計算機アーキテクチャ 146－１３
（２００２．２．１）

最近の値の局所性を利用するロード値予測手法

松本潤一[†] 片山清和[†]
安藤秀樹[†] 島田俊夫[†]

プログラムには最近の値の局所性が存在することが知られている．本論文ではこれを利用してロード値予測を行う方式を提案する．ロード命令と，それと同一の値を操作した先行ロード／ストア命令を関連付ける．予測値には関連付けされた先行ロード／ストア命令の値を用いる．ロード／ストア命令の値を結果キューと呼ばれる FIFO にインオーダで保持する．そして当該ロード命令が結果キューにロード値を格納するときにキューの前方を検索し，同一の値が存在すれば当該ロード命令とその先行ロード／ストア命令とを関連付ける．関連付けられた当該ロード命令と先行ロード／ストア命令との依存距離をテーブルに保持する．その依存距離を使い予測を行う．評価の結果，アドレスによる関連付けによる既存の方式と比較して，平均して 15.3%ポイント多くの依存関係を予測できることを確認した．また 2 レベル値予測器と組み合わせることで，ストライド値予測器と 2 レベル値予測器のハイブリッド値予測器よりも平均 12.0%ポイント多くのロード値を予測できることを確認した．

Load-Value Prediction using Recent-Value Locality

Junichi Matsumoto,[†] Kiyokazu Katayama,[†] Hideki Ando[†]
and Toshio Shimada[†]

It has been reported that programs have "recent-value locality". In this paper, we propose a mechanism that exploits the locality. In our mechanism, a load instruction is associated with other load/store instructions whose operated value is the same with the result of that load instruction. The values of load/store instructions are hold in a FIFO, which we call a result queue. When the value of a load is appended to the queue, the queue is associatively looked up. If the same value is found, its instruction is associated with the current load instruction. Then, the dependence distances between the current load instruction and its associated instructions are stored in a table. Our evaluation results show that our predictor can predict more dependence relations than conventional address-based predictors by an average of 15.3% point. In addition, we confirm a hybrid predictors that combines that our mechanism with a 2-level value predictor can predict more loads than the conventional hybrid predictor that combines a stride value predictor with a 2-level value predictor by an average of 12.0% point.

1. はじめに

プロセッサの性能向上を妨げている要因の 1 つに，命令間に存在するデータ依存がある．データ依存はプログラムの命令レベル並列性を制限しており，これを緩和する手法は重要である．値予測はこのデータ依存を緩和するのに効果的な手法であり，Lipasti らによって**最終値予測**[3] が提案されて以来，**ストライド値予測**[5]，**2 レベル値予測**[4),5] など様々な予測方式が提案されている．そのほとんどは静的命令自身の過去の値の振舞いと，現在の値の振舞いの間の相関を利用している．

これに対し静的命令に限らず，時間的に近い命令同士の値にも相関があることが，Jourdan らによって発見された[1]．本論文ではこの性質を**最近の値の局所性** (recent-value locality) と呼ぶ．本論文はこの最近の値の局所性を利用する値予測手法を提案する．本手法は結果を生成するあらゆる命令に適用可能であるが，本論文ではレイテンシが長く性能に与える影響が大きいロード命令のみに適用する．

本方式は結果キュー，信頼性テーブル，そして予測テーブルよりなる．結果キューで依存関係を検出し，信頼性テーブルでその依存関係の信頼性を保持する．依存関係の信頼性が閾値を越えたら予測テーブルに依存関係と信頼性を送る．そして予測テーブルより予測値を得る．

本論文の構成を述べる．まず 2 章で関連研究を紹介し，3 章で最近の値の局所性が存在する理由を述べる．そして 4 章でロード値予測手法の説明，及び実装した予測器の構成と動作説明をする．5 章で本手法の評価を行い，最後に 6 章で本論文をまとめる．

2. 関連研究

値予測は命令の実行結果を実行前に予測するものである．予測が正しく行われればデータ依存を削除することができ，命令レベル並列性は向上する．最も単純な予測方式は，Lipasti らによって提案された最終値予測[3] であり，同一の値が繰り返されると予測するものである．他の主要な予測方式

† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University

にストライド値予測[5]，2 レベル値予測[4),5)] などがある．ストライド値予測は値が一定の差分をもって遷移すると予測するものである．前回の値および前回の値と前々回の値の差分を保持しておき，命令がデコードされたときに前回の値と差分を加算して，それを今回の予測値とする．2 レベル値予測は過去の値の出現パターンに相関があるとして予測を行うものである．この方式では **VHT**(Value History Table) と **PHT**(Pattern History Table) を用いる．VHT は命令アドレスをインデクスとして，各エントリには命令の最近数回の結果とその出現の履歴パターンを保持する．PHT は履歴パターンをインデクスとし参照する．各エントリには VHT の 1 つのエントリに格納する値の数だけのカウンタを用意し，VHT の中のどの値が次に現れるかの頻度を記録して，これを予測に用いる．

値予測のほとんどの方式は，静的命令自身の過去の値の振舞いと現在の値の振舞いの間の相関を利用している．これに対し Jourdan らによって発見されたのは時間的に近い命令同士の間に相関があるというものである[1]．

Tullsen らはこの最近の値の局所性を利用して，Register Value Prediction(RVP) を提案した[2]．彼らは命令が定義するデスティネーション・レジスタの値は，既にレジスタ・ファイルのどこかに保持されている可能性が高いと述べている．またプロファイルに基づき，結果値が同一である近傍の命令の，デスティネーション・レジスタを同一のレジスタとし，デスティネーション・レジスタの実行前の値と新しく生成される値が等しいと予測する RVP を提案した．RVP では，予測値がレジスタ・ファイルの中に存在するため，予測値を保持するテーブルを必要としない．しかしこの手法は，レジスタに予測値が存在しなければならないという制限がある．しかし本論文の手法には，そういった制限はなく一般性がある．

こうした様々な値予測は，デスティネーションを定義する全ての命令に対して用いることができるが，ロード命令の値を予測する手法には，さらに**メモリリネーミング**という手法がある．Moshovos らの **Speculative Memory Cloaking**[6),7)]，佐藤の **2 ホップアドレス名前替え**[8] は RAW(Read-After-Write),RAR(Read-After-Read) の関係を予測することで，ロード値予測を行っている．過去の RAW,RAW の関係を基に現在の関係を予測できれば，ロード命令の予測値を先行ロード／ストア命令から得ることができる．

Moshovos らの Speculative Memory Cloaking では **DDT**(Dependence Detection Table) および **DPNT**(Dependence Prediction and Naming Table) を用いて，ロード命令と依存関係にあるストア命令を検出する．そしてこのロード命令とストア命令に共通のタグを割り当て，そのタグで **SF**(Synonym File) にアクセスする．ここにはストア命令が操作した値が入っており，ロード命令の予測値として使用される．Moshovos らの提案した機構は DDT,DPNT,SF のいずれも複雑な連想検索をしており，サイクル時間を増加させる恐れがある．

これに対し佐藤の 2 ホップアドレス名前替えは **LIST**(Load-Indexed Store Table), **SIVT**(Store-Indexed Value Table), **DIST**(Data-Indexed Store Table) の 3 枚のテーブルを用いている．DIST と LIST を用いてロード命令と，それと依存関係にあるストア命令を関連付け，LIST と SIVT を用いて，ロード命令の予測値を得る．佐藤の提案した機構は一切の連想検索を用いておらず，サイクル時間を増加させる恐れはないので，この点で Moshovos らの speculative memory cloaking に対して利点がある．

これらのメモリ依存は，アドレスによる関連付けを行うことで検出していた．これに対し，本手法では値による関連付けを行う．よってロード命令とそれに関連付けられた先行ロード／ストア命令の間に，RAW,RAR の関係が存在する必要はない．値に関する相関を利用し，全く別のアドレスをアクセスする命令とも関連付けることができる．

```
(a)              (b)              (c)              (d)
a[i]=c[j];       *ptr=...;        if(a==1)         if(x==y)
b[k]=c[j];       a=*ptr;           {                {
                 b=*ptr;            b=1;             ...;
                                   }                }
```

図 1 最近の値の局所性の例

3. 最近の値の局所性が存在する理由

我々は最近の値の局所性が存在するのは，大別して以下の 4 つの理由で起こると考えている．

- **同一アドレスの複数アクセス**
  図 1(a),(b) のような例を考える．(a) は同一の値を異なった配列に格納する例である．これは `c[j]` の値をロードし `a[i],b[k]` にストアしている．このような場合，同一のアドレスを何度もアクセスしていることになる．(b) のようにポインタを介して変数にアクセスする場合も同様である．この他にも save-restore,spill-in spill-out などレジスタの退避，復元の操作をする場合も同一のアドレスにアクセスする．このように同一アドレスを何度もアクセスすることが頻繁にあるとき，同一値が短時間に出現しやすくなる．

- **プログラムの意味上生じる同一値を持つ複数の変数**
  図 1(c) のような例では，変数 `a` と `b` は別の変数であるが，`b` の値は `a` の値によって決まる．即ち `a` が 1 の場合は必ず `b` も 1 となる．(d) のような例では条件分岐の判定のために，異なった変数同士を比較している．この場合 `x,y` の値は常に等しくなるとは限らないが，分岐方向は偏りが非常に大きいことが知られており，高い確率で等しくなる場合がある．このようにプログラムの意味上，同一値となるような複数の変数が存在することがある．このような変数の定義や使用は，お互い時間的に近い場合が多いため，同一値が短時間に出現する．

- **値の局所性の存在**
  Lipasti らによって発見された値の局所性は，静的命令の過去の値と現在の値の相関であるが，この静的命令が小さなループ内に存在する場合は，静的命令が短時間に何度も現われる．よって同一値が短時間に出現しやすくなる．

- **頻出値の存在**
  Zhang らは**頻繁な値の局所性** (frequent value locality)[9] が存在すると述べている．プログラムを通して頻繁に出現する値が存在する場合，その値が短時間に何度も出現する可能性は当然高い．

4. 最近の値を利用した値予測方式

本章では具体的に最近の値の局所性を，値予測に利用する手法を説明する．4.1 節で値による関連付け手法を説明し，4.2 節でそれを実現する予測器の構成及び動作を説明する．

4.1 値による関連付け手法

当該ロード命令の値とそれに先行するロード／ストア命

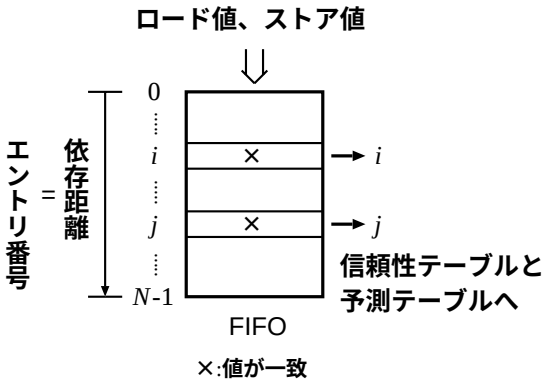

図 2 結果キュー

令の値を比較し，値が一致していれば，当該ロード命令とその先行命令を，同一値を操作するペアであるとする．そしてその 2 つの命令の間に存在する，ロード／ストア命令の数を数えて，2 つの命令間の**依存距離**とする．そしてこの依存距離をテーブルに保持しておく．このとき当該ロード命令までに至る実行パス (分岐履歴) によって分類して保持しておく．以上により，ペアの後方のロード命令がデコードされた時に，他方のロード／ストア命令の値を予測値とすることができる．

4.2 予測器の構成と動作

本方式は結果キュー，信頼性テーブル，予測テーブルより構成される．結果キューで依存関係を検出し，信頼性テーブルでその依存関係の信頼性を保持する．依存関係の信頼性が閾値を越えたら，予測テーブルに依存関係と信頼性を送る．そして予測テーブルより予測値を得る．それぞれの構成，動作を以下に詳しく説明する．

4.2.1 結果キュー

結果キューの構成を**図 2** に示す．結果キューは N 個のエントリからなりロード／ストア命令の値を FIFO で保持する．ストア命令なら保持するのみであるが，もしロード命令ならさらにキューを連想検索し，自分の値と先行の値が一致するかどうかをチェックする．もし一致するものが発見されれば，当該ロード命令とその先行命令との依存距離を求め，それを信頼性テーブルと予測テーブルに送る．このときの依存距離は FIFO の先頭を 0 エントリとすると，一致した先行命令のエントリ番号がそのまま依存距離となる．例えば当該ロード命令の値と第 i エントリの値が一致した場合は，依存距離は i となる．複数のエントリと一致した場合は，一致した全ての依存距離を信頼性テーブルと予測テーブルに送る．図 2 の例では第 i エントリと第 j エントリの値が当該ロード命令の値と一致している．この結果キューのエントリ数によって検出可能な依存距離の上限が決まる．図 2 の例では上限は N-1 である．しかし最近の値に局所性があるので，値が一致する命令同士はほとんどが非常に近い．よって距離をそれほど遠くまでみる必要はない．

4.2.2 信頼性テーブル

信頼性テーブルの構成を**図 3** に示す．信頼性テーブルはロード命令アドレスと分岐履歴をインデクスとして，依存距離の信頼性を保持する．各エントリはタグと N 個の 2 ビット飽和型リセッティングカウンタからなる．この N と結果キューのエントリ数の N は等しく対応している．結果キュー

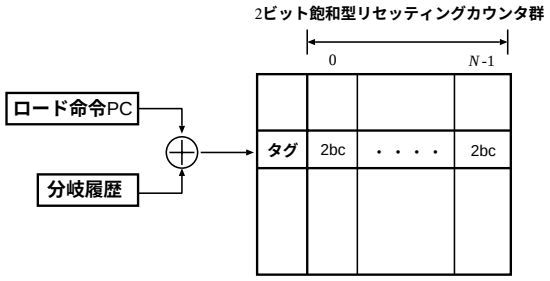

図 3 信頼性テーブル

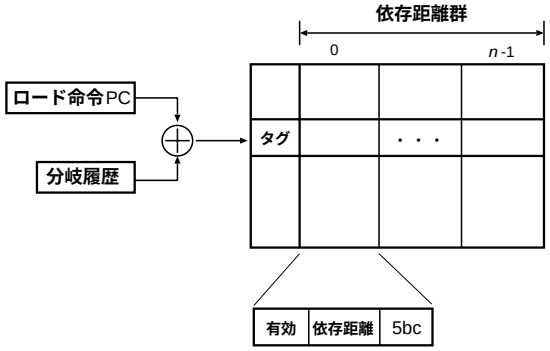

図 4 予測テーブル

から i,j という依存距離が送られてきたら，インデクスされたエントリの i 番目と j 番目のフィールドのカウンタをインクリメントする．それ以外のフィールドはリセットする．カウンタがあらかじめ決められた閾値を越え高信頼となったら，そのフィールド番号 (依存距離) を予測テーブルに送る．例えば i 番目と j 番目のカウンタ・フィールドをインクリメントした結果，i 番目のフィールドが閾値を越えたら，i という依存距離を予測テーブルに送る．

4.2.3 予測テーブル

予測テーブルの構成を**図 4** に示す．予測テーブルはロード命令アドレスと分岐履歴をインデクスとして，信頼性テーブルから送られた依存距離とその信頼性を保持する．各エントリは n 個の依存距離とそれぞれの信頼性，有効ビットからなる．信頼性テーブルで保持した信頼性は，全ての依存距離について，どの依存距離が確からしいかの見当をつけるためのものであった．それに対し予測テーブルの信頼性は，既に確からしい複数の依存距離の中で，最も確からしいものを選び出すためのものである．

予測は以下のようにしておこなう．まずロード命令のデコード時に予測テーブルを参照し，最も信頼性の高い依存距離を得る．依存距離が得られたらその分だけ前方のロード／ストア命令の値を予測値とする．

次に更新操作は以下のようにしておこなう．ロード命令のリタイア時に，第 1 に n 個の依存距離の信頼性カウンタを更新する．これは結果キューから送られてきた依存距離と n 個の距離を比較することで行う．第 2 に信頼性テーブルから送られてきた新しい依存距離とその信頼性を保持する．

更新時に既に保持している依存距離の数と，新しく保持する依存距離の数の和が，保持可能な依存距離の数を越えたら，古い依存距離のうち最も信頼性の低いものと置き換えて，新しい依存距離を保持する．

表 1　ベンチマークプログラムとその入力

プログラム	入力
compress95	30000 e 2231
gcc	genoutput.i
go	6 9 2stone9.in
ijpeg	specmun.ppm
li	train.lsp
m88ksim	ctl.in
vortex	vortex.in

5. 評　　価

最初に 5.1 で評価環境について述べる．次に予備評価として，値による関連付けがアドレスによる関連付けに比べ，どの程度予測精度に改善の余地があるかを述べる．その後値によって関連付ける手法 (以下，本手法)，アドレスによって関連付ける手法 (以下，アドレス手法)，2 ホップアドレス名前替えの手法 (以下，2 ホップ手法)，そして 2 ホップアドレス名前替えの手法を拡張し RAR の関係も検出できるようにした手法 (以下，2 ホップ拡張手法) の予測精度，予測できた割合を比較する．アドレス手法と 2 ホップ拡張手法の詳細は 5.1 で述べる．最後に本手法と 2 レベル値予測器のハイブリッド値予測器を構成し，これをストライド値予測器と 2 レベル値予測器のハイブリッド値予測器と比較する．

5.1 評 価 環 境

まずはじめに，アドレス手法と 2 ホップ拡張手法の詳細について説明する．そのあとでシミュレータについての説明をし，機構のサイズなどの諸設定を述べる．

本機構は値による関連付けを行うものであるが，この機構でアドレスによる関連付けを行うようにしたものがアドレス手法である．アドレス手法では結果キューに値のかわりに，アドレスを FIFO で保持する．そして結果キューを連想検索して，当該ロード命令のアドレスと一致するものが見つかった場合，その中で最も依存距離の近いもの 1 個のみを予測テーブルに保持する．アドレス手法では信頼性テーブルを必要としない．また予測テーブルに格納する依存距離は 1 個である．予測テーブルに格納されている依存距離は，新しい依存距離が送られてくるたびに置き換えられる．ロード命令がデコードされたときの予測方法は本手法と同じである．

2 ホップ拡張手法は，2 ホップ手法でストア命令が行っている DIST への登録動作を，ロード命令のリタイア時にも行うようにすれば実現できる．ロード命令のリタイア時に，データアドレスをインデクスとして DIST を参照する．そこに格納されている命令アドレスを，ロード命令アドレスをインデクスとして引いた LIST へ格納する．2 ホップ手法の動作はここで終了するが，2 ホップ拡張手法はさらにロード命令アドレスを DIST に上書きする．このようにすることで RAR の関係も検出できるようになる．

シミュレータは SimpleScalar Tool Set Version 3.0a の中の，インオーダ実行の命令レベル機能シミュレータに，本機構を組み込んだものを使用した．命令セットは MIPS R10000 を拡張した SimpleScalar/PISA である．使用したベンチマークプログラムは，SPECint95 の全 8 本である．それぞれの入力は**表 1** に示すとおりである．

評価においては結果キューのエントリ数を 128 とした．また信頼性テーブルのエントリ数を 4K，高信頼の閾値を 3 とした．そして予測テーブルのエントリ数を 4K，保持可能な依存距離の数を 2 個とした．アドレス手法の場合は，保持可能な依存距離の数は 1 個である．信頼性テーブル，予測テーブルのエントリと，予測テーブルの保持可能な依存距離の数を，これ以上増やしても予測精度に大きな変化は見られなかった．また信頼性テーブルの閾値は最も予測精度が高くなる値を選んだ．2 ホップ手法，および 2 ホップ拡張手法で用いる LIST,SIVT,DIST のエントリ数は論文 8) に準じて 4K とした．またストライド値予測器の VHT，2 レベル値予測器の VHT および PHT のエントリ数は，全て 4K とした．これ以上 VHT,PHT のエントリ数を増やしても，予測精度に大きな変化は見られなかった．

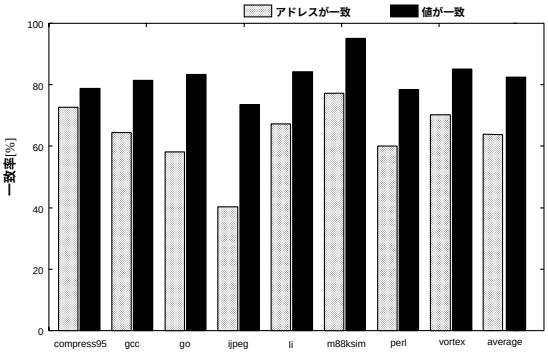

図 5　当該ロード命令からの距離が 128 以内に値またはアドレスが一致するロード／ストア命令が存在する割合

5.2 評 価 結 果

5.2.1 予 備 評 価

予備評価として，当該ロード命令からの依存距離が 128 以内の先行ロード／ストア命令に，当該ロード命令と値が一致する命令が少なくとも 1 個存在する確率，および当該ロード命令とアドレスが一致する命令が少なくとも 1 個存在する確率を調べた．これを**図 5** に示す．

このグラフより，当該ロード命令との距離 128 以内の先行ロード／ストア命令の少なくとも 1 つは，63.7% の確率でアドレスが一致するが，値については 82.5% の確率で一致することが分かる．全てのベンチマークで値の一致率の方がアドレスの一致率よりも上回っている．その理由は，アドレスが一致する命令のうちで当該ロード命令から最も距離が近いものは，それがストア命令であるなら RAW，あるいはそれがロード命令であるなら RAR の関係にあるので，必然的に値も一致するからである．つまりアドレスが一致する先行ロード／ストア命令が存在するときは，必ず値が一致する先行ロード／ストア命令も存在することになる．左側のアドレスの一致率の棒グラフと右側の値の一致率の棒グラフの開きは，アドレスは一致しないが，値だけは一致する先行ロード／ストア命令の存在により生じるものである．本手法では当該ロード命令と先行ロード／ストア命令との依存関係を値で関連付けるので，この開きが大きい程有利となる．

左側の棒グラフと右側の棒グラフの開きが最も大きいのは ijpeg の 33.3%ポイントで，逆に開きが最も小さいのは compress95 の 6.2%ポイントである．平均では 19.5% ポイントでありこの分だけ予測精度の向上が期待できる．

5.2.2 アドレスによる関連付けとの比較

図 6 に様々な手法を用いた際のロードの予測精度を示す．ここでいう予測精度とは，全ロード命令に対する予測が成功した割合のことをいう．棒グラフは左から順に本手法，アドレス手法，2 ホップ手法，2 ホップ拡張手法の予測精度である．各棒グラフは 3 つの部分からなり，下から順に予測が

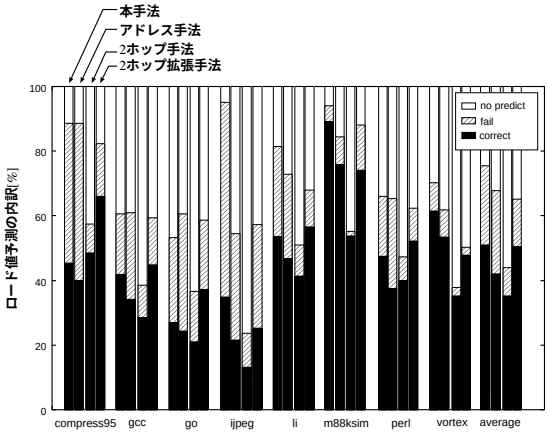

図 6　各手法の予測精度

成功した割合，予測が失敗した割合，予測しなかった割合である．

図 6 より分かるように，平均で最も予測精度の高い手法は，本手法と 2 ホップ拡張手法でともに 50.5% である．次に高い予測精度を示した手法はアドレス手法で，平均して 41.7% になる．最も低かった手法が 2 ホップ手法で 35.2% である．

この理由を予測精度が低い順に説明する．まず 2 ホップ手法は RAW のメモリ依存しか検出できない．一方アドレス手法は RAW,RAR の両方の依存を検出できるので，2 ホップ手法よりも多く予測することができる．しかし検出可能な依存距離に上限が存在するため，同じ RAW,RAR の関係を予測する手法でも，依存距離の上限が存在しない 2 ホップ拡張手法よりも予測できるロード命令は少ない．そして本手法は RAW,RAR の他にメモリ依存の関係にない命令同士を関連付けることもできるので，アドレス手法より多くのロード命令を予測できる．最後に本手法と 2 ホップ拡張手法の予測精度が等しかった理由は，本手法には検出可能な依存距離に上限が存在するというマイナス要因がある一方，2 ホップ拡張手法にはメモリ依存の関係にない命令同士を関連付けられないというマイナス要因があるためであると考えられる．

本手法とアドレス手法を比較した結果，値による関連付けの手法は，アドレスによる関連付けの手法よりも 8.8%ポイントの，予測精度の向上が得られると分かった．また 2 ホップ手法と 2 ホップ拡張手法を比較した結果，RAR 依存を検出できるようにすることで，15.3%ポイントの予測精度の向上が得られると分かった．本手法は RAR 依存の関係も検出可能なので，この点でも値による関連付けの手法は有効であると分かった．そして本手法と 2 ホップ拡張手法を比較した結果，RAW,RAR 依存の他に値は一致するが，アドレスは一致しない命令同士を関連付ければ，検出できる依存距離を制限しても，予測精度を高く保てることが分かった．

5.2.3　通常の値予測との比較

本手法と 2 レベル値予測の予測できるロード命令の差，およびストライド値予測と 2 レベル値予測の予測できるロード命令の差を調べた．**図 7** に測定結果を示す．左の棒グラフは本手法と 2 レベル値予測の，予測できるロード命令の比較であり，右の棒グラフはストライド値予測と 2 レベル値予測の，予測できるロード命令の比較である．左右の棒グラフはそれぞれ 4 つの部分からなる．まず左の棒グラフは下から順に，本手法と 2 レベル値予測の両方で予測できるロード命令

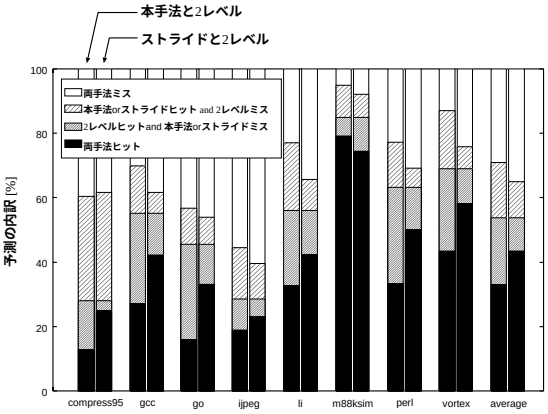

図 7　予測できたロード命令の内訳

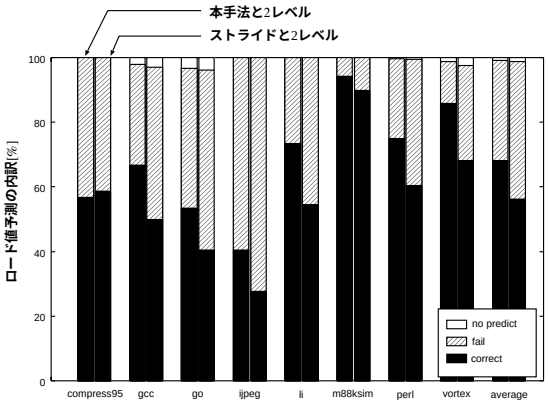

図 8　ロード値の予測精度の比較

の割合，本手法では予測できないが 2 レベル値予測なら予測できるロード命令の割合，2 レベル値予測では予測できないが本手法なら予測できるロード命令の割合，どちらでも予測できないロード命令の割合である．一方右の棒グラフは下から順に，ストライド値予測と 2 レベル値予測の両方で予測できるロード命令の割合，ストライド値予測では予測できないが 2 レベル値予測なら予測できるロード命令の割合，2 レベル値予測では予測できないがストライド値予測なら予測できるロード命令の割合，どちらでも予測できないロード命令の割合である．これをみて分かるように，本手法か 2 レベル値予測の，どちらか一方のみで予測できるロード命令の数は，ストライド値予測か 2 レベル値予測の，どちらか一方のみで予測できるロード命令の数よりも多い．よって本手法と 2 レベル値予測を組み合わせて値予測器を構成すれば，ストライド値予測と 2 レベル値予測を組み合わせたハイブリッド値予測器よりも，多くのロード値を予測できると考えられる．

図 8 に本手法と 2 レベル値予測を組み合わせた値予測と，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の予測精度の比較を示す．図 6 でも示したように，本手法で予測できるロード命令は全ロード命令の約 5 割である．しかしこれを 2 レベル値予測器と組み合わせると図 8 に示すように，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が全ロード命令の 56.2% を予測できるのに対し，本手法と 2 レベル値予測を組み合わせた予測器は全ロード命令の 68.2% を予測できると分かった．

6. ま と め

プログラムには最近の値の局所性が存在することが知られている．本論文ではこの性質を利用するロード値予測手法を提案した．本手法ではロード／ストア命令の操作した値を結果キューに保持しておき，ロード命令のリタイア時に同一の値を操作した先行ロード／ストア命令を検出し，検出された命令は信頼性テーブルを更新するとともに，信頼性が高ければ予測テーブルにその依存距離を保持する．予測テーブルに保持されている関連付けされた命令同士は，同一の値を操作すると予測する．評価した結果 SPECint95 の平均で，アドレスによる関連付けを行った場合より 8.8%ポイント多くのロード命令を予測することができた．また 2 レベル値予測器と組み合わせて用いることで，ストライド値予測器と 2 レベル値予測器のハイブリッド値予測器よりも 12.0%ポイント多くのロード命令を予測することができた．

謝辞 本研究の一部は，文部省科学研究費補助金基盤研究 (C)(課題番号 11680351) 及び財団法人大川情報通信基金の支援により行った．

参 考 文 献

1) Jourdan, S., Ronen, R., Bekerman, M., Shomar, B. and Yoaz, A.: A Novel Renaming Scheme to Exploit Value Temporal Locality through Physical Register Reuse and Unification, *Proc. 31st Int'l Symp. on Microarchitecture,* pages 216-225, November 1998.
2) Tullsen, D. M. and Seng, J. S.: Storageless Value Prediction Using Prior Register Values, *Proc. 26th Int'l Symp. on Computer Architecture,* pages 270-279, 1999.
3) Lipasti, M. H., Wilkerson, C. B. and Shen, J. P.: Value Locality and Load Value Prediction, *Proc. seventh Int'l Conf. on Architectural Support for Programming Languages and Operating Systems,* pages 138-147, October 1996.
4) Sazeides, Y. and Smith, J. E.: The Predictability of Data Values, *Proc. 30th Int'l Symp. on Microarchitecture,* pages 248-258, December 1997.
5) Wang, K. and Franklin, M.: Highly Accurate Data Value Prediction using Hybrid Predictors, *Proc. 30th Int'l Symp. on Microachitecture,* pages 281-290, December 1997.
6) Moshovos, A. and Sohi, G.: Streamlining Inter-Operation Memory Communication via Data Dependence Prediction, *Proc. 30th Int'l Symp. on Microarchitecture,* pages 235-245, December 1997.
7) Moshovos, A. and Sohi, G.: Read-After-Read Memory Dependence Prediction, *Proc. 32nd Int'l Symp. on Microarchitecture,* pages 177-185, November 1999.
8) 佐藤 寿倫:2 ホップアドレス名前替えを用いたロード命令の投機的実行, 情報処理学会論文誌, Vol. 40, No. 5, pages 2109-2118, November 1999.
9) Zhang, Y., Yang, J. and Gupta, R.: Frequent Value Locality and Value-Centric Data Cache Design, *Proc. ninth Int'l Conf. on Architectural Support for Programming Languages and Operating Systems,* pages 150-159, November 2000.
10) Burger, D. and Austin, T. M.: The SimpleScalar Tool Set, Version 2.0, Technical Report CS-TR-97-1342, University of Wisconsin-Madison, June 1997.
11) Reinman, G. and Calder, B.: Predictive Techniques for Aggressive Load Speculation, *Proc. 31st Int'l Symp. on Microarchitecture,* pages 127-137, December 1998.