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

吉瀬 謙二　坂井 修一　田中 英彦
東京大学大学院 工学系研究科

演算結果を予測することで真の依存関係を解消する値予測の手法を用い、プロセッサで利用できる命令レベル並列性の向上を目指す。値予測の機構として、予測する命令までの制御流の変化を考慮する2レベル・ストライド値予測機構を新たに提案する。更に2レベル・ストライド値予測機構を含む3種類の値予測機構を用いることで予測できる範囲を拡大するとともに、プロファイルを用いて予測ミスの回数を削減するマルチレベル・ストライド値予測機構 (Multi-stride 値予測機構) を提案する。シミュレーションによる評価により、Multi-stride 値予測機構を用いて、全実行命令の21%から51%の命令の演算結果を正しく予測できること、値予測ミスのペナルティを考慮した場合でもリオーダバッファのエントリ数を128と大きく設定することで平均40%の性能向上を達成できることを確認した。

Multi-Level Stride Value Predictor for Increasing ILP

Kenji KISE, Shuichi SAKAI and Hidehiko TANAKA
Graduate School of Engineering, The University of Tokyo

We propose multi-level stride value predictor, which attempts to collapse true-data dependences by predicting the result values. To increase the number of prediction success, multi-level stride value predictor uses three predictors: last-value predictor, stride predictor, and path-based two-level stride predictor. To decrease the number of misprediction, the highly predictable instructions are determined with program traces statically. From the experimental evaluation, we found that multi-level stride value predictor correctly predicts from 21% to 51% of executed results of instructions and that it attains the improvement of 40% in terms of ILP on average taking the misprediction penalty with the reorder buffer of 128 entries into account.

1　はじめに

プログラムに内在する並列性は、制御依存関係とデータ依存関係により制限されることが知られている。制御依存関係を解消するための手法として分岐予測を用いた投機処理があり、精度の高い分岐予測機構の提案は重要な研究課題のひとつになっている。一方のデータ依存関係は、レジスタ数の不足が原因で生じる出力依存関係、逆依存関係と、ハードウェア資源に依存しない真の依存関係に分類できる [3]。この中で、出力依存関係と逆依存関係はレジスタ名前替えで解決できる。残る真の依存関係を解消することは困難と考えられてきたが、近年、真の依存関係を解消する技術として、値予測機構を用いた投機処理の手法が提案された [5, 4, 7, 6]。これは、実際に計算をおこなってデータ値を得る代わりに、生成されるデータの値を予測することで処理を進めておくという投機処理技術である。この値予測機構により、命令レベル並列性の上限を引き上げることが可能となる [2]。

予測により正しい値を得ることができれば真の依存関係を解消できるが、予測に失敗すれば誤った値を利用した命令の再処理が必要となり、プロセッサ性能にペナルティを与えてしまう。幾つかの研究 [2, 4, 7] ではミス率の低減を考慮した値予測機構を提案しているが、これらの値予測機構を用いても、全実行命令に対して2%〜5%程度の頻度で予測ミスが発生する。値予測による性能向上を維持するためにはミス率の低減が重要な課題となっている。

本稿で提案する Multi-stride 値予測機構は、プロファイルを用いて静的に、予測する命令の集合を定義する。これにより、予測ミスを大幅に削減する。更に、予測アルゴリズムを複数用いることで予測できる領域の拡大を試みる。提案する Multi-stride 値予測機構は、Last-value 予測機構 [5, 4]、ストライド値予測機構 [7]、値予測をおこなう命令までの制御流の変化を考慮する2レベル・ストライド値予測機構の3つの値予測機構を利用する。

以下に本稿の構成を示す。次章で、提案されている値予測機構と関連研究をまとめる。3章で2レベル・ストライド値予測機構を提案する。4章では Last-value 予測機構、ストライド値予測機構、2レベル・ストライド値予測機構を利用する Multi-stride 値予測機構を提案する。5章では評価環境をまとめ、6章では評価をおこない、その結果を議論する。7章では本研究でおこなったいくつかの理想化について現実との落差などを議論する。8章で全体のまとめをおこない、今後の課題を述べる。

2 関連研究

研究 [5, 9] ではロード命令に焦点を絞り、ロードするデータ値を予測する値予測機構を検討している。本研究ではロード命令に加えシフト命令と算術論理演算命令の生成する値を予測する。このため本研究で値予測の対象となるのは全実行命令の 60%以上と非常に高い。これまでに発表された、ロード、シフト、算術論理演算を対象とした値予測機構には次のものがある。

2.1 Last-value 予測機構

Last-value 予測機構 [5, 4] は最も近い過去に得られた値を予測値とする。Last-value 予測機構による予測の例を下に示す。上段が生成された値で下段の Hit, Miss が予測の成功または失敗を示している。

```
Value sequence: 1  1   1 3    3 3 5 5 ...
                NP Hit H Miss H H M H ...
```

最初の予測では前回の値が定義されていないので予測はおこなわない。NP (No Predict) は予測がおこなわれないことを意味している。文献 [4] では、予測ミスの回数を削減するために、動的に変化するカウンタからなる分類テーブル (Classification Table) を用いて信頼性の高い命令のみ予測する Last-value 予測機構を提案している。

2.2 ストライド値予測機構

ストライド・ベースの値予測機構は、最も近い過去に得られた 2 回の値の差分 $Stride$ と、最も近い過去に得られた値 $Value$ から、当該命令の予測値を $Value + Stride$ により計算する。ストライド・ベースの値予測機構を図 1に示す。値履歴テーブル (Value History Table, VHT) は前回の演算結果やストライド等を格納するテーブルである。値履歴テーブルの更新の仕方に選択の余地があるが、3 つの状態を用いて予測ミスを削減する文献 [7] のアルゴリズムを説明する。

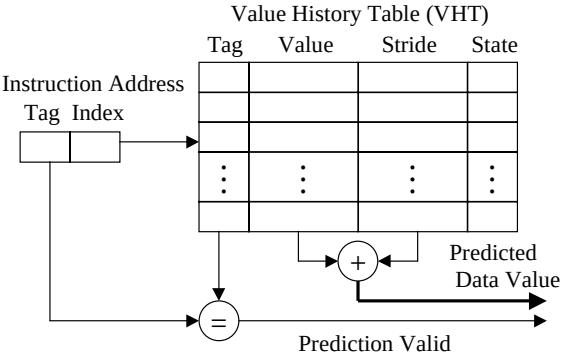

図 1: ストライド値予測機構のブロック図

VHT のエントリは図 1に示す Tag, Value, Stride, State(状態) の 4 つのフィールドを持つ。状態は、Init, Transient, Steady のどれかの値を持つ。

状態の変化と予測アルゴリズムを図 2に示す。最初に値を生成する命令に出会った時には予測をおこなわない。値が生成された時に VHT のエントリが割り当てられ、(i) 結果を Value フィールドに格納し、(ii) エントリの状態を Init に変更する。

状態が Init だった場合には予測をしない。しかし、その命令が値 (D1) を生成した場合にはストライドの開始となった可能性があるので、(i) ストライド (S1) を $S1 = D1 - Value$ より計算し、(ii)D1 を Value フィールドに、S1 を Stride フィールドに格納し、(iii) 状態を Transient に変更する。

状態が Transient の時に命令の別のインスタンスが実行されたときも予測をおこなわない。そのインスタンスが値 (D2) を生成した場合には、(i) ストライド (S2) を $S2 = D2 - Value$ で計算し、(ii) D2 をエントリの Value フィールドに格納し、(iii) もし S2 の値が前のストライドと等しかった場合には、状態を Steady に変更する。ストライドが等しくなければ S2 を Stride フィールドに格納する。

状態が Steady の時には Stride と Value を用いて値を予測する。計算したストライドが前回のストライドと等しい場合には、Value のフィールドを更新する。もし異なったストライドが得られた時には、状態を Transient に変更するとともに、Value, Stride の値を更新する。

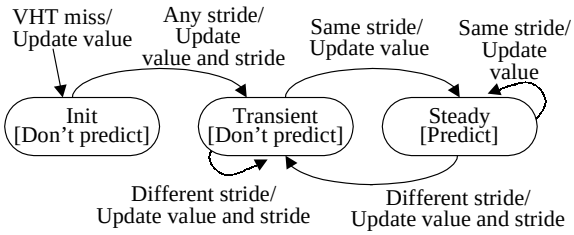

図 2: ストライド値予測における状態遷移

ストライド値予測機構による予測例を下に示す。最初の 3 回分は状態が Steady ではないため予測をおこなわない。

```
Value sequence:1  2  3  4 5 6 7 1 2  3  4 5...
               NP NP NP H H H H M NP NP H H...
```

2.3 その他の値予測機構

文献 [7] では、動的に生成される値の多くは最近のユニークな 4 個以内の値であるという結果に注目し、パターン・ベースの 2 レベル値予測機構を提案している。この 2 レベル値予測機構では、VHT に 4 つの異なった値を格納しておき、次の 2 段階で値を予測する。最初のレベルでは、予測するインスタンスの命令アドレスを用いて VHT のエントリを引き、そこに格納されている 4 つの値を出力する。2 レベル目で、4 つの中から 1 つの値を選択することで予測値を得る。2 レベル目の選択は、過去の p 回の予測結果とカウンタを用いて決定する

が詳細は省略する。

文献 [7] では、Last-value 予測とストライド値予測のハイブリッド値予測機構、ストライド値予測と 2 レベル値予測のハイブリッド値予測機構を提案評価しており、複数の予測アルゴリズムを利用することで予測範囲が広がることを確認している。

文献 [6] では、過去の連続した有限個の値の結果を保存しておき、同一のコンテキストが繰り返されると仮定して予測をおこなうコンテキスト・ベースの値予測機構 finite context method predictors を提案している。この値予測機構では、次のように

```
Value sequence: a a a b c a a a b c a a a  ?
```

値が変化した時 (a,b,c という文字は 32 ビットの値を表すシンボル)、a が 3 回連続した後には b がくる確率が高いという情報を保存しておき b という値を予測する。文献 [6] では コンテキスト・ベースの値予測機構がストライド値予測機構や Last-value 予測機構より高い予測成功率を達成する可能性があるとしているが、ハードウェア量や実装に関する議論は今後の課題としている。

2.4 ソフトウェア・サポート

値予測に対するソフトウェア・サポートの手法が提案されている。文献 [1] では、プロファイルを用いて、予測する/しないという情報を命令に付加する手法を提案している。また、予測をおこなう確信度を 50%〜90%に設定して提案手法を評価しており、閾値を 60%以上に設定することでハードウェアのみの値予測機構より低いミス率を達成できるとしている。文献 [8] 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おこなうが、予測ミスの処理をソフトウェアで実行する手法を提案評価している。

2.5 値予測機構による性能向上

値予測機構の評価は、プロセッサ性能に与える影響とハードウェア量という基準を用いて議論する必要がある。値予測機構を用いてプロセッサ性能を向上させるためには、性能向上につながる命令を正しく予測するとともに、予測ミスの回数を削減することが重要となる。

Classification Table を用いた Last-value 予測、ストライド値予測、2 レベル値予測、ストライドと 2 レベルのハイブリッド値予測について、全実行命令に対するヒット率とミス率を表 1にまとめる。これらの値は文献 [2] のデータを用いて計算した。表 1から、複雑な値予測機構を用いることで正しく予測できる割合とミス率が共に増加していくことがわかる。ハイブリッド値予測機構を用いた場合には、全実行命令の約 40%の演算結果を正しく予測できる反面、平均で 17 命令実行するたびに 1 回の値予測ミス (ミス率 5.9%) が発生することになる。予測ミスが発生した場合には、通常 1 サイクル以上のペナルティが必要となるため、ミス率の削減が性能向上のための条件の一つとなる。

	LV+CT	Stride	2-level	Hybrid
Hit	21.7%	29.8%	29.4%	39.9%
Miss	1.7%	2.9%	3.9%	5.9%

表 1: 全実行命令に対するヒット率とミス率

3 2 レベル・ストライド値予測

2.2節で示したストライド値予測機構では、次の例のように値が初期化される際にミスが起こる。この例で正しく予測できる命令はわずか 40%である。

```
Value Sequence: 1  2  3  4 5 1 2  3  4 5 1 ...
                NP NP NP H H M NP NP H H M
```

この点を改良するために、分岐履歴の情報を用いたパス・ベースの値予測機構として、2 レベル・ストライド値予測機構を提案する。

2 レベル・ストライド値予測機構は、値が一定間隔で変化して (予測が当たって) いる時と、値が初期化される時で異なった制御の流れをとり、この制御流の変化を検出することで初期化のタイミングを予測できるという仮定に基づいて予測をおこなう。この仮定を次の例で説明する。

```
for(i=0; i<10; i++)
  for(j=0; j<5; j++) sum += j;
```

ソースコードをコンパイルして次のコードを得る。

```
(1)   mov 0,%o1         ; sum=0
(2)   mov 9,%o2         ; i=9
(3)   mov 0,%o0         ; j=0
   .LL13:
(4)   add %o1,%o0,%o1   ; sum+=j
   .LL12:
(5)   add %o0,1,%o0     ; j++, Predict this!
(6)   cmp %o0,4         ; j<4 ?
(7)   ble,a .LL12       ; conditional branch
(8)   add %o1,%o0,%o1   ; delayed slot,sum+=j

(9)   addcc %o2,-1,%o2 ; i--
(10)  bpos,a .LL13      ; conditional branch
(11)  mov 0,%o0         ; delayed slot, j=0
```

命令 (5) が生成する値 (コード中の j++ に対応する) の予測を考える。この命令は本節の最初に示した様に 1,2,3,4,5 という値を繰り返し生成する。また、このプログラムを実行した時の条件分岐命令 (命令 7 と 10) の結果は分岐成立を 1 不成立を 0 で表現すると下のビット列となり、予測する値が 1 に

```
Branch result: 1111101 1111101 1111101...
```

初期化される時点までの分岐結果は ...01 となっていることがわかる。このコードの場合には値予測

の際に最近の分岐結果の 2 ビットからなるビット・パターンを見て、このビット列が 01 だった場合に値 1 を予測するようにすればよい。

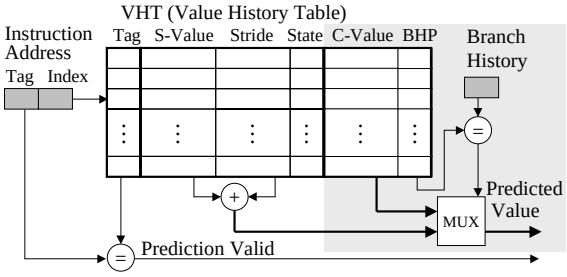

図 3: 2 レベル・ストライド値予測機構のブロック図

2 レベル・ストライド値予測機構のブロック図を図 3に示す。図 3の右側の網掛けの部分はストライド値予測機構からの拡張部分を示している。ただし、混乱を避けるために、ストライド値予測機構における Value フィールドを S-Value という名前に変更した。

予測アルゴリズムを説明する。VHT の状態が Steady になるまでは、2.2節で示したストライド値予測機構のアルゴリズムと同様に動作する。状態が Steady になってから、ストライドによる予測がはずれた時に (i) 計算により得られた値を C-Value に格納し (この値が初期化の値となる。)、(ii) 分岐履歴パターン (条件分岐命令の結果を連結したビット列) を VHT の分岐履歴パターン (BHP) フィールドに格納する。

以降の予測では、予測をおこなっている時点の分岐履歴パターンと VHT の BHP フィールドとの比較が一致した場合には C-Value の値を予測値とし、一致しない場合にはストライド値予測の値を利用する。C-Value の値が予測値として利用された場合は値が初期値に更新されることを意味する。この場合には C-Value の値を用いて S-Value の値を更新する。この時 Stride の値は変更しない。状態も Steady のまま変更する必要はない。

4 Multi-stride 値予測機構

複雑なアルゴリズムを用いた予測機構は多くのハードウェア量を必要とする。更に学習期間などの存在により単純な予測機構より性能が低下する場合もある。2 レベル・ストライド値予測機構はストライド値予測機構を拡張したものであるが、ストライド値予測機構を用いて高い成功率で予測できる静的な命令の集合に対して 2 レベル・ストライド値予測機構を用いる必要はない。同様に Last-value 予測機構で予測できる静的な命令の集合についてはストライド値予測機構を使う必要はない。

以上の考察から、Last-value、ストライド、2 レベル・ストライドの 3 つの値予測機構とプロファイルを組み合わせた Multi-stride 値予測機構を提案する (図 4)。

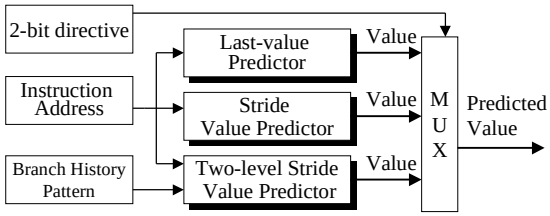

図 4: Multi-stride 値予測機構のブロック図

3 つの値予測機構のどれを用いるかという指示 (予測しないという指示を合わせて 2 ビットで表現) を静的にそれぞれの命令に付加する。この分類指示を動的に生成または変更することはない。このような静的な手段を用いない場合には、複数の中から利用する値予測機構を選択するハードウェアとその性能が問題となるが、静的に利用する値予測機構を指定する本方式では、複数の値予測機構を用いることによる性能低下はおこらない。

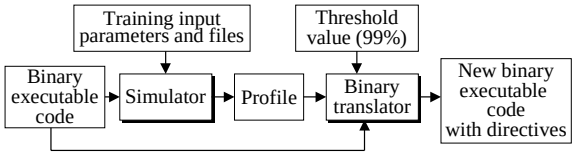

図 5: プロファイルを用いた分類指示の生成

それぞれの命令に指示を付加することは単純な作業ではないが、図 5に示す様に、プロファイルを解析することで指示を付加する。値予測をおこなうかどうかの選択のためにある閾値を用い、その閾値以上の予測成功率を達成する命令にのみ、予測をおこなう指示を付加する。

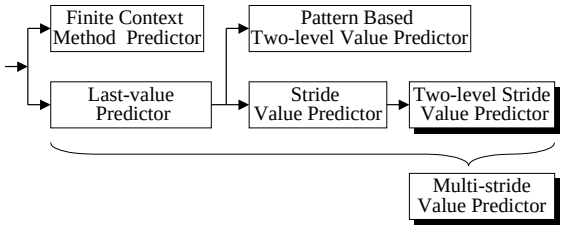

図 6: 値予測機構の分類

従来から提案されている値予測機構と本稿で提案する値予測機構の関係を図 6にまとめる。図の矢印はベースとなった値予測機構を表している。2 レベル・ストライド値予測機構は分岐履歴パターンを用いるパス・ベースの値予測機構という点に特徴がある。Multi-stride 値予測機構は 3 つの値予測機構からなるハイブリッドであるという点、プロファイル情報を用い静的に指示を与えるという点に特徴がある。

5 評価環境

5.1 ベンチマーク・プログラム

SPEC95 の整数系ベンチマークの中から 6 本[1] のプログラムを用いて値予測機構を評価する。それぞれのベンチマーク・プログラムの入力セットは実行命令数が 1 億程度に収まるように調整した。ベンチマーク・プログラムと入力セットを表 2にまとめる。これらのプログラムは SPARC Architecture version 7 をターゲットとし、Gnu C compiler(オプション -O) を用いてコンパイルされている。

Program	Description	Input Set
cc1	From GCC 2.7.1	genrecog.i
compress	File compression	14000 e 2231
go	Plays game of Go	9 9
m88ksim	Moto88K simulator	ctl.in
perl	Manipulates strings	admits, 1/8 input
xlisp	LISP interpreter	6 queens

表 2: ベンチマークプログラム

Benchmark	Dynamic Instr.Count	Dynamic Count of Eligible Instr. for Value Prediction
cc1	120M	82M (68.8%)
compress	54M	37M (69.9%)
go	136M	107M (78.7%)
m88ksim	124M	86M (69.9%)
perl	101M	65M (64.0%)
xlisp	42M	25M (60.1%)

表 3: 実行命令数と値予測の対象となる命令数

各ベンチマーク・プログラムの実行命令数と、値予測の対象となる命令数 (割合) を表 3にまとめる。値予測の対象となる命令は値を生成する、ロード、シフト、算術論理演算である。値予測の対象とならない命令には分岐, ストア, 浮動小数点演算などがある。また、本評価では複数ワード (64 ビット以上) の値を生成する命令も値予測の対象外とした。

5.2 プロセッサ・モデル

命令フェッチ、ディスパッチ、オペランドフェッチ、実行、メモリアクセス、完了という 6 段のパイプライン・ステージからなる標準的なアウトオブオーダ実行のプロセッサモデルをベースとして、値予測の影響を測定するプロセッサ・モデルを定義する。このモデルでは、データ依存関係とそれを解消する値予測機構が命令レベル並列性に与える影響を明確に測定するために次の仮定を用いている。100%の精度の分岐予測機構とミスをおこさない命令およびデータキャッシュ。ロード・ストアのアドレスはパイプラインの早い段階で予測され、メモリ上で依存関係のないロード・ストア命令はアウトオブオーダで処理可能とする。リオーダバッファのエントリ数を除くハードウェア資源 (命令ウィンドウのエントリ数や、機能ユニットの数等) を無限大とする。

Multi-stride 値予測機構は命令フェッチ・ステージでプログラム・カウンタ、分岐履歴パターン、利用する値予測機構を示す指示を受け取り、オペランドフェッチ・ステージの終了までに予測値を生成する。表 1にまとめた様にハードウェアのみの (プロファイルを用いない) 値予測機構によるミス率が 1.7%〜5.9%であることを考慮して、これらの値より低いミス率を達成するために、値予測をおこなう閾値を予測成功率 99%に設定する。ベンチマークの全実行トレースを解析して、それぞれの命令が利用する値予測機構を決める分類指示を得る。それぞれの値予測機構の値履歴テーブルのエントリの数に制限を加えない。静的に同一の命令から動的に生成される複数の値の予測を 1 サイクルで処理できるとする。

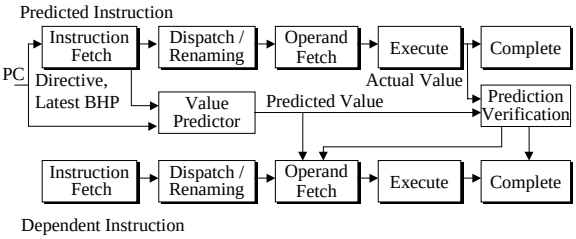

図 7: 依存関係のある命令の実行タイミング

値予測に失敗した場合には、正しい実行を保証するために、予測した値に依存しており、すでに実行の開始されている命令を再実行する必要がある。この値予測からの回復については幾つかの戦略が考えられる。文献 [9] の評価で用いられているように、予測ミスした命令以降の全ての命令を再実行することで予測ミスから回復できる。この方法は分岐予測に失敗した場合と同様の処理で値予測ミスに対処できるために実装が容易となる利点がある反面、再実行する命令数が多い（ペナルティが大きい）という欠点がある。文献 [4] では予測によって得られた値に依存する命令については発行された後にもリザベーション・ステーションのエントリを解放せずに、予測がミスした場合には正しいオペランドを用いて依存関係のある命令のみを再実行するとしている。この手法を用いた場合のデータの流れを図 7に示す。この手法の実装について文献 [9] で議論されているが、回復機構に関する評価は今後の課題としている。

本評価では、値予測に失敗した場合には 1 回のミスについて 5 サイクルのペナルティが必要と想定して評価をおこなうことにする。文献 [4] で用いている 1 サイクルというペナルティと比較し、5 サ

[1] シミュレータの動作が不安定なため SPECint95 に含まれる ijpeg と vortex は評価に加えなかった。

イクルというペナルティは予測値と計算結果との比較、依存関係にある命令の検出、それらの再実行といった処理を考慮して現実的なサイクル数として設定した。

6 評価結果

6.1 分岐履歴パターンのビット長

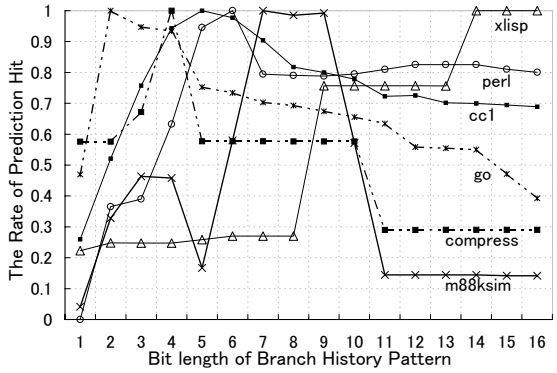

図 8: 分岐履歴のビット長と予測できる量の推移

2 レベル・ストライド値予測機構で用いる分岐履歴パターン (BHP) のビット長を 1 ビットから 16 ビットまで変化させた時に正しく予測できた量を図8に示す。図8では、予測できた命令数ではなく、最も多く正しい値を予測できた設定における回数を 1 とした相対値で評価している。

図8に示す結果より、xlisp と perl の一部を除いて、ビット長が 2 から 7 の間にピークを持ち、その後ビット数が増加するに従い正しく予測できる数が減少していくことがわかる。また、正しく予測できる命令数はビット長に大きく影響を受けることがわかる。例えば compress ではビット長を 4 ビットから 1 ビット変更し 5 ビットにするだけで、正しく予測できる範囲が 40%以上減少してしまう。図8の結果から、プログラム全般についての最適なビット長を固定的に決定することは望ましくなく、各プログラム毎に最適なビット長を選んで設定する必要がある。以降この設定を用いて評価を進めていくことにする。

6.2 正しく予測した命令の割合とミス率

Multi-stride 値予測機構で正しく予測できた割合を図9 にまとめる。各プログラムについて、(1)Last-value 予測機構で正しくし予測した割合、(2) ストライド値予測機構で正しく予測した割合、(3)2 レベル・ストライド値予測機構で正しく予測した割合、(4) 予測ミスまたは予測をおこなわなかった割合、(5) 値予測の対象とならない命令の割合、の 5 つに分けて表示した。Multi-stride 値予測機構では 99%以上の予測成功率を満足する静的な命令だけが予測の対象となるが、3 つの値予測機構で正しく予測した割合は全実行命令数の 21%から 51%に達することがわかる。特に 51%と割合の多かった m88ksim の場合には 2 命令毎に 1 度の値予測が必要となり、高い並列性を利用するプロセッサでは 1 サイクルに複数の値予測が必要となる。

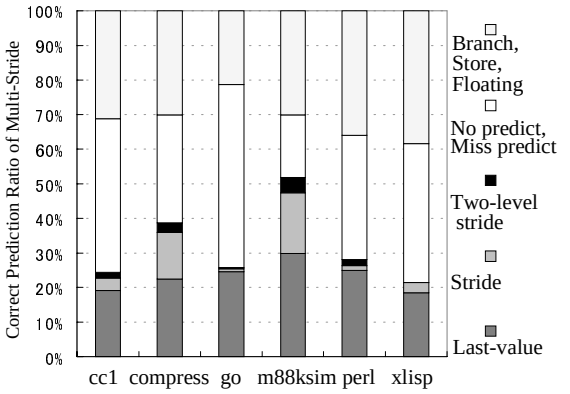

図 9: Multi-stride 値予測機構の予測成功率

Multi-stride 値予測機構で用いた 3 つの値予測機構のそれぞれについて、予測に失敗した回数と予測失敗率 (予測を失敗した回数と予測をおこなった回数の割合) を表 4にまとめる。トレースを解析して予測成功率が 99%以上の命令のみを予測対象としているためミス率が 1%以下となることは明らかだが、測定結果は 1%よりかなり低い値となっている。

Program	Last-value	Stride	2-level Str.
cc1	10734(.05%)	33591(.77%)	13287(.68%)
compress	2679(.02%)	3094(.05%)	1489(.01%)
go	1934(.01%)	4827(.37%)	593(.11%)
m88ksim	16262(.04%)	43832(.20%)	29041(.53%)
perl	1245(.01%)	1544(.11%)	2950(.01%)
xlisp	1904(.02%)	2950(.24%)	9(.05%)

表 4: 値予測でミスした回数 (ミス率)

6.3 Multi-stride 値予測による性能向上

5.2 節で定義したプロセッサ・モデルで利用できる命令レベル並列性を表 5,6 にまとめる。表 5,6 に示す並列性とは、実行命令数をシミュレータで測定した実行サイクル数で割った値である。値予測機構を用いない通常のプロセッサで利用できる並列性を表5にまとめる。リオーダバッファのエントリ数 32 という値は現在利用できるハードウェア量を考慮した値である。その 4 倍のエントリ数 128 という値は数年先のプロセッサを想定した値である。

値予測機構を用いた場合の並列性を表6にまとめる。表の 2,4 列には値予測がミスした際のペナルティを考慮しない並列性を示した。一方、表の 3,5 列はペナルティを考慮した場合の並列性である。1 回の予測ミスにつき平均して 5 サイクルのペナルティを想定し、表 4のミス回数を利用して予測ミス

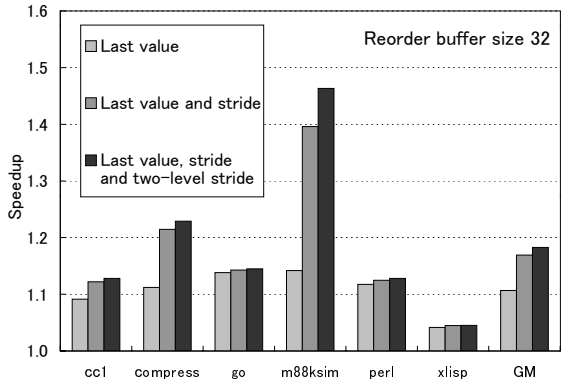

図 10: Multi-stride 値予測機構による性能向上率

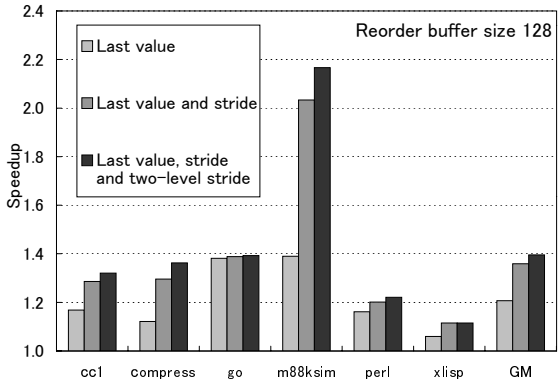

図 11: Multi-stride 値予測機構による性能向上率

の影響を計算した。m88ksim で並列性が 1 程度変化しているが、それ以外のプログラムでは予測ミスによる影響は非常に小さい。

Reorder Buffer	32 entry	128 entry
cc1	4.33	8.20
compress	4.34	8.56
go	3.68	6.83
m88ksim	4.61	7.62
perl	4.30	8.40
xlisp	4.55	9.70

表 5: 値予測を用いない場合の命令レベル並列性

Reorder Buffer Entry ミスペナルティの影響	32 なし	32 あり	128 なし	128 あり
cc1	4.95	4.89	11.11	10.82
compress	5.36	5.34	11.75	11.66
go	4.22	4.22	9.53	9.51
m88ksim	6.93	6.76	17.55	16.51
perl	4.85	4.85	10.27	10.26
xlisp	4.77	4.76	10.88	10.81

表 6: Multi-stride 値予測機構による並列性の向上

値予測機構を用いなかった場合の性能（表 5）を 1 として、値予測機構により得られる性能向上率を図 10, 11 に示す。図 10,11 はそれぞれリオーダバッファのエントリ数を 32, 128 とした場合の結果であり、右端の GM は 6 つのプログラムの幾何平均を意味している。図 10,11の結果では予測ミスのペナルティを考慮している。

正しい値を予測する割合の高い m88ksim（図 9）で高い性能向上率を得た。逆に予測できる割合の低い xlisp が性能向上率も最も悪い。幾何平均でみると、リオーダバッファのエントリ数が 32 の時で 18%の性能向上を確認できた。この時 Last-value 予測機構に起因する向上率は 10%, ストライド値予測機構に起因する部分が 6%、2 レベル・ストライド値予測機構による向上率が 2%だった。同様に、リオーダバッファのエントリ数が 128 の時には 40%の性能向上を確認できた。この時 Last-value 予測機構に起因する向上率は 21%, ストライド値予測機構に起因する部分が 15%、2 レベル・ストライド値予測機構による向上率が 4%だった。

以上の結果より、予測ミスによるペナルティを 5 サイクルと現実的な値に設定した場合でも、提案した Multi-stride 値予測機構により平均 40%の性能向上率 (リオーダバッファのエントリ 128 の時) が得られることを確認できた。

7 議論

7.1 値履歴テーブルにおけるエントリ数

先の評価では、十分大きなエントリ数を持つ値履歴テーブルを仮定した。実行時に必要となった値履歴テーブルのエントリ数を表 7にまとめるが、Last-value, ストライド、2 レベルストライドと順番に利用するエントリ数が少なくなっている。

Benchmark	Last-value	Stride	Two-level Str.
cc1	30,539	1131	254
compress	754	94	12
go	15,545	540	88
m88ksim	1,914	344	39
perl	4,159	152	20
xlisp	1,122	112	7

表 7: 利用された値履歴テーブルのエントリ数

値履歴テーブルのエントリ数を変化させて正しく予測できる命令数を測定したところ、Last-value 予測のためにダイレクトマップ 4K エントリ (4K entry × (20bit + 32bit) = 記憶容量 26KB)、ストライド値予測のためにダイレクトマップ 512 エントリ (512 entry × (23bit + 32bit + 32bit +2bit) = 記憶容量 6KB) のテーブルを用意することで、エントリ数無限大のテーブルを用いた場合の 95%以上を正しく予測できるという結果を得た。2 レベル・ストライド値予測については数十エントリのフルアソシィアティブまたは 256 エントリ程度のダイレクトマップを用意することで競合を回避できる。

エントリの競合により予測できる量は5%程度減少するが、これは予測できない回数を増やすのであって、予測ミスの回数を増やすことにはならない。このため、現実的なハードウェア量のテーブルを用いることによる性能低下は深刻にならないと考えられる。

7.2 プロファイルを用いた命令の選択

本稿の評価では、ベンチマークプログラムの全トレースを解析して、それぞれの値予測機構で予測する命令の集合を得た。この意味で、本評価は静的な解析を理想的におこなった場合の評価である。しかし、プログラムには入力データに依存しない固有の性質があり、ある入力データで高い予測成功率を示す命令は他の入力データにおいても高い成功率を示すことが確認されている [1]。特に、今回用いた 99% という閾値と実際の予測ミス率は 1%より非常に低かったという結果 (表 4) より、今回利用した高い予測成功率を示す命令は、プログラム固有の性質に強く影響を受けていると考えられる。数種類のトレーニング・データを解析することでこれらの命令を検出できる可能性が高い。静的に分類指示を付加するアルゴリズムの検討と評価は今後の研究課題である。

7.3 分岐履歴パターンのビット長

2 レベル・ストライド値予測機構の性能は分岐履歴パターンのビット長に大きく依存する (図 8)。これは、VHT 中のビット列と分岐履歴パターンとの単純なマッチングを用いて値が初期化されるタイミングを検出しており、初期化に関係ない分岐の結果が悪影響を与えているためと考えられる。マッチングの際に、悪影響を与える部分をマスクするといった改良により 2 レベル・ストライド値予測機構の性能を改善できる可能性がある。

8 まとめと今後の課題

Last-value 予測機構、ストライド値予測機構、2 レベル・ストライド値予測機構、の 3 つの値予測機構を用いた Multi-stride 値予測機構を提案した。この Multi-stride 値予測機構はプロファイルを用いて静的に予測する命令の集合を定義する。本稿では 99%という閾値を用いて予測する命令の集合を求めた。

シミュレーションによる評価により、予測ミスの回数を抑えつつ全実行命令の 21%から 51%の命令の演算結果を正しく予測できることを確認した。100%の精度の分岐予測、ミスをおこない命令およびデータキャッシュ、豊富なハードウェア資源を用いた理想的なプロセッサにおいて、分岐予測 1 回の予測ミスに対して 5 サイクルというペナルティを考慮した場合でもリオーダバッファのエントリ数を 128 と大きく設定することで平均 40%の性能向上を達成できることを確認した。この時 Last-value 予測機構に起因する向上率は 21%, ストライド値予測機構が 15%、2 レベル・ストライド値予測機構による向上率が 4%であった。

今後、分岐予測、データキャッシュ、機能ユニットの数といった本評価で理想化した部分の影響と、予測ミスからの回復機構を含めた検討評価が必要と考えている。

謝辞

本研究は我々研究室内の大規模データパス・プロジェクトの一部としておこなわれたものであり、多くの貴重な御意見を下さったプロジェクトメンバ、並びに査読者の方々に感謝致します。本プロジェクトの一部は (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] Feddy Gabbay and Avi Mendelson. Can program profiling support value prediction? *Proc. of the 30th Annual International Symposium on Microarchitecture*, pp. 270–280, December 1997.

[2] Jose Gonzalez and tntonio Gonzalez. The potential of data value speculation to boost ILP. *In Proc. of the International Conference on Supercomputing*, pp. 221–228, 1998.

[3] J. L. Hennessy and D. A. Patterson. *Computer Architecture a Quantitative Approach*. Morgan Kaufman Publishers, Inc, 1995.

[4] H. Lipasti and John Paul Shen. Exceeding the dataflow limit via value prediction. *Proc. of the 29th Annual International Symposium on Microarchitecture*, pp. 227–237, December 1997.

[5] Mikko H. Lipasti, Christopher B. Wilkerson, and John Paul Shen. Value locality and load value prediction. *In Proceedings of ASPLOS VII*, pp. 138–147, October 1996.

[6] Yiannakis Sazeides and James E. Smith. The predictability of data values. *Proc. of the 29th Annual International Symposium on Microarchitecture*, pp. 248–258, December 1997.

[7] Kai Wand and Manoj Franklin. Highly accurate data value prediction using hybrid predictors. *Proc. of the 30th Annual International Symposium on Microarchitecture*, pp. 281–290, December 1997.

[8] Chao ying Fu, Matthew D. Jennings, Sergei Y.Larin, and Thomas M. Conte. Value speculation scheduling for high performance processors. *In Proceedings of ASPLOS VIII*, pp. 2621–271, 1998.

[9] 佐藤寿倫. アドレス名前替えによるロード命令の投機的実行. 並列処理シンポジウム JSPP'98 論文集, pp. 15–22, 1998.