データ値予測とアドレス予測を組み合わせたデータ投機実行

佐　藤　寿　倫†

近年，命令の演算結果を予測してデータ依存関係にある命令を投機実行することが検討されている．本稿では，ロード命令の読み出す値を予測してデータ依存を投機的に解消するプロセッサを検討している．ロードデータ値の予測だけでなくデータアドレスを予測することで，より積極的な投機実行を可能にし，さらにデータ値予測失敗によるペナルティの軽減を検討している．ロードデータ値を予測不可能な時に限り，データアドレスの予測による投機実行を行なう．我々はこの予測器を協調型予測器 (cooperative predictor) と呼んでいる．さらに，予測されたデータと，予測されたデータアドレスを用いて読み出されたデータとを比較することで，投機的にデータ値予測の成功を判定することを提案する．この方法を投機的判定 (speculative verification) と呼んでいる．投機的判定によって，誤って予測されたデータ値を早期に発見でき，その結果ミスペナルティを軽減できる．シミュレーションの結果，提案する手法は，従来のデータ値予測機構に比較して，さらに積極的にデータ依存の投機実行を可能にし，命令レベル並列度を向上できることが確認できている．

Combining Data Value Prediction with Data Address Prediction

Toshinori Sato†

In this paper, we propose to combine load value prediction with load address prediction in order to reduce misprediction penalty. When a load value predictor can not predict a load value, it is supported by a load address predictor and data dependences can be speculated. We call this predictor a *cooperative predictor*. Furthermore, we propose to verify a value prediction by comparing the predicted value with the one obtained using the predicted load address. We call this scheme *speculative verification*. With the help of the speculative verification, a recovery action caused by a misprediction is initiated earlier than the time when the misprediction is really detected. Using a cycle-by-cycle simulator, we have evaluated the cooperative predictor and the speculative verification and found that data dependences are more aggressively speculated to exploit instruction level parallelism.

1. まえがき

近年，データ依存関係を投機的に解消し，より大きな命令レベル並列度 (instruction level parallelism) を引き出そうとする研究が注目されている．データ値予測器を用いて命令の演算結果を予測することで，その命令と依存関係にある命令を同時に機能ユニットにディスパッチでき，その結果命令レベル並列度を向上できる．データ値予測器は大きく二種類に分類される．一つは全ての命令の演算結果を予測する予測器であり[4),7),17),21)]，もう一つはロード命令の読み出すデータだけを予測する予測器である[5),7),9),15),19),22)]．本稿では後者に注目する．

データ値予測器の性能を向上するためには，以下の 2 点を検討する必要がある．ひとつは予測精度であり，もう一つは予測率である．予測精度とは，データを予測できたロード命令の中でその予測が正しかった命令の割合である．一方予測率とは，全てのロード命令の中で正しくデータを予測できた命令の割合である．予測に失敗するとペナルティを被るので，予測精度は高いことが望ましい．さらに，命令によって予測の容易なものと困難なものとが存在するため，予測の容易な命令だけを予測の対象とすることが望ましい．しかし，予測の対象を限定すると予測率を低下させてしまう．一方で，プロセッサの性能向上に寄与しているのは予測精度よりもむしろ予測率であることがわかっている[12)]．したがって，予測率を低下させることなく予測精度を改善しなければならない．

上記の目的を満たすために，われわれはデータ値予測機構とデータアドレス予測機構を併用することを提案する．アドレス予測はデータ予測よりも予測精度が高いと思われるが，予測に成功した際のプロセッサ性能向上への寄与は小さいと考えられる．したがって，データ予測が不可能な時に限り，アドレス予測を実施することが望ましい．つまり，アドレス予測器はデータ値予測器を補助する目的で使用される．われわれは，このような予測器を協調型予測器 (cooperative predictor) と呼んでいる．さらに，われわれは予測されたアドレスから読み出されたデータを用いて，データ予測の成功判定を行なうことを検討している．われわれはこの方法を投機的判定 (speculative verification) と呼んでいる．これによって，データ予測の失敗を早期に検出でき，その結果ミスペナルティを軽減できる．

本稿は以下の構成である． 2節で従来の関連研究をまとめる． 3節で協調型予測器と投機的判定を説明する． 4節で評価環境を述べ， 5節でシミュレーション結果をまとめる． 6節は結論である．

2. 関連研究

命令の演算結果を予測する手法には様々な提案がある[4),5),7),9),15),17),19),21),22)]．これらは，予測対象の命令

† 株式会社東芝 セミコンダクター社 マイクロエレクトロニクス技術研究所
Toshiba Microelectronics Engineering Laboratory
toshinori.sato@toshiba.co.jp

に応じて大きく二つに分類される．ひとつは全ての命令の演算結果を予測する機構であり[4),7),17),21)]，もうひとつはロード命令が読み出すデータだけを予測する機構である[5),7),9),15),19),22)]．Lipasti の提案している最終値型予測器 (last-value predictor)[7)] は最も初期に提案された予測機構の一つであり，値の局所性を利用して予測を行なう．ある命令が実行されるとその結果を保持しておき，将来同じ命令が再び出現した時には，保持されている値を予測値として利用する．オペランド・プリフェッチ・キャッシュ (operand prefetch cache)[22)] も同様に動作する． Gabbay らの提案しているストライド型予測器 (stride-based predictor)[4)] は，最後の演算結果だけでなく，連続する二つの演算結果の差も同時に保持している．次に同じ命令が出現した時には，両者の和を予測値として利用する．

予測精度を改善するために，これまでいくつかの混成型予測器 (hybrid predictor) が提案されている[17),21)]．混成型予測器は複数の予測器と，それらの予測器の中から一つを選択する選択器とから構成される．予測精度を向上させる他の方法としては，プロファイルを用いた方法がある[4)]．プロファイルを用いて各命令を予測の容易さで分類し，予測容易な命令だけを予測の対象にする．この情報はコンパイラによって付加され，プロセッサに渡される．ロード命令のデータ値予測精度を向上させるには，さらに他の方法がある．それはストア命令の情報を利用することである[9),15),19)]．この方法を本稿ではリネーミング型予測器 (renaming-based predictor) と呼ぶことにする．リネーミング型予測器を用いることで，データアドレスが計算される前に，ストアされるデータが投機的にロード命令にフォワードされる．本稿で利用するロード値予測器は 15) に基づいている．

一方，データアドレスを予測する方法にも多くの提案がある[5),14),16),22)]．これらの提案の基本的な動作は非常に似ている．予測器は，最後に計算されたデータアドレスと，連続する二つのデータアドレスの差を保持し，両者の和を次のデータアドレスとして利用する．本稿で利用するアドレス予測器は 14) に基づいている．

これまでデータ値予測とアドレス予測を併用した予測器の検討はほとんどない． Gonzalez らの提案している予測器[5)] はデータ値とデータアドレスの両者を予測するが，予測されるデータ値は予測アドレスから読み出されたデータであり，二つの予測器が独立に動作するわけではない．一方，協調型予測器を構成するリネーミング型予測器では，予測データは予測アドレスを用いて読み出されるわけではない．さらに我々は，予測アドレスを用いてデータ予測の成功判定を行なう投機的判定も検討している．ごく最近 Reinman ら[10)] は協調型予測器と投機的判定に良く似た機構を検討しているが，我々の検討とは独立に無関係に行なわれたものである．

3. 提 案 機 構

協調型予測器は，それを構成するデータ値予測器とアドレス予測器の種類によらない．本節では，まず本稿で用いるデータ値予測器とアドレス予測器を簡単に説明し，つづいて協調型予測器と投機的判定の提案を行なう．

3.1 ロード値予測器

本稿で用いるリネーミング型予測器は， 3 つのテーブルから構成される．それらは， data-indexed store table (DIST)， store-indexed value table (SIVT)，そして load-indexed store table (LIST) である． DIST はデータアドレスを用いて参照され，ストア命令によるメモリ参照の履歴を保持している．ロード命令が DIST を参照すると，同じメモリ領域を参照しているロード命令とストア命令のリンクが形成される． LIST はロード命令アドレスを用いて参照され， DIST が形成したリンクを保持する．ロード命令のフェッチと同時に LIST も参照され，参照されたエントリに保持されているリンクを指し示すタグが獲得される． SIVT はそのタグを用いて参照され，そのタグが指し示すストア命令によってメモリに書き込まれたデータを保持している．これらのテーブルを参照することで，ロード命令はデータアドレスを計算する以前にデータを獲得することが可能になる．詳細な説明は 15) を参照されたい．

3.2 ロードアドレス予測器

本稿で用いるアドレス予測器は参照アドレス予測表 (reference prediction table: RPT)[2)] を応用している．RPT はキャッシュメモリと似た構成をしており，ハードウエアプリフェッチングを実行する目的で Chen らによって提案された． RPT はメモリ参照の履歴を保持することでアドレスを予測する． RPT は命令アドレスを用いて参照され，以下の情報を保存している．すなわち，最も最近参照したデータアドレス，データアドレスのストライド，そして予測の可否を示す状態である．ストライドは同じ命令が最近二回に参照したデータアドレスの差から求められる．予測可否状態は過去の履歴がエンコードされて保存されており，次回の予測が可能かどうかを示している．予測アドレスは，最終値アドレスとストライドとの和で求められる．機構の詳細な説明は 14) を参照されたい．

3.3 協調型予測器

もし，データ値予測器がロードされるデータを予測不可能な時にアドレス予測器を用いれば，予測率を向上させることが出来る．データアドレスを予測し，その予測アドレスを用いてデータを読み出すことでデータ依存を投機的に解消するわけである．我々は，このようなアドレス予測器に補助されるデータ値予測器を，協調型予測器と呼んでいる．協調型予測器は混成型予測器[17),21)] に必要な予測器選択機構を持たないため，ハードウエア規模を小さくできるという利点がある．

ここで「予測不可能」と「予測失敗」の違いに注意されたい．「予測不可能」とは，なんらかの理由でデータ値あるいはアドレスが得られない場合であり，投機を行なえない．一方「予測失敗」とは，誤った予測値が得られた場合であり，失敗する投機を行なう可能性がある☆．

図 1 を用いて，協調型予測器がデータ値予測器やアドレス予測器よりも予測率が高い理由を説明する．図 1(i) においては，同じメモリ領域を参照する一組のストア命令とロード命令が同じループ内で交互に出現する．一方図 1(ii) においては，ストア命令とロード命令は異なるループに属している．リネーミング型のデータ値予測器[15)] は (i) の場合に有効に予測できるが， (ii) の場合では予測困難である．一方，ストライド型のアドレス予測器[14)] は (ii) の場合を得意とする．また，配列データの場合にはアドレス予測器が適しているが，ポインタを介したデータの場合はアドレス予測は困難でありデータ値予測器の方が適している．したがって，これら二つの予測器は相補的に動作し，これらを組み合わせた協調型予測器は両者の予測率を足し合わせた予測率を期待できる．

協調型予測器の動作を例を用いて説明する．**図 2** は命令シーケンスの例であり，命令 `I1` と `I2` との間にはデータ依

☆ 演算器数の制約などにより，誤った予測値を用いた命令ディスパッチが起こらない場合がある．

time →

I1		ad←r10+4	r11←mem[ad]	
I2				r13←r11+r12

図 3　投機実行を行なわない場合

time →

I1		ad←r10+4	r11←mem[ad]	*compare* r11
I2	r13←**r11**+r12			

図 4　データ値予測に成功した場合

time →

I1	**r11**←mem[**ad**]	ad←r10+4 *compare* ad		
I2		r13←**r11**+r12		

図 5　アドレス予測に成功した場合

time →

I1	**r11**←mem[**ad**]	ad←r10+4 *compare* ad	r11←mem[ad]	
I2		r13←**r11**+r12 *invalidated*		r13←r11+r12

図 6　アドレス予測に失敗した場合

```
loop:    :              loop1:   :
         store                   store
         :                       :
         load                    bne loop1
         :              loop2:   :
         bne loop                load
                                 :
                                 bne loop2
```

(i)　　(ii)

図 1　2 種類のループの例

```
I1:   load   r11 ← r10(4)
I2:   add    r13 ← r11 + r12
```

図 2　命令シーケンスの例

存関係が存在する．ここで，レジスタ $r12$ は確定しているとしレジスタ $r10$ は 1 サイクル遅れて確定するものと仮定する．データ依存関係を投機的に解消しない場合では，この命令シーケンスは**図 3** のように実行される．**図 4** ～**図 6** は，命令 I2 が投機的に実行される場合を示している．命令 I1 が演算結果 $r11$ を正しく予測できた時には，命令シーケンスは図 4 のように実行される．命令 I2 は予測された値 **r11** を用いてディスパッチされる．図において予測値は強調印字されている．実際のデータが読み出されると実際の値と予測値とを比較し，この場合は一致しているので投機実行が終了する．この例では，実行サイクルは 3 サイクル短縮されている．

命令 I2 がロードデータを予測不可能な場合には，命令シーケンスは図 3 と同様に実行される．もしデータ値予測器だけでなくアドレス予測器も利用可能であれば，すなわち協調型予測器を利用可能であれば，命令シーケンスはアドレス予測の成功失敗に応じて**図 5** あるいは図 6 のように実行される．この場合，ロードデータは予測アドレス **ad** を用いて読み出されている．図において，予測アドレスとそれを用いて読み出されたデータは強調印字されている．予測アドレスと実際のアドレスとの比較は，実際のアドレスの計算と同時に実行できる[3]．したがって，二つのアドレスが一致する場合，すなわちアドレス予測が成功した場合には，実際のアドレスを用いてメモリにアクセスする必要はない．命令 I2 は予測アドレスを用いて読み出されたデータを使って投機実行されている．アドレス予測が成功した場合は，図 5 のように投機実行は終了する．一方予測に失敗すると，図 6 のように命令 I2 は破棄され再発行される．

まとめると，データ値予測器がロードデータを予測不可能な場合でも，協調型予測器は投機実行が可能で，上記の例では，予測に成功した場合には実行が 2 サイクル短縮され，予測に失敗した場合でも実行サイクルは増加しない．

3.4　投機的判定

前節で提案した協調型予測器は，データ値を予測不可能な時に限り有効に働く．データ値の予測に失敗した場合には，データ値予測器と同様にミスペナルティを被る．このペナルティを軽減するために，投機的判定を用いて協調型予測器を拡張することを検討する．投機的判定とは，予測データ値を予測アドレスを用いて読み出されたデータと比較することで，データ値予測の成功失敗の判定を行なう方法である．もし両者が一致すれば，データ値予測は従来通りに実施される．もし一致しなければ，データ値予測失敗が投機的に検出され，そのロード命令は破棄され再発行される．投機的判定を用いることで，データ値予測の失敗を早期に検出でき，その結果ミスペナルティを削減できる．

図 7 と**図 8** に投機的判定の実施例を示す．命令シーケンスには前節の例と同じものを用いた．データ値の予測に失敗した場合には，命令シーケンスは図 7 のように実行される．予測値は実際の値と比較され，両者が一致しない場合，すなわち予測に失敗した場合には，命令 I2 は破棄および再発行され次のサイクルで実行される．この場合，プロセッサは予測失敗のミスペナルティを被り，実行サイクルが 1

time →

I1		ad←r10+4	r11←mem[ad]	*compare* r11	
I2	r13←**r11**+r12			*invalidated*	r13←r11+r12

図 7 データ値予測に失敗した場合

time →

I1	**r11**←mem[**ad**]	ad←r10+4 *compare* **r11**			
I2	r13←**r11**+r12	*invalidated*	r13←r11+r12		

図 8 投機的判定に成功した場合

time →

I1	**r11**←mem[**ad**]	ad←r10+4 *compare* **r11** *compare* ad	r11←mem[ad]	*compare* r11	
I2	r13←**r11**+r12				

図 9 投機的判定をキャンセルした場合

サイクル長くなる．これは，命令 I2 の再発行が 1 サイクル遅れるためである．投機的判定を用いて予測値の比較を早期に行うと，ペナルティは隠蔽され実行サイクルは 2 サイクル削減される．その様子を図 8 に示す．図において，予測値と予測アドレスを用いて読み出された値との比較は強調印字されている．

ここで問題となるのは，データ値予測に成功しているにも関わらずアドレス予測に失敗した場合をどのように扱うかということである．投機的判定では，このような場合を特別には扱っていない．アドレス予測の成功失敗が判定できるタイミングによって，ペナルティを被る時と被らない時とが存在する．もしこの判定が投機的判定よりも早ければ，投機的判定はキャンセルされペナルティを被らない．誤った投機的判定をキャンセルする例を**図 9** に示す．逆の場合はキャンセルが間に合わないので，ペナルティを被ってしまう．この時には，正しく投機実行されているにも関わらずロード命令が破棄されて再発行されてしまう．ただし，アドレス予測精度はデータ予測精度よりも十分大きいと考えられるので，多くの場合は投機的判定は成功し，十分ミスペナルティは補われると期待される．

まとめると，投機的判定を行なうと，データ値の予測が出来ない場合だけでなく予測に失敗する場合にも，アドレス予測によってミスペナルティを軽減できる．ただし，誤った投機的判定によるペナルティを被る場合が存在する．

3.5 LIST と RPT の共用

協調型予測器は，データ値予測器およびアドレス予測器よりもハードウエア規模が大きい．そこでハードウエア量を軽減するために，一つのテーブルで LIST と RPT を共用することを考える．共用されたテーブルは，ロードデータの予測と同時にロードアドレスの予測にも用いられる．このテーブルを**図 10** に示す．この共用により，二つのテーブルのうち一方のタグアドレスフィールドが不要になり，ハードウエア量を削減できる．

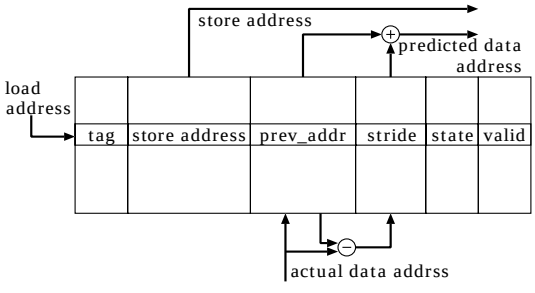

図 10 共用 LIST/RPT

4. 評価環境

本節では，シミュレーションに用いたプロセッサのモデルとベンチマークプログラムを紹介し，提案手法の評価方法について説明する．

提案手法の評価には，SimpleScalar ツールセット[1] を用いた．SimpleScalar/PISA アーキテクチャは MIPS アーキテクチャに基づいており，サイクルレベルのシミュレータが SPARCstation 上で動作する．基本となるプロセッサモデルはアウト・オブ・オーダー実行を行なうスーパースカラ・プロセッサである．アウト・オブ・オーダー実行を実現するためにはレジスタ更新ユニット (register update unit)[18] を採用している．6) での結果に基づいて，基本モデルの構成は**表 1** にまとめた通りとした．

LIST，SIVT，DIST はダイレクトマップ方式とし，それぞれのエントリ数は特に断らない限り 4096 とした．この時のハードウエアは，タグアレイ部も含めてそれぞれ 25.5KB，25.5KB，24.5KB となる．同様に RPT も 4096 エントリのダイレクトマップで構成し，ハードウエアはタグ部を含めて 34.5KB となる．したがってこれらのハードウエアは合わせて 110KB となる．LIST と RPT を共用すると 9KB 削減可能であるが，それでも 100KB 以上のハードウエア量となる．このため，実世界に適応するためにはハードウエア量削減が必要となる．そのための検討はすでに報告例[8),11)] もあり，またキャッシュメモリのハードウエア量削減のための方法[20)] も適用可能と思われる．そこで本稿では，LIST と RPT の共用以上のハードウエア量削減は検討しない．

データ投機実行に失敗した時にはプロセッサの状態を回復させる必要がある．それには命令再発行[14)] を用いた．

評価には SPECint95 ベンチマークプログラムを用いた．各プログラムの入力ファイルには，SPEC から配布されている `test` ファイルを使用した．ウイスコンシン大学が配布

表 1 プロセッサモデル

命令フェッチ幅	8 命令
分岐予測機構	512 エントリ 2 ウエイ・セットアソシアティブ BTB, gshare 法, 12 ビット BHR, PHT4096 エントリ, デコードステージで投機的に更新, リターン・アドレス・スタック 8 エントリ, ミスペナルティ 3 サイクル
命令ウインドウ	命令キュー 64 エントリ, ロード・ストアキュー 8 エントリ
命令ディスパッチ幅	8 命令
命令コミット幅	8 命令
機能ユニット	5 iALU's, 1 iMUL/DIV, 4 Ld/St, 2 fALU's, 2 fMULs, 2 fDIV/SQRT's
レイテンシ (全 / 投入間隔)	iALU 1/1, iMUL 3/1, iDIV 35/35, Ld/St 2/1, fADD 2/1, fMUL 3/1, fDIV/SQRT 6/6
レジスタファイル	32 ビット整数レジスタ 32 本, 32 ビット浮動小数点レジスタ 32 本
命令キャッシュ	64K 4 ウエイ・セットアソシアティブ, ライン幅 32 バイト, 4 ポート, ミスペナルティ 6 サイクル
データキャッシュ	64K 4 ウエイ・セットアソシアティブ, ライン幅 32 バイト, 4 ポート, ライトバック, ノンブロッキング・ロード, ミスペナルティ 6 サイクル
二次キャッシュ	共用, 256K 4 ウエイ・セットアソシアティブ, ライン幅 64 バイト, ミスペナルティ 32 サイクル

しているオブジェクトファイルを用いて実行した[1)]．各プログラムは終了まであるいは最初の一億命令を実行した．

5. シミュレーション結果

本節でシミュレーション結果を紹介する．評価には 1 サイクル当たりの完了命令数 (committed instructions per cycle: IPC) を用いた．カウントされる命令には nop 命令を含んでいない．プロセッサの性能向上は，増加した IPC を基本モデルの IPC で割った増加率で示す．

まず，データ値予測を行なった場合の予測率とプロセッサの性能向上との関係を評価する．つづいて本研究の動機を検証するために，ロードデータ値予測器とロードアドレス予測器の予測率を評価する．さらに，協調型予測器と投機的判定の効果を評価する．最後に，LIST と RPT を共用した際の効果を評価する．

5.1 予備評価

図 11 に理想的なデータ値予測器を用いた時の予測率と性能向上率との関係を示す．この予測器はロード命令をランダムに選択してデータを予測し，100% の精度でデータを予測できる．各プログラムに対応した 6 本のグラフは，左から順に予測率を 30% から 80% まで変化させた時の性能向上率を表している．明らかに，予測率が向上すると性能向上率も増加している．また，性能を平均で 10% 向上させるためには予測率は 70% 以上でなければならない，すなわち全ロード命令の 70% 以上が正しくデータを予測出来なければならないことがわかる．

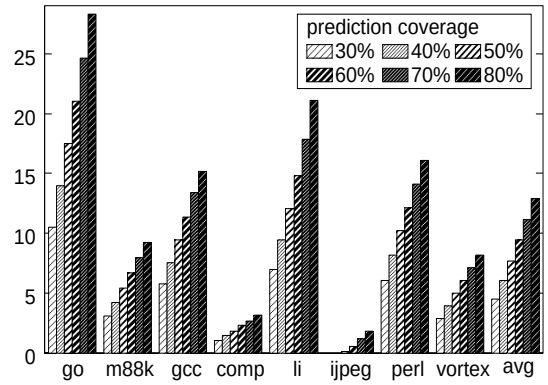

図 11 (%) 理想的な予測器の性能向上率

図 12 は本稿で用いているロードデータ値予測器の予測率を示している．各グラフは 3 つの部分に分けられている．下部 (黒) は予測に成功したロード命令の割合を示している．中部 (白) は予測に失敗した割合を示している．そして上部 (灰) は予測不可能だった割合を示している．したがって，グラフの黒の部分が予測率を表している．予測率は平均で約 50% であり，図 11 を考慮するとプロセッサの性能向上率は 10% に満たないと予想できる．すなわち，予測精度を下げることなく予測率を向上させる必要がある．

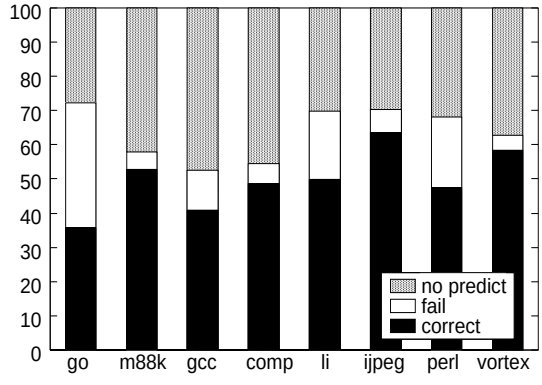

図 12 (%) データ値予測器の予測率

図 13 は本稿で用いられているロードアドレス予測器の予測率を示している．図 12 と同様に各グラフは 3 つの部分に分けられている．容易にわかるように，`124.m88ksim`, `129.compress`, `132.ijpeg` は予測率が非常に高い．したがってこれらのプログラムでは，データ値予測器が予測不可能な場合にはアドレス予測器が効果的に補助できると予想される．残りのプログラムは比較的予測率は小さいが，3.3節で説明したように，もしデータ値予測器の予測するロード命令の分布とアドレス予測器の予測する分布とが異なれば，お互いに補い合うことが期待され予測率は向上す

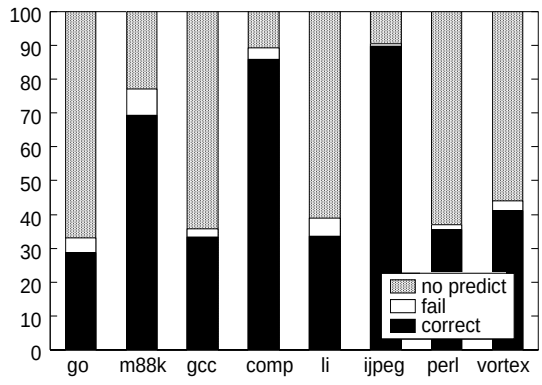

図 13 (%) アドレス予測器の予測率

ると思われる．

5.2 協調型予測器

図 14 は協調型予測器を用いた時のプロセッサの性能向上率を表している．各プログラムに対応した 3 つのグラフはそれぞれ左から順に，データ値予測器 (Value)，アドレス予測器 (Adrs)，協調型予測器 (Coop) を用いた時の性能向上率を表している．全てのプログラムで，協調型予測器が最もプロセッサの性能を向上させている，すなわち最も命令レベル並列度を引き出していることがわかる．協調型予測器はプロセッサの性能を平均 5.2% 向上しており，これはデータ値予測器と比べて 1.8% 改善されている．

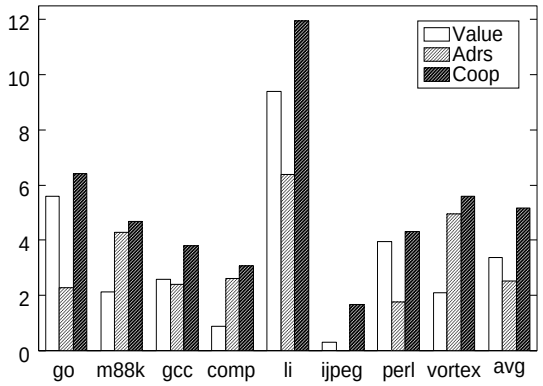

図 14 (%) 協調型予測器の性能向上率

図 15 は協調型予測器の予測率を表している．データ値を予測不可能な時に限りアドレスを予測していることに注意されたい．したがって，図 12 の上部に示されるデータ値予測不可能の場合のみがアドレス予測の対象となって，さらに 3 つの部分に分けられる．つまり図 15 の各グラフは 5 つの部分に分けられている．最も下の部分はデータ値の予測に成功した割合であり，次の部分はデータアドレスの予測に成功した割合である．次の部分はデータ値の予測に失敗した割合であり，次はアドレスの予測に失敗した割合である．最後に最上部はデータ値もアドレスも予測不可能だった割合である．図 15 の v:correct と v:fail の部分は，それぞれ図 12 の correct と fail に等しく，図 15 の a:correct, a:fail, no predict の和は図 12 の no predict に等しいことに注意されたい．協調型予測器の場合は，図 15 の下から二つの部分の和が予測率を表しており，平均で 68.1% である．これはデータ値予測器の場合と比較すると 18.4% 改善されている．再び図 11 を参照すると，68.1% の予測率であれば 10% 近い性能向上を達成できているはずである．しかし，性能向上率は 5.2% に過ぎない．これは以下の理由による．第 1 に図 11 の予測器は理想的なのでミスペナルティを被らない．第 2 に，現実の予測器は予測容易なロード命令だけが予測可能であるが，理想的な予測器は予測困難なロード命令も予測可能である．予測困難な命令を予測できる場合の方がプロセッサの性能向上に大きく寄与できると考えられるので，予測率が同じ場合では現実の予測器よりも理想的な予測器の方がプロセッサ性能を大きく向上できる．

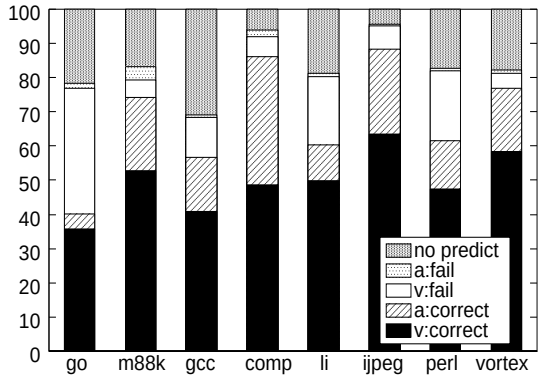

図 15 (%) 協調型予測器の予測率

さらに，LIST の容量を倍にしたデータ値予測器と協調型予測器とを比較する．協調型予測器のハードウエア規模はデータ値予測器のそれよりも大きいので，容量の大きなデータ値予測器との比較をすることで公平な評価が出来る．LIST に追加された 4096 エントリに相当するハードウエアは，協調型予測器で用いられるアドレス予測器 (すなわち RPT) のハードウエア規模とほぼ同じであることに注意されたい．すなわち，LIST の容量を倍にしたデータ値予測器のハードウエア規模は，協調型予測器のそれとほぼ同じである☆．**図 16** にシミュレーション結果を示す．各プログラムに対応する 3 つのグラフは，左から順にエントリ数 4096 の LIST を用いたデータ値予測器 (Value(4096))，エントリ数 8192 の LIST を用いたデータ値予測器 (Value(8192))，そして協調型予測器 (Coop(4096)) を用いた時のプロセッサ性能の向上率を表している．容易にわかるように，LIST の容量を倍にしてもプロセッサ性能はほとんど変化せず平均で 0.1% 改善されるに過ぎない．したがって，同じハードウエア規模で比較した場合でも，協調型予測器の方がデータ値予測器よりも有効であることが確認できる．

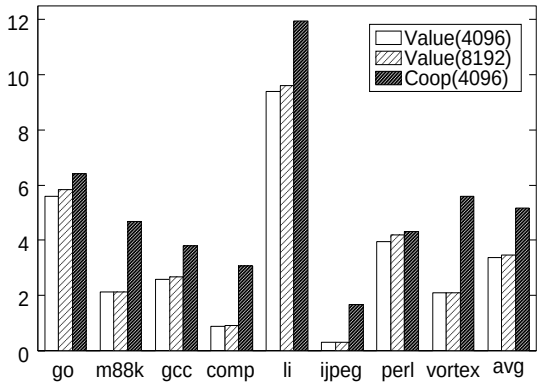

図 16 (%)LIST 容量を増加した時の性能向上率

以上をまとめると，協調型予測器は予測率を改善し，その結果プロセッサ性能を向上できる，すなわち命令レベル並列度を向上できることが確認できた．

5.3 投機的判定

図 17 に投機的判定を採用した協調型予測器を用いた時の，プロセッサの性能向上率を示す．各プログラムに対して 3 つのグラフは左から順にそれぞれ，データ値予測器 (Value)，投機的判定を行なわない協調型予測器 (Coop)，そして投機的判定を行なう協調型予測器 (Spec) を用いた時の，プロセッサ性能の向上率を表している．6 つのプログラムで投機的判定は協調型予測器の効果を改善し，より多

☆ 両者の差は 9KB であり決して小さな値ではないが，全体のハードウエア量に対する割合は 8% に過ぎない．

くの命令レベル並列度を引き出していることがわかる．投機的判定を行なわない場合と比較して平均 0.32% 改善し，全体として平均5.5% プロセッサの性能を向上させている．

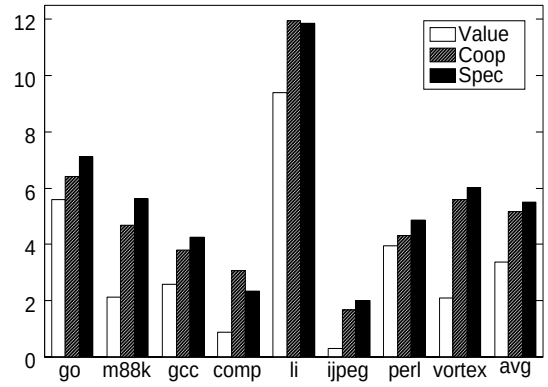

図 17 (%) 投機的判定を行なう場合の性能向上率

図 18 は，投機的判定を行なう場合の協調型予測器の予測率を表している．各グラフは，データ値を正しく予測した (`v:c`) か，誤って予測した (`v:f`) か，あるいは予測不可能だった (`v:n`) かの 3 つの場合と，アドレスを正しく予測した (`a:c`) か，誤って予測した (`a:f`) か，あるいは予測不可能だった (`a:n`) かの 3 つの場合との組合せで，合計 9 つの部分に分けられる．投機的判定を行なう場合には，グラフの下から 5 つの部分までの和が予測率を表すことになる．投機的判定を行なわない場合と比較すると，予測率は3.2% 改善され71.3% に達することがわかる．このことが，0.32% の性能向上に貢献している．

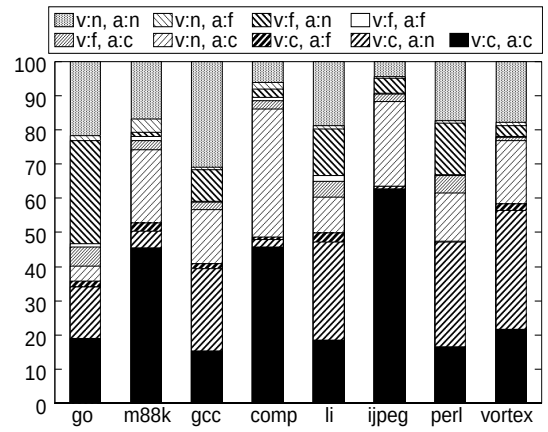

図 18 (%) 投機的判定を行なう場合の予測率

つづいて，2 つのプログラムで性能が低下する原因を考察する．まず第一に，3.4節の検討から，正しいデータ値予測が誤ったアドレス予測によって破棄されたことによるペナルティが原因であると予想できる．しかし，図 18 からはこの結論は導き出せない．なぜなら，どのプログラムにおいても `v:f,a:c` に対する `v:a,a:f` の比率は大差ないからである．そこで分岐予測精度を調査した．**表 2** に投機的判定を行なう場合と行なわない場合の分岐予測失敗率をまとめる．容易にわかるように，`129.compress` と `130.li` において予測失敗率が著しく増大している．これが性能低下の原因である．データ投機実行による命令流の変化が，分岐予測精度の低下をひき起こしたと考えられる．データ投機実行と分岐予測精度との関係の考察は将来の課題である．

表 2 (%) 分岐予測失敗率

プログラム	投機的判定	
	あり	なし
099.go	19.49	19.50
124.m88ksim	3.46	3.51
126.gcc	12.61	12.59
129.compress	3.38	2.99
130.li	4.07	3.82
132.ijpeg	1.20	1.20
134.perl	3.00	3.03
147.vortex	5.31	5.32

以上シミュレーション結果からは，投機的判定を行なっても性能向上率はわずかしか改善されず，投機的判定を実装するためのハードウエアを正当化できるほどの効果がないことがわかる．

5.4 共用 LIST/RPT

図 19 は，図 10 のようにハードウエア規模の削減のために LIST と RPT を共用させた時の，プロセッサ性能向上率を表している．各プログラムの 5 つのグラフは左から順にそれぞれ，データ値予測器 (`Value`)，投機的判定を行なわない協調型予測器 (`Coop`)，投機的判定を行なわない LIST と RPT を共用した協調型予測器 (`Coop:LIST`)，投機的判定を行なう協調型予測器 (`Spec`)，そして投機的判定を行なう LIST と RPT を共用した協調型予測器 (`Spec:LIST`) を用いた時のプロセッサ性能の向上率を表している．図よりわかるように，LIST と RPT とを共用すると，全てのプログラムで共用しない協調型予測器を用いた場合よりも性能向上率が低下している．これは，3.3節で予想したように，データ値予測器とアドレス予測器とが予測容易なロード命令が異なりその重なりが小さいため，登録された情報がテーブルから頻繁に追い出されるためである．さらに，データ値予測器と比較しても向上率が低下している場合すら存在する．したがって，LIST と RPT とを共用することは望ましくないことがわかった．

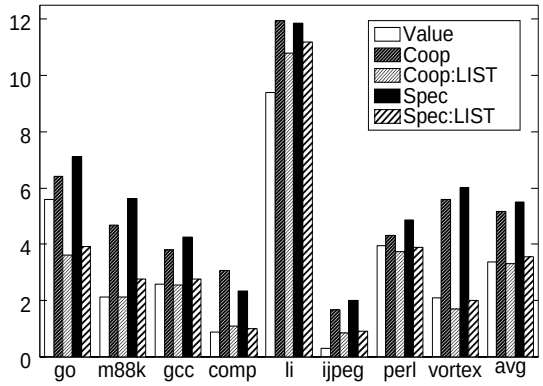

図 19 (%)LIST と RPT を共用した時の性能向上率

6. ま と め

本稿では，ロードデータ値予測器とロードアドレス予測器とを組み合わせることで予測率を改善し，その結果命令レベル並列度を向上させることを検討した．予測精度よりも予測率がプロセッサ性能の向上率を決定づけているので，予

測精度を維持したまま予測率を改善することが望ましい．その目的で我々は協調型予測器を提案した．ロードデータ値を予測不可能な時にはデータアドレスを予測することでデータ依存を投機的に解消し，命令レベル並列度の向上を目指す．さらに，データ値予測失敗のペナルティを軽減する目的で投機的判定を提案した．予測アドレスを用いて獲得されたデータと予測データとを比較することで，データ値予測失敗を早期に検出しミスペナルティを軽減する．

シミュレーションによる評価の結果，協調型予測器は予測率を改善しプロセッサの性能向上率を向上できることが確認できた．予測率は平均68.1% まで改善され，データ投機実行による性能向上への寄与は，平均5.2%，最大11.9% に向上した．これは，データ投機実行の寄与としては，データ値予測器もしくはアドレス予測器単体の54% および105% 増にあたる．さらに投機的判定を行なうと，平均の予測率はさらに3.2% 改善し71.3% になることがわかった．残念ながらこの改善は，プロセッサ性能にはわずか0.32% の向上しか寄与しなかった．これらのシミュレーション結果から，協調型予測器はデータ値予測器やアドレス予測器よりもより一層データ依存関係を投機的に解消可能で，より大きな命令レベル並列度を引き出すことができることが確認できた．しかし，投機的判定は実装のためのハードウエアを考慮するとコストに見合わないと考えられる．

将来の課題としてまず混成型予測器との比較が挙げられる．動的に予測器を選択する混成型予測器は予測器選択機構のためのハードウエアが必要となるが，あらかじめコンパイル時に選択を決定しておけばこのハードウエアは不要になる[13]．すなわち協調型予測器とハードウエア規模が同じであり，両者を比較することは興味深い．またハードウエア量とのトレードオフの検討も必要であろう．キャッシュや分岐予測機構，あるいは命令ウインドウの容量を増加させることによる効果と，アドレス予測機構あるいはデータ値予測機構を実装することによる効果とを比較することは，限られたハードウエア量の有効利用という観点から有意義であると思われる．ただし，容量の増加は速度とのトレードオフであることにも注意を要するだろう．

参 考 文 献

1) Burger,D., Austin,T.M.: The SimpleScalar tool set, version 2.0, *ACM SIGARCH Computer Architecture News*, vol.25, no.3 (1997).
2) Chen,T-F., Baer,J-L.: Effective hardware-based data prefetching for high-performance processors, *IEEE Trans. Comput.*, vol.44, no.5 (1995).
3) Cortadella,J., Llaberia,J.M.: "Evaluation of A+B=K conditions without carry propagation, *IEEE Trans. Comput.*, vol.41, no.11 (1992).
4) Gabbay,F., Mendelson,A.: Can program profiling support value prediction?, *Proc. of 30th Int'l Symp. on Microarchitecture* (1997).
5) Gonzalez,J., Gonzalez,A.: Speculative execution via address prediction and data prefetching, *Proc. of 11th Int'l Conf. on Supercomputing* (1997).
6) Jourdan,S., Sainrat,P., Litaize,D.: Exploring configuration of functional units in an out-of-order superscalar processor, *Proc. of 22nd Int'l Symp. on Computer Architecture* (1995).
7) Lipasti,M.H.: Value locality and speculative execution, *Ph.D. Dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University* (1997).
8) Morancho,E., Llaberia,J.M., Olive,A.: Split last-address predictor, *Proc. of Int'l Conf. on Parallel Architectures and Compilation Techniques* (1998).
9) Moshovos,A.I., Sohi,G.S.: Streamlining inter-operation memory communication via data dependence prediction, *Proc. of 30th Int'l Symp. on Microarchitecture* (1997).
10) Reinman,G., Calder,B.: Predictive techniques for aggressive load speculation, *Proc. of 31st Int'l Symp. on Microarchitecture* (1998)
11) Rychlik,B., Faistl,J.W., Krug,B.P., Kurland,A.Y., Sung,J.J., Velev,M.N., Shen, J.P.: Efficient and accurate value prediction using dynamic classification, *Technical Report CMuART-98-01, Department of Electrical Computer Engineering, Carnegie Mellon University* (1998).
12) Sato,T.: Analyzing overhead of reissued instructions on data speculative processors, *Digest of Workshop on Performance Analysis and its Impact on Design held in conjunction with 25th Int'l Symp. on Computer Architecture* (1998).
13) Sato,T.: Profile-based selection of load value and address predictors, *Proc. of Int'l Symp. on High Performance Computing* (1999).
14) 佐藤寿倫: 命令再発行機構によるデータアドレス予測に基づく投機実行の効果改善, 情報処理学会論文誌, 第40巻, 第5号 (1999).
15) 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, 第40巻, 第5号 (1999).
16) Sazeides,Y., Vassiliadis,S., Smith,J.E.: The performance potential of data dependence speculation & collapsing: *Proc. of 29th Int'l Symp. on Microarchitecture* (1996).
17) Sazeides,Y., Smith,J.E.: The predictability of data value, *Proc. of 30th Int'l Symp. on Microarchitecture* (1997).
18) Sohi,G.S.: Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers, *IEEE Trans. Comput.*, vol.39, no.3 (1990).
19) Tyson,G., Austin,T.M.: Improving the accuracy and performance of memory communication through renaming, *Proc. of 30th Int'l Symp. on Microarchitecture* (1997).
20) Wang,H., Sun,T., Yang,Q.: CAT – caching address tags: a technique for reducing area cost of on-chip caches, *Proc. of 22nd Int'l Symp. on Computer Architecture* (1995).
21) Wang,K., Franklin,M.: Highly accurate data value prediction using hybrid predictors, *Proc. of 30th Int'l Symp. on Microarchitecture* (1997).
22) Widigen,L., Sowadsky,E., McGrath,K.: Eliminating operand read latency, *ACM SIGARCH Computer Architecture News*, vol.24, no.5 (1996).