社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－ARC－178（3）
2008／5／13

回路面積指向レジスタ・キャッシュの評価

塩谷 亮太† 入江 英嗣†† 五島 正裕† 坂井 修一†

† 東京大学大学院 情報理工学系研究科
†† 科学技術振興機構

あらまし レジスタ・ファイルは，最近のスーパスカラ・プロセッサの構成要素のうち，もっとも高コストなものの1つとなっている．本稿では，我々が提案した，レジスタ・ファイルの回路面積の縮小を目的とする回路面積指向レジスタ・キャッシュの評価と議論を行う．既存のレジスタ・キャッシュは，通常のキャッシュと同様，アクセス・レイテンシの短縮とそれによるパイプライン段数の縮小を第一の目的としている．それに対し，回路面積指向レジスタ・キャッシュの第一の目的は，回路面積の縮小であって，アクセス・レイテンシの短縮を図らない．評価により，回路面積指向レジスタ・キャッシュでは，性能をほとんど落とすことなく，メイン・レジスタ・ファイルのポート数を4まで減らすことができ，その回路規模をおよそ1/9程度にすることができることを確かめた．

キーワード レジスタ・ファイル，レジスタ・キャッシュ，回路面積

Evaluation of Area-Oriented Register Cache

Ryota SHIOYA†, Hidetsugu IRIE††, Masahiro GOSHIMA†, and Shuichi SAKAI†

† Graduate School of Information Science and Technology, The University of Tokyo
†† Japan Science and Technology Agency

Abstract Register file is one of the most costly units in recent superscalar processor. In this paper, we evaluate Area-oriented Register Cache which we previously proposed. Conventional register cache aims at reducing access latency and shortening pipeline. Area-oriented Register Cache, on the other hand, aims at reducing area and does not reduce latency. The evaluation shows that Area-oriented Register Cache can reduce number of ports of main register file to 4 without performance penalty and downsize circuit area to roughly 1/9.

Key words Register File, Register Cache, Circuit Area

1. はじめに

レジスタ・ファイルは，最近のスーパスカラ・プロセッサの構成要素のうち，もっとも高コストなものの1つとなっている．

Out-of-order スーパスカラ・プロセッサでは，レジスタ・ファイルの容量は，命令ウィンドウ・サイズの1.5〜2倍程度必要である．また，SMT (Simultaneous Multi-Threading) などのマルチスレッディングを行うプロセッサでは，同時に実行されるスレッドのコンテキストを保持するため，スレッド数に応じた容量が必要となる．

これらの理由のため，レジスタ・ファイルは巨大化する傾向にある．たとえば，2-way SMT をサポートする Intel Pentium 4 Processor の整数レジスタ・ファイルは，128 エントリにもなる．RAM が消費する電力は面積に比例するため [6]，電力消費も多くなる．

レジスタ・ファイルは，多ポートの RAM で構成される．通常，1命令あたり，2つのリード・ポートと1つのライト・ポートが必要であるから，4つの整数系命令を同時に実行するスーパスカラ・プロセッサの整数レジスタ・ファイルのポート数は，合計 12 にもなる．RAM の回路面積は，ポート数の2乗に比例するため，レジスタ・ファイルは，その容量の割に非常に大きなものとなる．[3], [5].

レジスタ・ファイルのレイテンシは配線遅延に支配されるため，LSI が微細化してもほとんど短縮されない．そのため近年では，レジスタ・ファイルを1サイクル程度でアクセスすることはもはや不可能であり，通常 2〜3 程度のパイプライン・ステージが充てられている．

しかしレジスタ・ファイル・アクセスのパイプライン化は，新たにいくつかの問題を引き起こす．まず，パイプラインが深化することの一般的な悪影響として，IPC の低下が挙げられる．パイプラインが深くなればその分だけ，分岐予測をはじめとする各種予測ミス・ペナルティが増加する．また，命令ウィンドウ・エントリや物理レジスタ自体の解放が遅れるため，資源不足によってフロントエンドがストールする確率も増える [9]．さ

らに，レジスタ・ファイル・アクセスのパイプライン化に固有の問題として，バイパス・ネットワークの複雑化がある [12].

このような問題に対し，我々は文献[8] において，**回路面積指向レジスタ・キャッシュ（AORC：Area-Oriented Register Cache）**を提案した．AORC は，その名の通り，回路面積の縮小を第一の目的とするレジスタ・キャッシュ（RC：Register Cache）である．メイン・レジスタ・ファイル（MRF：Main Register File）に，小容量ゆえに高速な RC を追加するという点では，通常の RC と同様であるが，その目的が全く異なる．既存の RC は，通常のキャッシュと同様，アクセス・レイテンシの短縮とそれによるパイプライン段数の縮小を第一の目的としている．それに対して，AORC の第一の目的は，回路面積の縮小であって，アクセス・レイテンシの短縮を図らない．AORC における RC は，レイテンシを短縮するのではなく，MRF へのアクセスを減らすフィルタとして働く．MRF へは，RC にミスした命令のみがアクセスを行うため，そのポート数を非常に少なくすることができる．

AORC は，物理的構成の点では既存の RC などと変わらないため，それらとの比較が欠かせない．そのため，本稿は，以下のような構成とした．まず，2. 章で，背景となるレジスタ・ファイルのパイプライン化について述べる．続く 3. 章で，AORC について簡単に説明したうえで，RC などとの違いを明確にする．その後，4. 章で AORC の詳細な説明を行い．5. 章で評価を行う．

2. レジスタ・ファイルのパイプライン化

レジスタ・ファイルのパイプライン化によって生じる問題は，以下のように，**パイプラインの深化よって生じる一般的な問題**と，**レジスタ・ファイルのパイプライン化に固有の問題**に分けて考えることができる．

（1） パイプラインの深化による一般的な問題：
　(a) 予測ミス・ペナルティの増大
　(b) 資源不足によるストールの増大
（2） レジスタ・ファイルに固有の問題：
　バイパス・ネットワークの複雑化

前者に関しては，後の 5. 章で示すように，その影響は比較的軽微である．しかし，バイパス・ネットワークの複雑化に関しては，バイパスそのものがクリティカルであるため，その複雑化は受け入れがたい．

2.1 バイパス・ネットワークの複雑化

レジスタ・ファイルのパイプライン化は，レジスタ・ファイルの読み書きにかかる時間を増加させる．このため，値をオペランド・バイパスによって得なければならない期間もまた，増加する．

これを行うために，通常のバイパス・ネットワークを単純に拡張したもののブロック図（1 ビット・スライス分）を**図 1** に示す．以下，この回路を**完全バイパス・ネットワーク**と呼ぶことにする．

各演算器の下流には，1，2，3 サイクル前の実行結果を保持

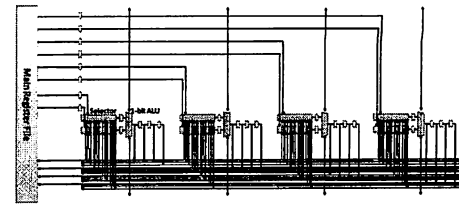

図 1　完全バイパス・ネットワーク（1 ビット・スライス分）

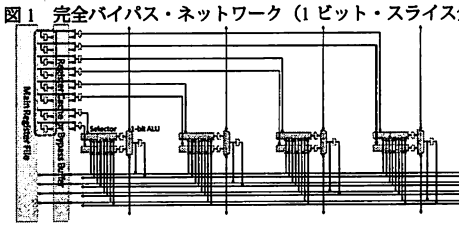

図 2　レジスタ・キャッシュとバイパス・バッファ

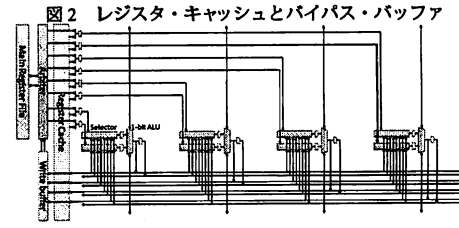

図 3　回路面積指向レジスタ・キャッシュ

するパイプライン・ラッチが，そして，それぞれの内容をソース・ラッチへと転送するためのネットワークが必要となる．このネットワークを構成するためのセレクタは，演算器本体に匹敵する規模の回路となる [12](注1)．

レジスタ・キャッシュ[2] [1] や**バイパス・バッファ**[12] は，こうしたバイパスの複雑化を緩和するものである．次章では，これらの手法について，AORC と比較を行いながら，その違いについて述べる．

3. レジスタ・キャッシュなどとの違い

提案手法と類似の物理構成をもつ手法として，**レジスタ・キャッシュ（RC：Register Cache）** [2], [7], [10], [11] と**バイパス・バッファ**[12] がある．RC とバイパス・バッファは，共にレジスタ・ファイル・アクセスのパイプライン化によって生じる問題の緩和を目的としている．

RC とバイパス・バッファは，共に 1 サイクルでアクセス可能な小型のバッファである．**図 2** に，RC とバイパス・バッファのブロック図を示す．両者は，その周辺までを含めたデータ・パスの構成もほぼ同じである．ただし，同図中破線で囲んだポートは，バイパス・バッファでは必要ない．これはレジスタ・キャッシュ・ミス時などに，MRF の内容を RC にリフィルするためのものである．

回路面積指向レジスタ・キャッシュ（AORC：Area-Oriented Register Cache）もまた，従来の構成と同様の **RC** と **MRF** を持つ．**図 3** に，AORC のブロック図を示す．

以下では，これらの手法について，比較を行いながら説明を

(注1)：なお実際には，必要なオペランドを 1 サイクル以上前に選択しておく ahead pipelining を施すことによって，回路規模を若干縮小することができる．

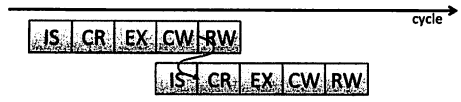

図4　レイテンシ指向レジスタ・キャッシュのパイプライン

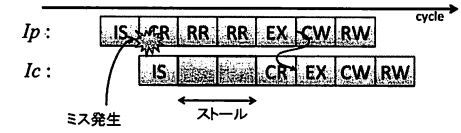

図5　レイテンシ指向レジスタ・キャッシュのミス時の動作

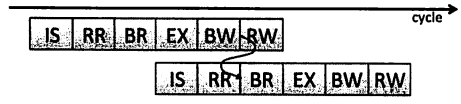

図6　バイパス・バッファのパイプライン

行う．なお，提案するAORCとの区別のため，従来のRCの構成を**レイテンシ指向レジスタ・キャッシュ（LORC：Latency-Oriented Register Cache）**と呼ぶことにする．また，RCと言った場合，MRFの一部を保持する小型のバッファそのものを示すものとする．

3.1　パイプライン構成

それぞれの手法の大きな違いは，そのパイプライン構成にある．LORCは**ヒットを仮定したパイプライン構成**をとり，バイパス・バッファとAORCは**ミスを仮定したパイプライン構成**をとる．以下では，まず，パイプライン構成を中心に概説し，その後3.2章で利害得失についてまとめる．

3.1.1　レイテンシ指向レジスタ・キャッシュ

図4に，RCを持つプロセッサのパイプライン・チャートを示す．図中の**CR**と**CW**は，それぞれ，RCの読み出しと書き出しを意味する．

LORCは，一般的なL1データ・キャッシュと同様に，**ヒットを仮定したパイプライン構成**をとる．すなわち，LORCでは，全ての命令は，そのオペランドがRCにヒットするものとしてスケジューリングされる．

レジスタ・キャッシュ・ミスが発生した場合の考慮についても，通常のデータ・キャッシュの場合と同様である．**図5**に，レジスタ・キャッシュ・ミスが発生した場合の動作を示す．図中の**RR**は，MRFからの読み込みを表す．オペランドがRCにミスした命令I_pは，MRFからの値の読み込みを行う．その結果，I_pと，それに依存するI_cの実行は，ヒットした場合と比べて2サイクル遅くなってしまう．

3.1.2　バイパス・バッファ[12]

バイパス・バッファはRCと同様の，1サイクル(以下）でアクセス可能な小型のバッファである．バイパス・バッファは，完全バイパス・ネットワーク内のラッチが保持していた，バイパスされる値を，FIFOによって保持するものである．バイパス・バッファには，毎サイクル，演算結果の値が書き込まれると同時に，MRFから読み込めるようになった値は順次捨てられる．

図6に，バイパス・バッファのパイプライン・チャートを示す．図中の**BR**と**BW**は，それぞれ，バイパス・バッファの読み出しと書き込みを意味する．バイパス・バッファはLORCと

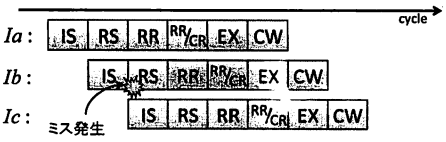

図7　回路面積指向レジスタ・キャッシュのパイプライン

は逆に，**ミスを仮定したパイプライン構成**をとる．このため，LORCとは異なり，バイパス・バッファの場合は必ずMRFにアクセスが行われる．したがって，レイテンシは短縮されないかわりに，ミスも発生しない．

3.1.3　回路面積指向レジスタ・キャッシュ

図7に，AORCの基本的なパイプライン・チャートを示す．この図は，MRFのアクセスに2ステージを割り当てたものである．**RS（Register Scheduling）**は，RCのヒット・ミス判定と，後で述べるMRFアクセスのスケジューリングを意味する．また，**RR**はMRFの読み出しステージを意味する．**RR/CR**は，MRF読み出しと並列して，さらにRCのデータ・アレイ読み出しを行うステージである．

発行されてきた命令は，まずRSでRCに対するオペランドのヒット・ミス判定を行う．従来のLORCと異なるのは，AORCが，バイパス・バッファと同様に**ミスを仮定したパイプライン構成**をとる点である．AORCでは，命令はオペランドのヒット・ミスに関わらず，必ず後続のレジスタ読み出しステージを通過する．このため，レジスタ・キャッシュ・ミスが起きただけでは，ペナルティは発生しない．

RSにおいてヒットと判定されたオペランドは，後続のステージにタグの一致情報だけを送る．RCのデータ・アレイへのアクセスは，実行ステージの直前にあるRR/CRで行う．また，ミスと判定されたオペランドは，後続のRRとRR/CRでMRFへのアクセスを行う．このため，MRFはアクセス数に見合った少数のポートだけを持つ．ミスしたオペランドが多数あり，ポートが足りない場合は，バックエンドのその他のユニットをストールさせ，MRFから値の取得を行う．

5.章の評価で述べるように，MRFのポート数は4程度で十分である．前述したように，RAMの回路面積はポート数の2乗に比例するため，MRFの回路規模は$(4/12)^2 = 1/9$程度になる．このため，MRFは1サイクルでアクセス可能である．

これを利用し，MRFからオペランドを時分割で取得することを行う．たとえば，図7に示すパイプライン構成の場合，命令はMRFに対するアクセスの機会をRRとRR/CRの2回持つ．この，MRFへのアクセス・タイミングのスケジューリングは，RSステージでヒット・ミス判定を行った後に行う．

実行後に行うMRFへの書き込みは，一旦小容量のライト・バッファに書き込んだ後，サイクル・スチールによって，MRFへ順次書き込みを行う．また，このライト・バッファへの書き込みはRCに対して，ライトスルーで行う．

3.2　各方式の得失

上記で述べた各方式は，その目的や問題点などの得失が異なる．以下ではこれらの違いについて議論を行う．

3.2.1 LORC

LORC は，それがヒットする限りにおいては，1 サイクルでアクセス可能なレジスタ・ファイルを持つプロセッサと機能的に等価となる．したがって，理想的には，上記の問題の全てを緩和する効果がある．しかし，実際には，ミス・ペナルティのため，大幅な性能低下を起こすことがある [2].

ミス・ペナルティによる影響を軽減するため，ストールを発生させるのではなく，ミスを起こした命令を選択的に遅らせることが考えられる．しかし，これを行うことは，Out-of-order スーパスカラ・プロセッサの構造上，非常に難しい．

一般に，Out-of-order スーパスカラ・プロセッサでは back-to-back で実行を行うため，依存する命令間の wakeup は毎サイクル連続して行われる．すなわち，ある命令が select された場合，それに依存する命令の wakeup は次のサイクルで行われる．これは，ある命令がレジスタ読み出しのステージに到達した際，それに依存する命令は既に wakeups/select を終えていることを意味する．したがって，レジスタ・キャッシュ・ミスが判明した時点では，自身に依存する命令は既にバックエンドのパイプラインに発行されており，これらを選択的に遅らせるためには，何らかのスケジューリングをもう一度行うことが必要である．これは，通常の命令ウィンドウと同等の機構を，命令発行後のステージにもう一つ設けることを意味するが，そのような複雑化は到底受け入れがたい．

また，この追加スケジューリングを行ったとしても，ヒット・ミスが判明するまでは，それに依存する命令を wakeup することはできない．このため，back-to-back で実行を行うことはできず，そのスループットは大幅に低下してしまう．

これとは別に，通常のロード命令で行われているのと同様にしてオペランドのヒット・ミス予測を行い，ミスを起こすと予測した命令にによる wakeup を遅らせることが考えられる．しかし，この場合，ミスを起こすと予測した命令は，その発行を二回行う必要がある．

MRF から数サイクルかけて値を取得するためには，まず一度発行を行い，MRF に対するアクセスを開始しなければならない．そして，実際に実行を行うため，MRF から値が取れてくるタイミングで，さらにもう一度発行を行う必要がある．なお，一度目に発行された命令を実行されるタイミングまで留めておくことは，演算器などの競合を招くため，資源を増やすことなく行うことは難しい．

後の 5. 章で述べるように，発行幅を二倍消費することによる影響は大きく，ヒット・ミス予測が完全にヒットする（ストールは発生しない）場合であっても大きな性能低下を示す．

3.2.2 バイパス・バッファ

バイパス・バッファは，機能的には完全バイパス・ネットワークと等価であり，MRF から値が取得できない場合にのみバイパス・バッファにアクセスが行われる．このため，キャッシュ・ミスが発生することは無いが，パイプラインの短縮も行われない．したがって，バイパス・バッファは，上記の問題のうち，バイパスの複雑化のみを緩和するものである．

バイパス・バッファは，レジスタ・ファイルのパイプライン化によって生じる問題のうち，(1) 一般的な問題の緩和を放棄することにより，RC のミスを無くしたものであると考えることができる．バイパス・バッファがこのようなアプローチをとるのは，後の 5. 章で示すように，RC のミスによる性能低下が大きいことと，パイプラインの短縮による効果が小さく，性能低下に見合うものではないためである．

表 1　RC と MRF の面積オーダ

Table 1　Area order of RC and MRF

ユニット	エントリ数	ポート数	面積のオーダ
RC	16	12	16*12*12=2304
MRF	128	4	128*4*4=2048

3.2.3 AORC

AORC は，RC を MRF へのアクセスのフィルタとして用いることにより，MRF のポート数を削減することを目的としている．理想的に動作した場合，AORC は，パイプライン化されたレジスタ・ファイルと機能的に等価となる．AORC は，バイパス・バッファと同様に，レイテンシの短縮によるパイプラインの縮小を放棄しており，キャッシュ・ミスが起きただけでは，ペナルティは発生しない．AORC のペナルティは，レジスタ・アクセスのために用意された期間に，全てのオペランドを揃えられなかった場合に発生する．

RC のミス率は，最大で 13%程度 [11] と L1 データ・キャッシュなどと比べると著しく高い．RC をレイテンシ短縮のために用いようとした場合，ミスの発生はペナルティに直結するため，これほど高いミス率では受け入れがたい．しかし，AORC において，アクセスのフィルタとして用いる場合には，このミス率でも十分有効に機能する．5. 章で詳しく述べるが，AORC では，MRF のポートをミス率に応じた数だけ設ければ，ほぼペナルティは発生しなくなる．

4. 回路面積指向レジスタ・キャッシュの詳細

本章では **AORC** について，その詳細や考慮すべき点を述べる．

4.1 回路規模の検討

3. 章で述べたように，AORC のパイプラインでは，RC にミスしたオペランドのみが MRF に対してアクセスを行う．このため，MRF は，アクセス数に見合った少数のポートを持つだけでよい．たとえば，RC のヒット率を 80%とし，毎サイクル 8 個のオペランドを供給する必要があった場合，MRF へアクセスを行うオペランドは，サイクルあたり平均 1.6 個となる．

SRAM の回路面積は，およそポート数の 2 乗に比例する [3], [5]．このため，ポート数の少ない MRF は非常に小さなものとなる．

たとえば，**表 1** に示す構成の場合，RC と MRF の面積は同程度の大きさとなる．したがって，RC が 1 サイクルでアクセス可能であるのならば，MRF も 1 サイクルでアクセス可能であると言える．

4.2 アレイ・アクセスの遅延による効果

AORC のパイプラインでは，ヒット・ミス判定とデータ・アレイ・アクセスを逐次的に別のステージで行う．これにより，

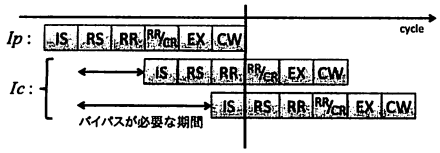

図 8　AORC におけるバイパス期間の短縮

バイパス期間の短縮を行うことができる．

4.2.1　バイパス期間の短縮

データ・アレイ・アクセスを実行ステージの直前まで遅らせることにより，バイパスを行わなければならないサイクル数を短くすることができる．**図 8** に例を示す．今，パイプライン上の命令 I_p に対し，I_c が依存しているとする．I_c が，RC を介して値を得ようとした場合，I_p の RC への書き込みが終了した後に読み出しを開始しなければならない．このため，RS ステージに読み出しを行った場合には，4 サイクル間バイパスを行う必要がある．しかし，RR/CR に読み出しを行った場合には，2 サイクル間バイパスを行うだけで済み，2 サイクルの短縮となる．

4.2.2　リフィルの省略

上記のようにして，データ・アレイ・アクセスを遅延させた場合，RC への**リフィル**を行うことは難しくなる．これは，あるオペランドがヒットと判定された後，データ・アレイ・アクセスまでの間に，RC の置き換えが発生すると，動作に矛盾が生じるためである．このため，AORC では RC へのリフィルを省略する．

後の 5. 章で述べるように，AORC の場合は，リフィルを省略しても直ちに性能が下がるわけではなく，MRF のポートを 1 つ程度増やすことで十分緩和可能である．また，リフィルを省略した場合，そのためのポートを RC から削減できるため，その回路規模を縮小することができる．

4.3　RC への書き込み

前述したように，MRF への書き込みは，一旦ライト・バッファ上へバッファリングされた後，サイクル・スチールによって行われる．レジスタ・アクセスの際，このライト・バッファ上にある値に対してアクセスがおきる場合がある．このような場合，動作の正しさを保証するため，バックエンドをストールさせ，MRF への値の書き込みを完了した後に，あらためて MRF から値を得るものとする．これは，ライト・バッファ上に存在する値は，同時に RC にも書き込まれているため，ほとんどの場合は RC から値を取得できることが期待できるためである．また，ライト・バッファからのフォワーディングを行わないことにより，そのためのポートを増やさずに済む．

5.　評　　価

5.1　評価方法

我々の研究室で開発したシミュレータである**鬼斬弐**に対し，提案手法を実装して評価を行った．

5.1.1　ベンチマーク

評価には，SPEC CPU2000 [4] ベンチマーク・セットの中から，21 本のアプリケーションを使用した．アプリケーションのコンパイルには gcc 4.2.2 を用い，コンパイル・オプションについては，SPEC CPU2000 に含まれる環境設定スクリプトが指定するものに従った．各ベンチマークの入力データ・セットには train を用い，最初の 1G 命令をスキップして，続く 100M 命令を実行した．

表 2　プロセッサの構成

パラメータ	値
ISA	Alpha
fetch width	4 inst.
execution unit	int : 2, fp : 2, mem : 2.
instruction window	int : 32, fp : 16, mem : 16
main register file	int : 128, fp : 128
branch prediction	8KB g-share
misspenalty	10～13 cycle
BTB	2K entry, 4-way
RAS	8 entry
L1C	32KB, 4-way, 64B/line, 3 cycle
L2C	4MB, 8-way, 64B/line, 10 cycle
main memory	200 cycle

5.1.2　評価モデル

LORC，バイパス・バッファ，そして AORC について，以下のモデルを実装して評価した．

- **LORC**　LORC を用いたモデル．RC の構成は容量に関わらずフルアソシアティブとし，置き換えは LRU で行った．また，ヒット・ミス予測に関しては完全にヒットするものとした．
- **LORC PERFECT**　LORC において，RC に必ずヒットするモデル．このモデルは，1 サイクルでアクセス可能なレジスタ・ファイルと等価であり，LORC の性能向上の上限を与える．
- **BB**　バイパス・バッファを用いたモデル．このモデルはパイプライン化されたレジスタ・ファイルと性能的に等価である．
- **AORC**　AORC を用いたモデル．RC の構成はフルアソシアティブであり，置き換えは，LRU よりも単純な実装で済む NLU（Not-last used）を用いた．

RC のエントリ数については，LORC では 8,16,32 の 3 通りに，AORC では，8 と 16 の 2 通りに幅を振って測定を行った．また，AORC のライト・バッファの容量は，一律 8 エントリとした．

5.1.3　構　　成

評価したプロセッサのパラメータを**表 2** に示す．分岐予測ペナルティが範囲を持つのは，モデルによってパイプライン段数が異なるためである．

5.2　評価結果

各モデルの IPC を**図 9** と**図 10** に示す．この IPC は，各ベンチマークごとの IPC をバイパス・バッファのそれによって正規化し，さらにベンチマーク全体で平均化したものである．

AORC の構成については，図 9 は MRF のアクセスに 1 ステージを割り当てたもの，図 10 は MRF のアクセスに 2 ステージを割り当てたものの性能である．

グラフからは，以下のことが読み取れる：

- MRF のポート数が 4 ポートの場合，AORC の性能低下

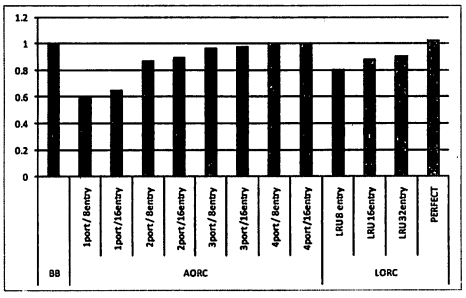

図9　平均相対 IPC（MRF のレイテンシが 2 サイクルの場合）

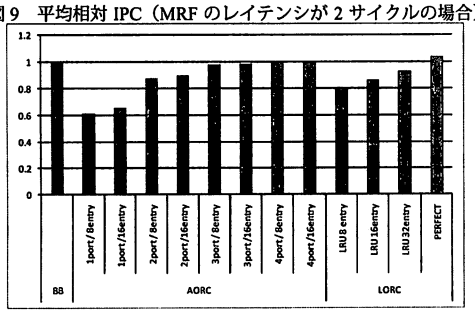

図 10　平均相対 IPC（MRF のレイテンシが 3 サイクルの場合）

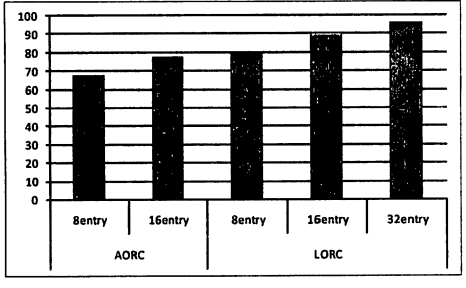

図 11　キャッシュ・ヒット率（MRF のレイテンシが 2 サイクルの場合）

率は最大で 0.5%程度である．

- AORC は，MRF のポート数が 3 ポートより少ない場合，徐々に性能が低下する．特にポート数が 1 の場合の性能低下は激しく，40%以上の性能低下を起こす．
- LORC PERFECT の性能向上は，MRF のレイテンシが 2 サイクルの場合で 1.6%，3 サイクルの場合で 3.5%である．これは，2. 章で述べた（1）パイプラインの深化による一般的な問題による影響が軽微であり，改善による LORC の伸びしろが少ないことを意味する．

5.2.1　レジスタ・キャッシュ・ヒット率

各モデルにおける，RC へのヒット率を**図 11** に示す．このヒット率は，各ベンチマークにおけるヒット率を平均化したものである．なお，MRF のレイテンシやポート数に関しては，それぞれ変化させた場合でも，傾向はほぼ一致していたため，MRF のレイテンシを 2 サイクルとし，ポート数を 4 としたもののデータを示す．

エントリ数が同じ場合，AORC のヒット率は LORC よりも 10%程度低い．これは，AORC がリフィルを省略していることによる影響であると考える．

先の図 9 より，RC のエントリ数を 8 ないしは 16 とした場合，LORC は前者で 19%，後者で 12%の性能低下を見せている．一方，同図の AORC では，ポート数が 4 の場合，その性能低下は RC のエントリ数が 8 の場合で 0.5%，16 の場合で 0.2%である．これにより，LORC で深刻な性能低下を起こすようなヒット率であっても，AORC では，ポート数が十分な場合には性能低下を起こさないことがわかる．

6. おわりに

本稿では，回路面積の縮小を目的とする，AORC の議論と評価を行った．評価の結果，提案手法では性能をほとんど落とすことなく，MRF のポート数を大きく減らすことができることを確かめた．

今後の課題としては，提案手法を SMT に適用した場合の評価がある．1. 章で述べたように，SMT は，物理レジスタを巨大化させる大きな動機であり，提案手法による回路面積削減の恩恵を大きく受ける．このため，今後はこの SMT に対して実装を行った場合の評価を行いたい．

謝　辞

本論文の研究の一部は，文部科学省 科学研究費補助金 No. 20300015 による．

文　献

[1] J. A. Butts and G. S. Sohi. Use-Based Register Caching with Decoupled Indexing. In *Proceedings of the 31th International Symposium on Computer Architecture(ISCA)*, pp. 302–313, 2004.

[2] J. L. Crutz, A. Gonzalez, and M. Valero. Multiple-Banked Register File Architecture. In *Proceedings of the 27th International Symposium on Computer Architecture(ISCA)*, pp. 316–325, 2000.

[3] K.J. Kim, J.M. Youn, S.B. Kim, J.H. Kim, S.H. Hwang, K.T. Kim, and Y.S. Shin. A novel 6.4 μm2 full-cmos sram cell with aspect ratio of 0.63 in a high-performance 0.25 μm-generation cmos technology. *VLSI Technology, 1998. Digest of Technical Papers. 1998 Symposium on*, pp. 68–69, 1998.

[4] The Standard Performance Evaluation Corporation. *SPEC CPU2000 suite*
http://www.spec.org/cpu2000/.

[5] Y. Tatsumi and H.J. Mattausch. Fast quadratic increase of multiport-storage-cell area with port number. *Electronics Letters*, Vol. 35, No. 25, pp. 2185–2187, 1999.

[6] Neil H. E. Weste, David Harris, and Addison Wesley. *CMOS VLSI Design -a circuits and systems perspective 3rd edition*. Pearson/Addison-Wesley, 2005.

[7] N.C. Yung, R.; Wilhelm. Caching processor general registers. *Proceedings of the International Conference on Computer Design(ICCD)*, pp. 307–312, 1995.

[8] 塩谷亮太, 入江英嗣, 五島正裕, 坂井修一. 回路面積指向レジスタ・キャッシュ. 先進的計算基盤システムシンポジウム SACSIS, 2008.

[9] 山本哲弘, 安藤秀樹, 島田俊夫. 先行実行を利用したロード命令のレイテンシ削減および正確なスケジューリング手法. 先進的計算基盤システムシンポジウム SACSIS, pp. 403–410, 2006.

[10] 小林良太郎, 梶山太郎, 島田俊夫. クリティカル・パスに着目した階層型レジスタ・ファイル. 先進的計算基盤システムシンポジウム SACSIS, pp. 33–40, 2006.

[11] 小林良太郎, 堀部大介, 島田俊夫. 物理レジスタ番号の割り当て順に着目したレジスタ・キャッシュの高精度化手法. 先進的計算基盤システムシンポジウム SACSIS, pp. 13–22, 2006.

[12] 三輪忍, 一林宏憲, 入江英嗣, 五島正裕, 富田眞治. 小容量 RAM を用いたオペランド・バイパスの複雑さの低減手法. 情報処理学会論文誌 コンピューティングシステム ACS, 第 48 巻, pp. 58–69, 2007.