計算機アーキテクチャ 149－11
（２００２．８．２２）

値予測における値履歴テーブルエントリ数の削減

飯塚大介[†] 角田忠信[††]
坂井修一[††] 田中英彦[††]

値予測は真の依存関係を解消することで並列度を抽出する手法である。性能向上率を向上させるために予測精度を高めた種々の予測方式が提案されているが、予測に必要なハードウェア量が膨大なため実装するのが困難となっている。特に予測履歴テーブルは大きな資源を必要とするため、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る手法が近年いくつか提案されている。本稿では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た場合の性能低下を抑制する方式を提案する。その結果、同じエントリ数の場合に性能向上率が若干向上することを確認した。

Decreasing Value History Table Entry on Value Prediction

Daisuke Iizuka,[†] Tadanobu Tsunoda,[††] Shuichi Sakai[††]
and Hidehiko Tanaka[††]

Value prediction is a technique to resolve true dependency and exploit more parallelism. Various value prediction mechanism have been proposed to achieve high prediction accuracy. However, those sophisticated mechanisms often require a large amount of hardware, especially for the Value History Table(VHT). Recently, several researches have been done on the techniques to reduce the VHT size by cutting the length of tag or prediction values, or by limiting the number of table entries. This paper proposes a technique to reduce the VHT entries without loosing performance based on static analysis. We propose to use VHT only for instructions that have been selected by the compiler. Evaluation results show that, using our technique, a higher performance can be achieved with the VHT of the same number of entries.

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、プログラム中には真の依存関係が存在するため、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データ投機手法として、命令が生成する値を予測する値予測が考案されている[5]。値予測では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。しかし、予測ミスを起こした場合は誤った予測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。値予測の精度を向上させるための手法として、ハイブリッド型値予測機構[10]やコンテキスト型値予測機構[6]が提案されている。これらの予測機構を用いれば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る。また、値予測でヒットする命令数を増加させるためには、予測値を格納しておく値履歴テーブル (Value History Table, VHT) のエントリ数を大きくしておく必要がある。しかし、VHT を大きくすると大きなハードウェアコストが必要となり、また高クロックで動作させることも難しくなる。本稿では、値予測の実装可能性を高めるために、VHT のエントリ数を削減した際に生じる性能低下を、予め静的に予測命令を選択しておくことで抑える手法を提案する。本稿の構成は以下の通りである。2 節で関連研究を述べる。3 節で評価方法を説明し、4 節で提案手法を紹介する。5 節で評価を行い、最後に 6 節でまとめる。

† 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

2. 関連研究

本節では、値予測のハードウェア量を削減する方式や、高クロックで動作する値予測機構、VHT のポート数が少ない時の性能向上方法等を提案している関連研究について述べる [2)7)15)]。文献 [12)] では、VHT のタグビット幅を削減する手法を、文献 [13)] では、予測値を 0/1 に限定する手法を提案している。VHT を用いずに値予測を行う方式も提案されている [4)]。文献 [1)] では、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時と同時に PVC に格納されている値をフェッチし、その値を使用して値予測を行う。これにより、高クロックかつ低消費電力で値予測が実現できるとしている。文献 [3)] では、既存の分岐予測機構を値予測の確信度の評価の機構として利用し、予測値を取り出す命令と VHT を更新する命令を追加し、更にクリティカルパスを考慮して予測命令を選択する方式を提案している。その結果、最も最適化した場合は静的に最大 136 命令だけを選択的に予測すれば十分な利得が得られるとしている。文献 [11)] 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。

3. 評価方法

本節では評価方法について説明する。4 節で述べる手法は本節で述べる評価方法の下で行っている。

3.1 プロセッサモデル

評価に用いるプロセッサモデルとして、リオーダバッファエントリが 64 のスーパースカラプロセッサを用いた。命令セットは Compaq Alpha と互換である。パイプラインは 10 段である。命令フェッチ幅とデコード幅、リネーム幅、命令発行幅は 4 である。機能ユニットは、ロードストア以外の全命令が実行可能なものが 4 つ、ロードストアユニットが 2 つある。ロード命令を除く全命令の実行レイテンシは 1 サイクルである。ロード命令はアドレス計算に 1 サイクル、データキャッシュアクセスに 2 サイクル必要とする。キャッシュミス時には 6 サイクルを必要とする。データキャッシュは容量 64kB、ブロックサイズ 64B、4 ウェイセットアソシアティブである。先行するストア命令が完了しなければ後続するメモリアクセス命令は実行できない。命令キャッシュは容量 64kB、ブロックサイズ 16B、2 ウェイセットアソシアティブである。ミスレイテンシは 6 である。2 次キャッシュはデータと命令で共有で、容量は無限大とした。分岐先アドレス予測には 1k エントリ 2 ウェイセットアソシアティブの分岐バッファを用いた。条件分岐の予測には 4k エントリ 2 レベル適応型の gshare 予測器を使用した。値予測機構としては LastValue 予測のみを用いた。その他の予測機構は LastValue 予測機構よりも複雑で実装可能性が低く、かつ高精度で予測される命令数が LastValue よりも少ないために本研究では使用しない。ハードウェア量削減のため、VHT の各エントリにタグは使用しない。各エントリに 4 ビットの飽和カウンタを設け、最大値になった時にのみ予測を行う。前回と同じ値が生成された時はカウンタを 1 増加させ、違う値が生成された場合はカウンタをリセットする。値予測ミスが発生した際には、分岐予測ミス同様にパイプラインをフラッシュし、当該命令の次の命令から再実行する方式をとった。

表 1 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	127M
gcc	genrecog.i	142M
go	9 9	143M
ijpeg	specmun.ppm	127M
li	train.lsp	217M
m88ksim	dcrand.lit	147M
perl	scrabbl.in.small	129M
vortex	persons.250	120M

3.2 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 を用いた。ベンチマークは Alpha 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。表 1 にあるように、ベンチマークは 100M〜200M 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てある。

4. 値履歴テーブルエントリ数の削減

本節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 のエントリ数を削減した際に生じる性能低下を抑える方式を提案する。

4.1 高ヒット命令の分類

プログラムに含まれる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は他のクラスに所属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

RETVAL 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

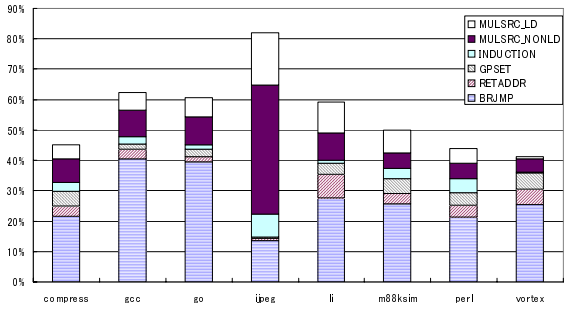

図 1 値を生成する命令の分類

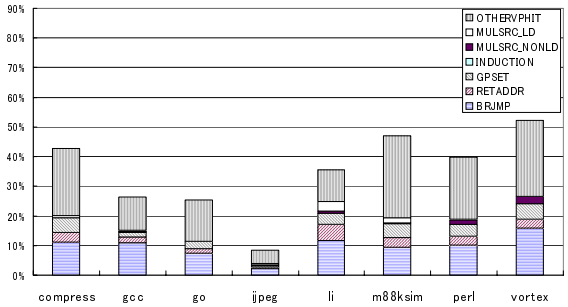

図 2 99%LastValue 予測ヒット命令の分類

MULSRC_NONLD 予測命令がロード命令以外であり、かつ、当該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ことがある命令

MULSRC_LD 予測命令がロード命令であり、かつ、当該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ことがある命令

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 1 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する。それらが値を生成する全実行命令に対してどの程度の割合を占めるかを図 2 に示す。図 2 には 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を **OTHERVPHIT** として示してある。命令を静的にこれらのクラスに分類した際に、いくつかのクラスの命令についてを LastValue 予測を行わないようにして、VHT の容量性ミスによる性能低下を防ぐ。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。

4.2 BRONLY: 分岐命令にのみ使用される値

図 3 にあるようなコードがあるとする。そのデータフローグラフは図 4 のようになる。現在のプロセッサは高クロックで動作させるためにパイプライン処理を行っている [9]。パイプライン処理を行い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 [8]。図 4 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、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事が言えると予想される。従って、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。

```
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 3 BRONLY コード例

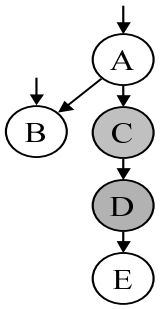

図 4 図 3 のデータフローグラフ

逆に、図 4 において、命令 D を値予測する場合を考えてみる。命令 E の条件分岐予測がヒットしたが、命令 D で誤った値を予測し、命令 E で誤った値を使用し分岐予測ミスが発生し、さらにその後命令 D の予測ミスが発覚し正しい値が生成され、再度命令 E で分岐予測ミスが発生するという事態が発生するということが起きる可能性がある。分岐命令にのみ使用される値は値予測を行わないようにすることで、このような値予測ミスによる誤った分岐予測ミスが起きる可能性を削減することができる。

4.3 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一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、ロード命令 1 命令だけで値をセットすることができる。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 5 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

```
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 5 呼び出し元関数例

```
  <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 6 呼び出し先関数例

は常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。従って、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。

4.4 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 5 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 6 の命令 K) 際に生成される。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 6 の命令 J)、リターン命令で使用される (図 6 の命令 L)。さらに、Alpha の場合は、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 5 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、4.2 節で説明した分岐命令の時と同様、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、4.3 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。従って、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。

4.5 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 1 のように多くの誘導変数

```
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 7 MULSRC コード例

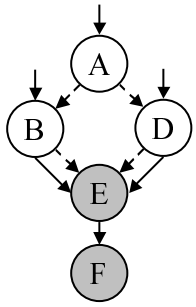

図 8 図 7 の制御・データフローグラフ

生成命令が実行されているが、図 2 からわかるように LastValue でヒット率が高い誘導変数は非常に少ない。従って、誘導変数は予測対象から外すこととする。

4.6 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 7 のコードについて考える。このコードのの制御フロー (破線) とデータフロー (実線) は図 8 のようになる。図 8 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 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。命令 F に関しても同様の事が言える。図 E の命令がロード命令以外の場合、ある決まった入力値に対しては、必ずある決まった値を毎回返す。従って、入力値が異なれば出力値も異なる場合が多くなると考えられ、LastValue 予測機構でヒットしにくくなると考えられる。このように、ロード命令以外で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_NONLD** クラスに所属するものとする。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。これに所属する命令を値予測しないようにすると、VHT を汚染しないことにより予測した場合よりも性能向上が得られる場合と、本来ならば予測がヒットして得られるはずの利得が得られずに性能が低下する場合の両方が考えられる。しかし図 1、図 2 からわかるように、MULSRC_NONLD に属する命令は、実行される命令としては多いものの、LastValue 予測機構でヒットする命令数が非常に少ないプログラムが多い。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。

図 E の命令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。また、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

表 2　予測除外命令選択法

BRONLY+	GPSET INDUC BRONLY
MULUSE-	GPSET INDUC BRONLY MULUSE_NONLD
MULUSE+	GPSET INDUC BRONLY MULUSE_NONLD MULUSE_LD

えられる。アドレスが異なっていてもそこに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非常に少ないプログラムが多い。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。

4.7　予測除外命令選択法式

本節では、4.2 節〜4.6 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説明する。まず、BRONLY、RETVAL、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。次に、BRONLY+に加えて MULUSE_NONLD を予測対象から外したものを **MULUSE-**とする。MULUSE-に加えて MULUSE_LD を更に予測対象から外したものを **MULUSE+**とする。以上の 3 つの予測除外命令選択法を表 2 に示す。

5. 評　　価

本節では、4 節で提案した予測命令選択法に基づき、静的に予測しない命令を選択しておいた上で、VHT エントリが 128,256,512,1024 の場合について、速度向上率について評価を行う。

5.1　速度向上率

速度向上率を図 9、図 10 に示す。グラフの各ベンチマーク毎に、左から 4 本を一組と見た場合、左から VHT エントリサイズが 128,256,512,1024 の場合を示す。同じ VHT エントリサイズのグラフは 4 本あるが、その中で左から順に、通常の値予測、分岐命令に使用されるものを除外 (BRONLY+)、分岐命令と複数の入力があるもの (ロードを除く) を除外 (MULSRC-)、分岐命令と複数の入力があるもの (ロードを含む) を除外 (MULSRC+)、となっている。

5.2　考　　察

go、gcc、ijper、perl では、通常の値予測と比較して、BRONLY+、MULUSE-、MULUSE+の順に実行速度が向上している。これは予測命令数が削減されることで、VHT エントリの容量性ミスが生じる頻度が減少したことによると考えられる。compress では他のベンチマークに比べ、global 変数と static 変数のアドレスを取得する命令が多い。そのため VHT での容量性ミスが発生しやすく、これらアドレス生成の元となる GPSET を予測しなかったため性能が低下したと考えられる。えられる。li では BRONLY+では性能が向上するが、MULUSE-、MULUSE+では性能が低下する。特に MULUSE+では著しく性能が低下するが、これは図 2 からわかるように li では MULUSE+に属する命令がヒットする割合が他のものに比べて大きいからである。MULUSE+を予測対象から外すことによる性能低下は m88ksim にも見られるが、これも同じ理由であると考えられる。

6. ま と め

本稿では、プログラムを静的に解析して値予測となる対象を制限することで、VHT エントリ数を削減した時に生じる容量製ミスにより性能向上率が低下する手法を提案した。その結果、エントリ数が 128 の ijpeg のように、選択しなければ予測ミスによるペナルティーにより通常実行よりも性能が低下するが、選択することで速度向上が得られ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わかった。今後の課題としては、更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良いのかを検討する。また、これと合わせてクリティカルパスにも注目して予測命令を選択する方式についても検討する。

参 考 文 献

1) Ravi Bhargava and Lizy K. John. Latency and Energy Aware Value Prediction for High-Frequency Processors. In *16th Annual ACM International Conference on Supercomputing(ICS)*, Jun 2002.
2) Brad Calder, Gleen Reinman, and Dean M. Tullsen. Selective Value Prediction. In *26th International Symposium on Computer Architecture(ISCA-26)*, May 1999.
3) Eric Larson and Todd Austin. Compiler controlled value prediction using branch predictor based confidence. In *Proc. 33rd Annual International Symposium on Microarchitecture*, Dec 2000.
4) Sang-Jeong Lee, Yuan Wang, and Pen-Chung Yew. Decoupled Value Prediction on Trace Processors. In *Sixth International Symposium on High-Performance Computer Architecture(HPCA-6)*, Jan 2000.
5) John Paul Shen Mikko H. Lipasti. Exceeding the Dataflow Limit via Value Prediction. In *Proc. 29th Annual International Symposium on Microarchitecture(MICRO-29)*, Dec 1996.
6) Tarun Nakra, Rajiv Gupta, and Mary Lou

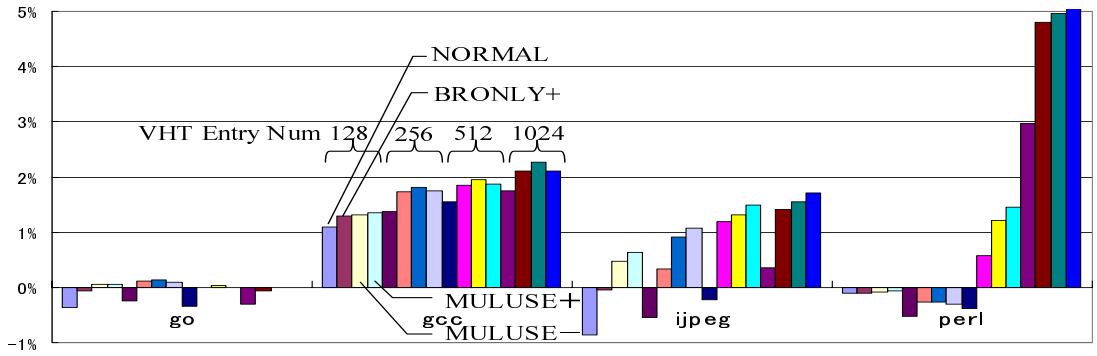

図 9 速度向上率

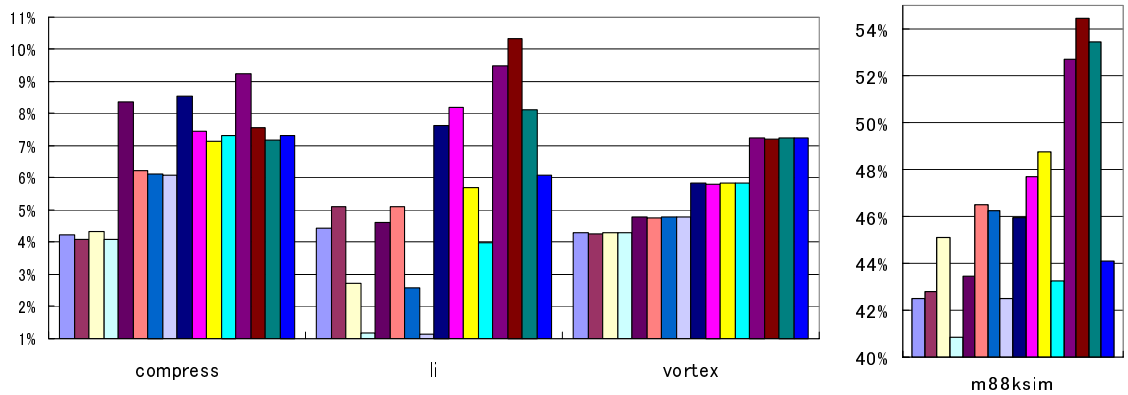

図 10 速度向上率 (続き)

Soffa. Global Context-Based Value Prediction. In *Fifth International Symposium on High-Performance Computer Architecture(HPCA-5)*, Jan 1999.

7) Bohuslav Rychlik, John W. Faistl, Bryon P. Krug, Albert Y. Kurland, John J. Jung, Miroslav N. Velev, and John P. Shen. *Efficient and Accurate Value Prediction Using Dynamic Classification.* Technical Report, CMuART-1998-01, Carnegie Mellon Microarchitecture Research Team, 1998.

8) Andre Seznec, Stephen Felix, Venkata Krishnan, and Yiannakis Sazeides. Design Tradeoffs for the Alpha EV8 Conditional Branch Predictor. In *29th International Symposium on Computer Architecture(ISCA-29)*, May 2002.

9) Eric Sprangle and Doug Carmean. Increasing Processor Performance by Implementing Deeper Pipelines. In *29th International Symposium on Computer Architecture(ISCA-29)*, May 2002.

10) Kai Wand and Manoj Franklin. Highly accurate data value prediction using hybrid predictors. *Proc. 30th Annual International Symposium on Microarchitecture(MICRO-30)*, Dec 1997.

11) Qing Zhao and David J. Lilja. *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.

12) 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.

13) 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.

14) 神代剛典, 佐藤寿倫, 有田五次郎. トレースレベル値予測におけるハードウェア量削減方法. 並列処理シンポジウム (JSPP), May 2002.

15) 飯塚大介, 小沢年弘, 坂井修一, 田中英彦. プロファイルを用いた値予測命令削減手法. 情報処理学会研究会報告 2000-HPC-82, Aug 2000.