クロスバスイッチをなくしたマルチバンクキャッシュ

嶋田　創[†]　安藤秀樹[†]　島田俊夫[†]

プロセッサの性能を向上させるためには，命令の実行に必要なデータが遅れることなく供給される必要がある．そのために，現在の多くプロセッサにおいて，マルチバンク化という手法でキャッシュをマルチポート化している．しかし，クロスバスイッチの存在により，従来のキャッシュよりもキャッシュアクセスに時間がかかるという欠点がある．我々はこのクロスバスイッチをなくしたマルチバンクキャッシュの構成と，その実現のために必要なバンク番号の早期計算機構を提案する．さらに，提案する機構においてより高い性能を得るために，バンク予測による投機的命令発行の併用を提案する．評価の結果，バンク番号の早期計算とバンク番号予測を用いた投機的命令発行により，最大7.8%，平均5.1%の性能向上を達成できることを確認した．

Multi-Bank Cache without Latency of Crossbar Switch

HAJIME SHIMADA,[†] HIDEKI ANDO[†]
and TOSHIO SHIMADA[†]

Data must be provided without delay when instruction requests them to sustain processor performance. Caches must have sufficient numbers of ports to meet this requirement in wide-issue processors. A general way to organize multiported caches is to separate caches in banks. Although multi-bank caches are more efficient than any other alternatives, cache access latency lengthens due to the crossbar switch that connects load/store units and cache banks. This paper proposes a mechanism that removes the crossbar to reduce the latency. The mechanism includes early calculation of bank numbers, bank number prediction, and speculative instruction execution based on that prediction. Our evaluation results show that our multi-bank cache with the mechanisms achieves 5.1% average performance improvements or 7.8% maximum performance improvements over a conventional multi-bank cache in an eight-issue processor.

1. はじめに

近年のプロセッサは，命令レベルの並列性を用いて1サイクル中に複数の命令を実行するスーパスカラプロセッサが主流となっている．そして，より多くの命令レベル並列性を抽出するために，プロセッサの命令発行幅は増加する傾向にある．プロセッサで実行されるプログラムの命令の平均3分の1はロード／ストア命令である[1]．そのため，多命令発行を行うプロセッサでは，1サイクル中に複数のロード／ストア命令が発行される頻度が高く，この要求に応えるデータキャッシュが必要となる．このため，近年のプロセッサでは，データキャッシュはマルチポート化されている．

実際のプロセッサで利用されているキャッシュのマルチポート化の手法は2つある．まず1つ目は多重化である．全く同じ内容のキャッシュを必要なポート数だけ用意することにより，複数のロード／ストア命令の要求に応えるものである．この構成は，任意の組合せのロード／ストア命令の要求に応えることができるが，Nポートキャッシュを実現すると，コストはシングルポートキャッシュのN倍になるという問題がある．この手法は，DEC/Compaq Alpha 21164[2]で採用されている．

2つ目はマルチバンク化である．この手法は，キャッシュをバンクと呼ばれるグループに分け，バンクごとにポートを用意する．これにより，異なるバンクにアクセスされた時のみ，複数のロード／ストア命令の要求に応えることができる．追加コストは，バンクごとのデータ選択機構と，バンクとロード／ストアユニットをつなぐクロスバスイッチだけであり，キャッシュの容量にかかわらず，追加コストは小さい．このため，マルチバンク化は多重化と比較して，大容量キャッシュのマルチポート化を実現をしやすい．このような面での有利さから，Intel Pentium[3], MIPS R10000[4], Sun UltraSPARC[5], AMD Athlon[6]など，近年の多くのプロセッサにおいて採用されている．しかし，同一バンクへのアクセス競合や，クロスバスイッチによるアクセス時間の増加などの問題がある．

本研究の目的は，このマルチバンクキャッシュにお

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

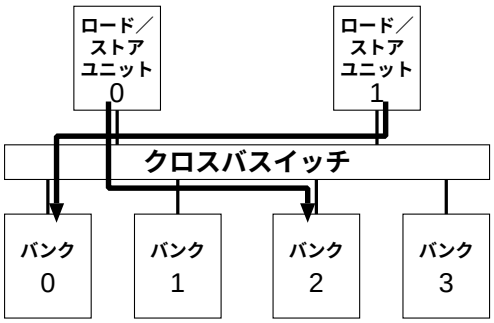

図 1　マルチバンクキャッシュの構成

いて，クロスバスイッチによるアクセス時間の増加を削減することある．本論文では，クロスバスイッチをなくしたマルチバンクキャッシュの構成と，その実現に必要なバンク番号の早期計算機構を提案する．さらに，その構成においてより高い性能向上を得るために，バンク番号予測による投機的命令発行の併用を提案する．

以下に本論文の構成を示す．2 章では本研究の基礎となるマルチバンクキャッシュについて詳しく述べ，3 章で提案するクロスバスイッチをなくしたマルチバンクキャッシュの構成を示す．また，提案する機構の実現に必要なバンク番号の早期計算と，さらなる性能向上を目指すためのバンク番号予測による投機的命令発行について説明する．4 章で本研究で提案する手法の評価方法と評価に用いるベンチマークを説明し，評価によって得られた結果を示し，評価結果の考察を行う．5 章で関連研究について述べ，最後に 6 章で本研究によって得られたことをまとめる．

2. マルチバンクキャッシュ

2.1 マルチバンクキャッシュの構成

マルチバンクキャッシュの構成を**図 1** に示す．キャッシュは，キャッシュラインやワードなどを単位として，バンクと呼ばれるグループに分けられており，各バンクごとにポートを持っている．各バンクのポートはクロスバスイッチを介してロード／ストアユニットと接続されている．このように，クロスバスイッチを用いてロード／ストア命令の要求ごとにロード／ストアユニットとキャッシュのバンクをつなぐことにより，1 サイクル中に複数のロード／ストア命令の実行を可能としている．

キャッシュアクセスに必要なライン番号とライン内オフセットはキャッシュアクセスに必要なライン番号とライン内オフセットは，アドレスの一部を用いて指定する．マルチバンクキャッシュで新たに必要となるバンク番号は，**図 2** のように，従来のキャッシュのライン番号の下位ビットをバンク番号として用い，複数のロード／ストア命令の要求が同一バンクへ集中することを防ぐ．なお，図 2 におけるバンク番号等の取り方は，4 章で行なう評価の際に仮定する 4 バンク，4 ポート，1 ライン 32 バイト，各バンク 512 ラインのマルチバンクキャッシュにおける取り方である．

TAG:**キャッシュタグ** LN:**ライン番号**
BN:**バンク番号** LO:**ライン内オフセット**

図 2　バンク番号の確保

2.2 マルチバンクキャッシュの欠点

マルチバンクキャッシュは，現在の数多くのプロセッサで採用されているが，利点と同時にいくつかの欠点もある．以下，欠点について述べる．

(1)　バンク競合

マルチバンクキャッシュでは 1 つのバンクに対してポートは 1 つしかない．したがって，ロード／ストア命令の要求が 1 つのバンクに集中した場合は，複数の要求のうち，1 つの要求にしか応えることができない．これを**バンク競合**と呼ぶ．

(2)　クロスバスイッチによるレイテンシの増加

マルチバンクキャッシュのアクセス時には，シングルポートキャッシュと比較して，クロスバスイッチを通過する時間が余分にかかる．これによるクロックサイクル時間の増加を避けるため，通常，データキャッシュのアクセスは 2 段以上のパイプラインで構成されている．

我々は，後者の問題に注目し，クロスバスイッチによるレイテンシを削減する研究を行なった．

3. クロスバスイッチをなくしたマルチバンクキャッシュの提案

本章では，まず最初に，クロスバスイッチをなくしたマルチバンクキャッシュの構成を提案する．次に，その構成の実現に必要となるバンク番号の早期計算機構，および，さらなる性能向上を得るためのバンク番号の予測による投機的命令発行について述べる．この機構により，従来のクロスバスイッチのあるマルチバンクキャッシュと比較して，ロード／ストア命令のレイテンシを削減できる．

3.1 クロスバスイッチをなくしたマルチバンクキャッシュの構成

従来のキャッシュでは，ロード／ストア命令の要求を各バンクに振り分けるため，ロード／ストアユニットとキャッシュの間にクロスバスイッチをいれる構成を取っている．我々は，クロスバスイッチにおけるロード／ストア命令の要求を振り分ける作業が，命令ウィンドウから各機能ユニットに命令を発行する作業と似

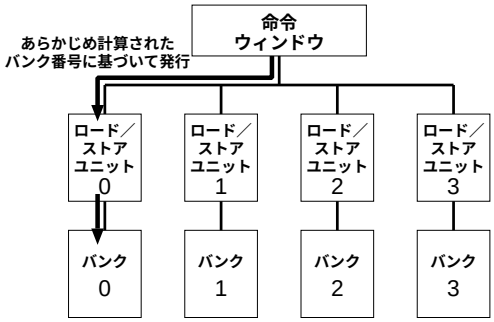

図 3　クロスバスイッチをなくしたマルチバンクキャッシュの構成

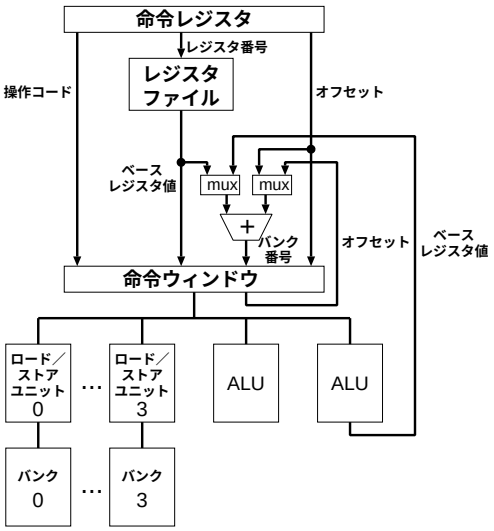

図 4　バンク番号の早期計算

ている点に着目した．そして，命令ウィンドウに命令が入る前にバンク番号を求め，命令ウィンドウから各機能ユニットに命令を発行する作業に，各バンクへの振り分け作業を兼ねさせることを考えた．

図 3 に，提案するクロスバスイッチをなくしたマルチバンクキャッシュの構成を示す．この構成では，バンクと同数のロード／ストアユニットを用意し，ロード／ストアユニットとバンクを一対一で接続する．ロード／ストア命令は，あらかじめ計算されて命令ウィンドウに入っているバンク番号を元に，各バンクに接続されているロード／ストアユニットに発行される．

バンク番号の計算は，レジスタ読み出しステージにおいてレジスタを読み出した直後に行う．バンク番号の計算を行う機構を**図 4** に示す．バンク番号は，レジスタファイルより読み出されたレジスタ値と，命令レジスタから読み出されたベースレジスタに対するオフセット値を加算して求められ，命令ウィンドウに書き込まれる．図 2 より，バンク番号はアドレスの下位にあるため，レジスタ値，およびオフセット値を全て加算しなくても求めることが可能である．この例では，下位の 7 ビットのみ計算すれば良い．

この構成によって，従来の構成に存在していたクロスバスイッチは不要となる．レジスタ読み出しステージでバンク番号が計算できるなら，クロスバスイッチがなくなったことにより，ロード／ストア命令の実行時のパイプラインは**図 5**(b) のようになり，図 5(a) に示す従来のクロスバスイッチのあるマルチバンクキャッシュと比較して，ロード／ストア命令の実行レイテンシは 1 サイクル短くなる．

問題となる点として，レジスタ読み出しステージにおいてバンク番号の計算を行えない場合がある．それは，ロード／ストア命令のベースレジスタがまだ計算されておらず，利用可能でない場合である．この時は，ベースレジスタが計算された後は，その値と命令ウィンドウに格納したオフセット値からバンク番号を計算した後，ロード／ストア命令を発行する．この時，命令の発行スロットを消費する．この計算の様子を**図 6** に示す．まず，図 6(a) に示すように，ベースレジスタが計算され，ライトバックされるサイクルにおいて，そのレジスタ値をバンク番号計算器にフォワーディングする．これを，命令ウィンドウより送られて来るオフセット値と加算し，バンク番号を計算して命令ウィンドウに書き込む．そして，図 6(b) に示すように，後のサイクルで，計算で得られたバンク番号に基づいて命令を発行する．

この時のロード／ストア命令の実行時のパイプラインは図 5(c) のようになる．バンク番号計算のステージが追加されていることに注意されたい．**図 7** に，ベースレジスタがフォワーディングされ，それによりロード／ストア命令が発行されるタイミングを示す．図 7 に示すように，従来の構成では，ベースレジスタを計算する命令の実行が行なわれるサイクルの次のサイクルにはロード／ストア命令のアドレス計算が行なえたが，提案する機構では，アドレス計算の前にバンク番号計算と発行のサイクルが必要になる．そのため，従来の場合と比較して実行レイテンシは 1 サイクル増え，ペナルティとなる．

3.2　バンク番号を計算するステージについて

前節では，レジスタ読み出しとバンク番号計算を同一のステージで行うと仮定した．これがクロックサイクル時間にどの程度悪影響を与えるかを見積もることは，実際に設計を行ってみる必要があり，難しい．参考になる研究として，スーパスカラプロセッサを構成する部品やクリティカルパスの遅延時間を見積もった文献 7) がある．この研究によれば，4 命令発行のスーパスカラプロセッサにおいて，0.5μmCMOS，3 層メタル配線の半導体プロセスでは，レジスタ読み出しステージは，結果の書き込みに 3.4ns，読み出しに 3.4ns の計 6.8ns かかる．これに対して，クリティカルパスは ALU での実行とバイパスの 8.4ns であり，1.6ns の

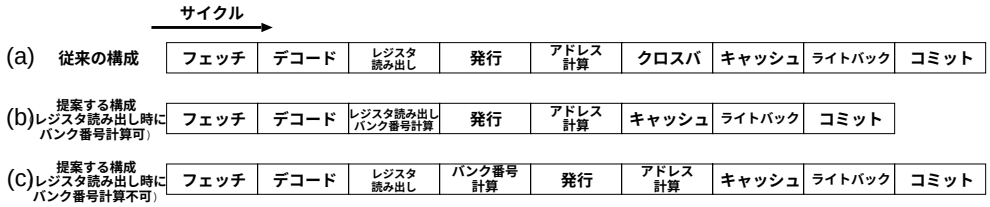

図 5　実行パイプラインの比較

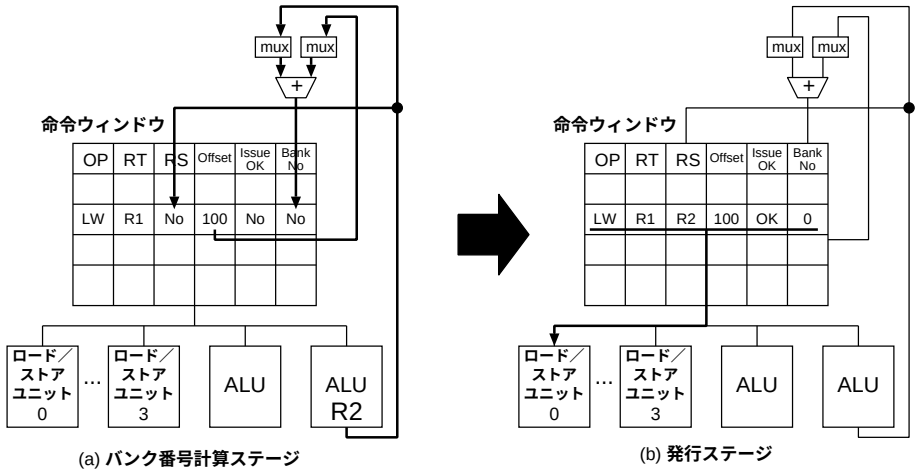

図 6　フォワードされたレジスタ値によるバンク番号計算

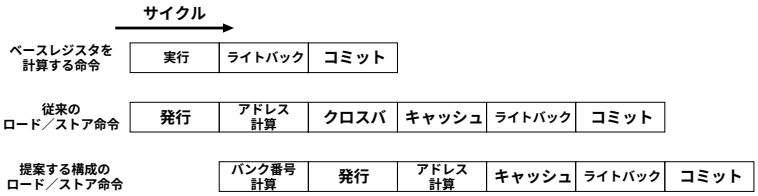

図 7　ベースレジスタがフォワーディングされロード／ストア命令が発行されるタイミング

差 (クロックサイクル時間の 19%) がある．バンク番号の計算はアドレスのバンク番号部とその下位のライン内オフセット部の計算である．図 2 に示す構成では，クロックサイクル時間の 19%の間に 7 ビットの加算を行う必要がある．

以上の事実から，クロックサイクル時間に影響を与えない設計が可能かそうでないかは，半導体プロセスや回路設計技術に依存するので，判断することは難しい．不可能な場合に対応し，次の 2 つの対処方法を考えた．

(1)　**バンク番号の計算を行うステージを独立させる**

これはバンク番号を計算するステージを，レジスタ値を読み出すステージから独立させるという対処方法である．クロックサイクル時間に与える影響はなくなるが，実行までのパイプラインが 1 ステージ長くなるので分岐予測ミスペナルティが 1 サイクル増加することになる．これにより，ロード／ストア命令の実行レイテンシ短縮による性能向上が抑えられる．

(2)　**バンク番号の計算に使うビット数を制限する**

この手法では，レジスタ読み出しステージにおいて，バンク番号の計算に使うビット数を制限する．具体的には，図 2 の例の下位より 5,6 ビット目であるバンク番号指定部 (BN) と，それへの桁上げ予測値の加算により求める．桁上げの予測は，ベースレジスタとオフセット値におけるライン内オフセット部 (LO) の最上位ビットの論理積をとったものとする．LO の最上位の桁のレジスタ値とオフセット値が (1,1) の時に桁上りがあり，(0,0) の時には桁上りがないことを利用するものである．(1,0) あるいは (0,1) の時は LO の最上位より下位のビットに依存するが，これらのビットがランダムでも 50%の確率で予測は成功する．従って，この予測手法は，最終的には 75%程度の確率で成功すると考えられる．また，オフセット値は 0 である確率は高く[1)]，このときは桁上がりはないので，予測成功の確率はさらに

高いと期待できる．これにより，計算を行うビット数が 7 ビットから 3 ビットに削減できる．しかし，キャリーがライン内オフセットの下位から伝播して来る場合は，計算結果が正しいバンク番号にならないため，これを用いた場合の命令発行は次節で述べる投機的命令発行と同様に，発行した命令が正しいバンクに接続されているロード／ストアユニットに発行されたどうかの確認と，発行ミス時の処理が必要になる．

3.3 バンク番号予測による投機的命令発行

3.1節では，レジスタ読み出しステージにおいてバンク番号を計算できない場合は，バンク番号の計算を行なうためのサイクルを設けることにした．本節では，このバンク番号の計算を行うサイクルを省くために，バンク番号を予測し，それをもとに投機的にロード／ストア命令を発行することを考える．以下，3 つのバンク予測の手法を提案する．

(1) **Last Bank**

メモリアクセスには空間的局所性があり，最近アクセスしたデータの周辺は再びアクセスされる可能性が高いことはよく知られている．Rivers らによれば，4 バンクのマルチバンクキャッシュにおいて，連続するロード／ストア命令が同一バンクをアクセスする確率は 30%〜55%である[8]．したがって，前のサイクルで最後のロード／ストア命令がアクセスしたバンクを予測バンク番号とすれば，高い確率で予測が成功すると考えられる．これを実現するため，最後にアクセスしたバンクの番号を保持するためのレジスタを準備しておく．このレジスタの値は，ロード／ストア命令の実行ごとに更新される．そして，命令発行時にレジスタを参照し，前のサイクルで最後のロード／ストア命令がアクセスしたバンクを予測バンク番号とする．ただし，Last Bank で予測されるバンクは 1 つであるから，同一サイクルに 2 つ以上の投機はバンク競合によりできない．

(2) **Old Register**

この手法では，レジスタ読み出しステージにてレジスタファイルに入っている古いレジスタ値を用い，通常のバンク番号と同様の計算を行う．この計算により得られる値をアドレスと推定し，バンク番号になる桁の値を，予測バンク番号とする．これは，計算後のレジスタ値は計算前のレジスタ値に対して，全ての桁が異なることは少ないという仮定に基づいている．計算後のレジスタ値において，バンク番号の計算に影響する桁が計算前のレジスタ値と変化しない場合，この手法によって得られる予測バンク番号は的中することになる．

この予測が正解となる例として，以下の命令列をあげることができる．

```
LOOP:
      i1: addi R2,R2,1
      i2: lb R3,0(R2)
      i3: add R4,R4,R3
      i4: subi R5,R5,1
      i5: bne R5,R0,LOOP
```

命令 i2 は，ストライドの大きさが 1 バイトである連続した要素へアクセスする．この場合のロード命令はオフセット値を固定し，ベースレジスタ値 (R2) をインクリメントしながらアクセスする．この時，図 2 のようにライン内オフセットが 5 ビット取られているキャッシュでは，インクリメントによってバンク番号の値が変化するのは，32 回のインクリメントに 1 回となる．このような命令列に Old Register を適用すると，高い確率で予測が成功する．

(3) **Combination**

この手法は，Last Bank による予測の欠点である複数投機を行えないという点を改良するため，Old Register と組み合わせたものである．基本的に Last Bank を使い，Last Bank では投機を行えない同一サイクル中の 2 回目以降のバンク番号予測に，Old Register を用いるという手法である．

投機的命令発行を行うに当たり，発行ミス時からの回復手法も考えなくてはならない．発行ミスは，投機的に発行された命令が，ロード／ストアユニットにおいてアドレス計算を行った時に確定する．もし，正しいアドレスのバンク番号と，予測されたバンク番号が異なる場合，発行ミスである．発行ミスした命令は，次のサイクルにおいて，命令ウィンドウから正しいバンク番号をもとに，正しいロード／ストアユニットに再発行される．この時に必要となる正しいバンク番号は，ロード／ストアユニットで計算されたアドレスより得られる．

4. 評価結果および考察

4.1 評 価 方 法

SimpleScalar Tool Set[9]中の out-of-order 実行シミュレータを基に，提案の機構を組み込んだシミュレータを用いて測定を行った．命令セットは MIPS R10000[4]である．ベンチマークプログラムは**表 1** に示すように，SPECint95 の 8 本を用いた．バイナリは SimpleScalar Tool Set に含まれているものを用いた．シミュレーション時間が過大にならないようにするために，関数の出現頻度をほぼ維持しつつ，実行命令数がおおよそ 100M〜200M になるようにそれぞれのベンチマークプログラムへの入力を調整した．

シミュレーションにおいて仮定したプロセッサの構

表 1　ベンチマークプログラム

ベンチマーク	入力	実行命令数	メモリ参照数	
			ロード	ストア
compress95	test.in	95M	20M	13M
gcc	jump.i	156M	41M	22M
go	2stone9.in	239M	50M	18M
ijpeg	specmun.ppm	207M	39M	18M
m88ksim	ctl.in	245M	36M	20M
li	train.lsp	183M	47M	30M
perl	scrabbl.in	155M	44M	28M
vortex	vortex.in	197M	59M	45M

表 2　プロセッサの構成

命令発行幅		8 命令
命令ウィンドウ		32 エントリ
機能ユニット	整数	4 個
	整数乗除算	1 個
	浮動小数点	2 個
	浮動小数点乗除算	1 個
TLB	命令	64 エントリ
	データ	128 エントリ
分岐予測	予測手法	gshare
	履歴長	8 ビット
	インデックス長	14 ビット
	BTB	2048 エントリ
	RAS	16 エントリ
	ミスペナルティ	下記以外：4 ステージ追加：5

表 3　キャッシュの構成

	L1 命令	L1 データ	L2
容量	64KB	64KB	2MB
ラインサイズ	32B	32B	64B
連想度	1	1	4
バンク数	1	4	4
ポート数	1	4	4
ヒットレイテンシ	1	提案：1，従来：2	6
ミスペナルティ	6	6	18

成を**表 2** に，キャッシュの構成を**表 3** に示す．表 2 にロード／ストアユニット数がないが，これは，SimpleScalar では整数ユニットがロード／ストアユニットを兼ねているためである．また，表 3 の L1 データキャッシュのヒットレイテンシが 2 つある理由は，それぞれ従来のクロスバスイッチを含むマルチバンクキャッシュにおけるレイテンシと，我々の提案するクロスバスイッチをなくしたマルチバンクキャッシュにおけるレイテンシである．同様に，表 2 の分岐ミスペナルティが 2 つある理由は，我々の提案するマルチバンクキャッシュにおいて，分岐予測ミスペナルティを増加させた場合と増加させない場合 (3.2節参照) を表している．そして，表 3 の L2 キャッシュは命令・データ混合型である．

4.2　提案する機構によって得られた結果

従来のマルチバンクキャッシュの場合を基準とし，

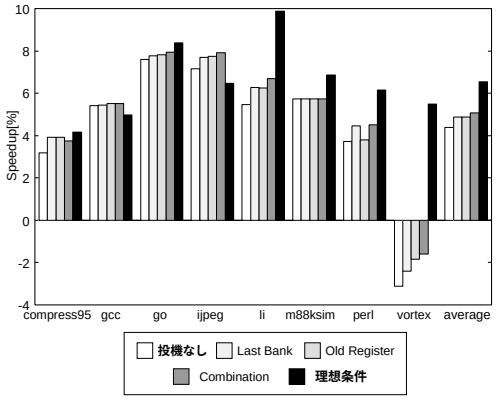

図 8　提案する機構によって得られた結果

提案する機構による性能向上を**図 8** に示す．各ベンチマークの 5 本の棒グラフは左からそれぞれ，投機を用いない場合，Last Bank 予測，Old Register 予測，Combination 予測，全てのロード／ストア命令がレジスタ読み出しステージでバンク番号の計算を行えると仮定した理想条件での性能向上である．

結果をまとめると，提案する機構により，投機を用いずに最大 7.6%，平均 4.4%の性能向上が得られた．さらに，Combination による予測を用いた投機的命令発行を行うことにより最大 7.9%，平均 5.1%の性能向上を達成し，投機を用いない場合よりも最大 0.3%，平均 0.7%多くの性能向上を得られた．

gcc, ijpeg では，理想状態より性能が上がっているが，これは，gshare 分岐予測器の履歴と PHT の更新をコミットステージで行なうため，ロード命令のレイテンシが短くなることによってロードに依存した分岐命令の完了が早まり，分岐予測器が先行する分岐命令の結果を予測に反映させやすくなったためだと考えられる．そのため，どのベンチマークも分岐予測率が向上した．特に，gcc では 0.2%，ijpeg では 0.1%と他のベンチマークと比較し大きく向上したため，この 2 つでは理想状態より高い性能となった．

予測手法の優劣については，Last Bank と Old Register を比較すると，それぞれベンチマーク毎に優劣があり，平均するとほぼ同等の性能向上となった．また，2 つを組み合わせた Combination による性能向

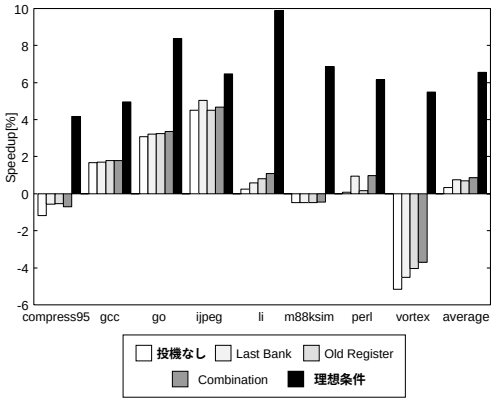

図 9　バンク番号計算ステージを追加した結果

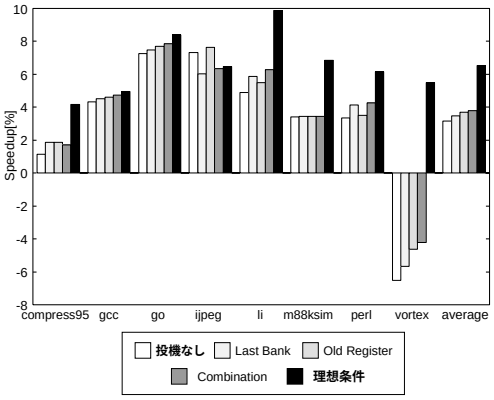

図 10　ビット数限定加算器を用いた結果

上は，compress95, m88ksim を除いて Last Bank と Old Register それぞれによって得られた性能向上の和に近い値になっており，Last Bank と Old Register で予測できる命令は重複が少なく，互いに予測できないものを補完していると言える．唯一，性能向上が得られなかった vortex を見ると，デコードステージにおいてバンク番号の計算ができないロード／ストア命令や，予測を失敗するロード／ストア命令の割合は他のベンチマークとあまり差がない．このため，vortex はペナルティを受けるロード／ストア命令に依存している命令が多いため，性能低下が起きていると推測される．

4.3　バンク番号計算ステージを追加した結果

従来のマルチバンクキャッシュの場合を基準とし，バンク番号の計算ステージを追加した場合の性能向上を**図 9** に示す．分岐予測ミスペナルティの増加により，どの手法においても性能向上が大きく落ち，グラフは図 8 の形をそのままに，性能向上を下げた形になっている．

投機を用いない場合の性能向上は最大は 4.5%あったが，compress95, m88ksim, vortex が性能低下を起こすため，平均はわずか 0.3%であった．さらに，Combination による予測を用いて投機的命令発行を行なっても，平均で 0.9%しか性能は向上しない．

4.4　ビット数限定加算器を用いた結果

ビット数限定加算器を用いた場合の性能向上を**図 10** に示す．この構成において投機を併用する場合，LO の最上位桁のレジスタ値 (r4) とオフセット値 (o4) が (0,1) あるいは (1,0) である場合，加算器による計算結果を用いずに投機を行うことにする．この時の r4,o4 の値に対する投機と計算結果の選択方法を**表 4** に示す．同表よりわかるように，r4,o4 の XOR を取った値が 0 であれば計算結果を使用し，1 であれば投機を行う．なお，投機なしにおいて，レジスタ値 (r4) とオフセット値 (o4) が (0,1) あるいは (1,0) である場合は計算結果を用いる．

表 4　投機と計算結果の選択

r4	o4	バンク番号の決定方法
0	0	計算結果
1	1	
0	1	投機
1	0	

図 8 と図 10 に示した測定結果を比較すると，バンク番号の計算に使うビット数を制限することによる性能向上の減少は，どの手法においても平均 1.2%〜1.4%程度であった．性能向上は，投機を用いずに最大 7.2%，平均 3.1%，Combination による予測を用いた投機的命令発行を行なうことにより最大 7.8%，平均 3.8%であり，投機を用いない場合よりも平均 0.7%多くの性能向上を得られた．

性能向上の減少が少ない大きな理由は，アドレス計算に使うベースレジスタに対するオフセット値が 0 のロード／ストアが非常に多いことである．**表 5** にオフセット値が 0 のロード／ストア命令の割合を示す．平均で 24.4%ものロード／ストア命令のオフセット値が 0 である．

5. 関 連 研 究

マルチバンクキャッシュについての関連研究を述べる．文献 10) ではラインバッファという小さなキャッシュをロード／ストアユニット内に用意し，キャッシュラインのバッファリングを行うことを提案している．ロードがラインバッファにヒットすれば 1 サイクルでデータを取り出せるため，ロード命令のレイテンシを減らす効果がある．しかし，ロード／ストアユニットが複数ある場合，発行されたユニットにアクセスするラインがバッファリングされているとは限らない．また，ストア時にはメモリの一貫性を保つためにキャッシュに書き込まなくてはならないため，ストア命令には効果がないという欠点がある．

文献 8) では競合を緩和する研究がされている．ここでは，同一キャッシュラインにおける競合に注目し，

表 5 オフセット値が 0 のロード／ストア命令の割合

benchmark	オフセット値 0
compress95	22.9%
gcc	19.3%
go	40.3%
ijpeg	43.3%
li	12.5%
m88ksim	28.5%
perl	11.5%
vortex	17.2%
average	24.4%

1 サイクル中に 1 つのラインから複数のデータを供給できる LBIC(Locality Based Interleaved Cache) を提案している．

文献 11) では我々と同様なクロスバスイッチをなくした構成を提案されている．ロード／ストア命令のバンクへの振り分けは，我々は，バンク番号の早期計算とバンク番号予測及びその組み合せで行うのに対し，文献 11) では予測のみで行っている．また，予測手法も 2 バンクに特化した手法であり，3 バンク以上の場合に対応できていない．さらに，予測にテーブルを使うなど，コストのかかる構成となっている．

6. ま と め

現在のプロセッサではキャッシュの容量や必要とされるポート数は増大する傾向にある．キャッシュの容量やポート数を増加させると，キャッシュのレイテンシが増加することが避けられない．本研究では，マルチバンク化によるポート数の増加のために，従来では必要とされたクロスバスイッチをなくした構成を提案することにより，クロスバスイッチによるレイテンシを削減した．提案する機構を評価したところ，バンク番号の早期計算と Combination バンク番号予測を用いた投機的命令発行により，最大 7.9%，平均 5.1%の性能向上を達成することができた．

提案の機構は，実装によってはプロセッサのサイクル時間を悪化させる可能性がある．これを避ける単純な方法として，バンク番号の計算を行う専用のステージを設ける方法が考えられるが，この方法では最大では 4.5%の性能向上を得られたが，性能が低下するベンチマークもあり，平均ではわずか 0.9%しか性能は向上しなかった．これに対し，本論文ではバンク番号計算のビット数を制限する方法を提案した．この方法では，最大 7.8%，平均 3.8%にまで性能向上を回復できることが分かった．

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)「広域命令レベル並列によるマイクロプロセッサの高性能化に関する研究」(課題番号 1068034) 及び文部省科学研究費補助金基盤研究 (C)「分岐予測と投機的実行に関する研究」(課題番号 11680351) 及び財団法人カシオ科学振興財団研究助成「高性能マイクロプロセッサのアーキテクチャに関する研究」の支援により行った．

参 考 文 献

1) J.L.Hennessy and D.A.Patterson: *Computer Architecture A Quantitative Approach*, Morgan Kaufmannn Publishers, Inc. (1990).
2) Digital Equipment Corporation: *Alpha 21164 Microprocessor Hardware Reference Manual* (1995).
3) Intel Corporation: *Pentium(R) Processor Family Developer's Manual Vol.1* (1995).
4) MIPS Technologies, Inc.: *MIPS R10000 Processor User's Manual, Version2* (1996).
5) P.Song: Hal Packs SPARC64 onto Single Chip, *Micro Processor Report Vol.11 No.16* (1997).
6) Advanced Micro Devices, Inc.: *AMD Athlon Processor Technical Brief* (1999).
7) T.Hara, H.Ando, C.Nakanishi, and M.Nakaya: Performance Comparison of ILP Machines with Cycle Time Evaluation, *In Proceedings of 23rd Annual International Symposium on Computer Architecture*, pp. 213–224 (1996).
8) J.A.Rivers, G.S.Tyson, E.S.Davidson, and T.M.Austin: On High-Bandwidth Data Cache Design for Multi-Issue Processors, *In Proceedings of 30th Annual International Symposium on Microarchitecture*, pp. 46–56 (1997).
9) D.Burger and T.M.Austin: The SimpleScalar Tool Set, Version 2.0, Technical Report CS-TR-97-1342, University of Wisconsin-Madison Computer Sciences Department (1997).
10) K.M.Willson and K.Olukotun: Designing High Bandwidth On-Chip Caches, *In Proceedings of 24th Annual International Symposium on Computer Architecture*, pp. 60–67 (1997).
11) A.Yoaz, M.Erez, R.Ronen, and S.Jourdan: Speculation Techniques for Improving Load Related Instruction Scheduling, *In Proceedings of 26th Annual International Symposium on Computer Architecture*, pp. 42–53 (1999).