastValue/Stride/Context のそれぞれの予測機構で予測可能な命令と予測しない命令とに分類する方式を提案している．これにより，VHT のポート数が少ない場合には，動的に予測を行う場合よりも性能向上率が高くなることを示している．本論文では LastValue 予測のみを使用しているため，文献 13) よりもハードウェア量や VHT アクセスレイテンシに関して利点がある．

6. まとめ

本論文では，高クロックで動作するプロセッサで値予測を行う場合において，VHT のアクセスレイテンシを低減させるためにはエントリ数を削減することが必要であることを示し，これにより生じる容量性ミスを低減するため，静的に予測命令を選択する手法を提案した．その結果，小さく単純な VHT を用いて効率良い値予測を行う可能性があることを示した．特に，gcc のように全命令を予測対象とするよりも特定の命令を選択的に予測対象とすることで速度向上率が向上するものがあることが判明した．その反面，compress のように全命令を予測した場合に比べて性能が低下してしまうものもあることが分かった．今後の課題としては，(1) さらに細かく命令をにクラス分けし，どの命令を予測対象とすればよいのかを検討すること，(2) 複数命令間の依存関係を考慮し，予測による速度向上効果の得られない命令も除外するようにする方式についても調査・検討することがあげられる．

参考文献

1) Bhargava, R. and John, L.K.: Latency and Energy Aware Value Prediction for High-Frequency Processors, *16th Annual ACM International Conference on Supercomputing* (*ICS*) (Jun. 2002).
2) Calder, B., Reinman, G. and Tullsen, D.M.: Selective Value Prediction, *26th International Symposium on Computer Architecture* (*ISCA-26*) (May 1999).
3) ITRS: International Technology Roadmap for Semiconductors 2001 Edition (2001). http://public.itrs.net/
4) Lee, C., Potkonjak, M. and Mangione-Smith, W.H.: MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems, *MICRO*, Vol.30 (1997).
5) Lee, S.-J., Wang, Y. and Yew, P.-C.: Decoupled Value Prediction on Trace Processors, *6th International Symposium on High-Performance Computer Architecture* (*HPCA-6*) (Jan. 2000).
6) Lipasti, M.H. and Shen, J.P.: Exceeding the Dataflow Limit via Value Prediction, *Proc. 29th Annual International Symposium on Microarchitecture* (*MICRO-29*) (Dec. 1996).
7) Nakra, T., Gupta, R. and Soffa, M.L.: Global Context-Based Value Prediction, *5th International Symposium on High-Performance Computer Architecture* (*HPCA-5*) (Jan. 1999).

8) Sazeides, Y. and Smith, J.E.: The Predictability of Data Values, *Proc. 30th Annual International Symposium on Microarchitecture* (Dec. 1997).
9) Seznec, A., Felix, S., Krishnan, V. and Sazeides, Y.: Design Tradeoffs for the Alpha EV8 Conditional Branch Predictor, *29th International Symposium on Computer Architecture* (*ISCA-29*) (May 2002).
10) Shivakumar, P. and Jouppi, N.P.: CACTI 3.0: An Integrated Cache Timing, Power, and Area Model, *WRL Research Report 2001/2*, COMPAQ, Western Research Laboratory (Feb. 2001).
11) Tullsen, D.M. and Seng, J.S.: Storageless Value Prediction using Prior Register Values, *26th International Symposium on Computer Architecture* (*ISCA-26*) (May 1999).
12) Wand, K. and Franklin, M.: Highly accurate data value prediction using hybrid predictors, *Proc. 30th Annual International Symposium on Microarchitecture* (*MICRO-30*) (Dec. 1997).
13) Zhao, Q. and Lilja, D.J.: Compiler-Directed Static Classification of Value Locality Behavior, Laboratory for Advanced Research in Computing Technology and Compilers Technical Report No.ARCTiC 00-07, University of Minnesota (Jul. 2000).
14) 佐藤寿倫，有田五次郎：タグビット幅を考慮したデータ値予測機構のハードウエア量削減，信学技報，CPSY2000-3 (Apr. 2000).
15) 佐藤寿倫，有田五次郎：0/1 の局所性を利用したデータ値予測機構のハードウェア量削減，情報処理学会研究会報告，2002-ARC-146 (Feb. 2002).
16) 飯塚大介，角田忠信，坂井修一，田中英彦：値予測における値履歴テーブルエントリ数の削減，情報処理学会研究会報告，2002-ARC-149 (Aug. 2002).
17) 中田育男：コンパイラの構成と最適化，朝倉書店 (1999).

(平成 14 年 9 月 24 日受付)
(平成 15 年 1 月 15 日採録)

飯塚　大介（学生会員）

1975 年生．1998 年東京大学工学部電気工学科卒業．2003 年同大学院工学系研究科情報工学専攻博士課程修了．同年，日立製作所中央研究所に入社．現在に至る．投機実行，プロセッサアーキテクチャの研究に従事．

バルリ　ニコ・デムス

1975 年生．1999 年東京大学工学部電子情報工学科卒業．2001 年同大学院工学系研究科情報工学専攻修士課程修了．現在，同大学院情報理工学系研究科電子情報学専攻博士課程在学中．プロセッサアーキテクチャの研究に従事．

坂井　修一（正会員）

1981 年東京大学理学部情報科学科卒業．1986 年同大学院工学系研究科情報工学専門課程修了．工学博士．同年工業技術院電子技術総合研究所入所．この間 1991 年～1992 年，米国マサチューセッツ工科大学招聘研究員，1993 年～1996 年 RWC 超並列アーキテクチャ研究室室長．1996 年～1998 年筑波大学電子・情報工学系助教授．1998 年東京大学大学院工学系研究科助教授，2001 年より同大学院情報理工学系研究科教授．計算機システム一般，特にアーキテクチャ，並列処理，スケジューリング問題，マルチメディア等の研究に従事．1990 年本会論文賞，1991 年日本 IBM 科学賞，1995 年市村学術賞，ICCD Outstanding Paper Award 等受賞．電子情報通信学会，人工知能学会，IEEE，ACM 各会員．

田中　英彦（正会員）

1965 年東京大学工学部電子工学科卒業．1970 年同大学院工学系研究科博士課程修了．工学博士．同年同大学工学部講師．1971 年同助教授．1987 年同教授．2001 年より同大学院情報理工学系研究科教授・研究科長．この間 1978 年～1979 年米国ニューヨーク市立大学客員教授．計算機アーキテクチャ，並列処理，自然言語処理，メディア処理，分散処理，CAD 等の研究に興味を持っている．著書「非ノイマンコンピュータ」,「情報通信システム」，共著書「計算機アーキテクチャ」,「VLSI コンピュータ I，II」,「ソフトウェア指向アーキテクチャ」．本会フェロー．電子情報通信学会，人工知能学会，日本ソフトウェア科学会，IEEE，ACM 各会員．