Vol.2009-ARC-185 No.8
2009/10/26

リオーダ・バッファのハードウェア量削減

岩原佑磨[†1]　安藤秀樹[†1]

スーパスカラ・プロセッサにおいて, リオーダ・バッファ(ROB: reorder buffer) は正確な例外を実現するために必要な機構である．一般に, ROB のハードウェア量は多く，大きなチップ面積を要する．さらに, フェッチされた全ての命令およびコミットされた全ての命令がアクセスし稼働率が高く，消費電力が大きい．これに対して，我々は，ROB に従来格納されていた情報の内，例外処理に関係する情報については，全ての命令について，フェッチからコミットまで保持する必要は必ずしもないことに着目した．本論文では，例外関係の情報について最小の期間だけ保持し，ハードウェア量と電力を削減する方式を提案する．評価の結果，128 エントリの従来の ROB に対し，ほとんど性能を低下させることなくハードウェア量を 60%削減できることがわかった．また，消費電力は，ROB を実現するハードウェアにおいては 53%，プロセッサ全体においては 10%削減できることを確認した．

Reducing Hardware Amount of Reorder Buffer

Yuma Iwahara[†1] and Hideki Ando[†1]

The reorder buffer (ROB) is required for a precise exception in a superscalar processor. In general, the hardware amount of the ROB is large, and it required a large area. Furthermore, it consumes a large power because every fetched and committed instruction accesses it. Considering that the information required for exception handling is not necessarily required to hold from the fetch until the commit, this paper proposes a scheme of the ROB, which reduces the hardware amount and power consumption, by holding the information during the minimum period. Our evaluation results show that our scheme reduces the hardware amount of the ROB by 60%, and the power consumption is reduced by 53% and 10%, regarding the hardware for the ROB and the whole processor, respectively, without little performance loss.

1. はじめに

サポートするインフライト命令を増加させれば，プロセッサの性能を向上させることができる．これは，命令レベル並列 (ILP: instruction-level parallelism) およびメモリ・レベル並列性 (MLP: memory-level parallelism) を多く利用できるからである．しかし, インフライト命令を増加させるためには, プロセッサ資源を増加させなければならない．これにより回路規模が増大, 複雑化し，面積，消費電力の増大やクロック速度の低下という問題を引き起こす．インフライト命令を増加させるために増加させなければならない資源のひとつに，**リオーダ・バッファ**(ROB: reorder buffer) がある．

ROB は，アウト・オブ・オーダに命令を実行するスーパスカラ・プロセッサにおいて, 正確な例外を実現するために用いられる[1)]．現在のプロセッサにおいても, ROB の容量は大きく, 大きなチップ面積を要する．また, フェッチされた全ての命令，およびコミットされた全ての命令がアクセスし稼働率が高く, 大きな電力を消費する．このため, ROB の容量を小さくし, 面積, 電力を削減することは重要な課題である．

本論文では，ROB が保持する情報を少なくすることにより ROB のハードウェア量を削減する方式を提案する．具体的には，PC と例外原因コードに着目した (本論文では，例外として，通常の例外以外に分岐予測ミスも例外として扱う)．これらは，従来の ROB では，全てのエントリに保持されているが，必ずしもその必要はない．まず，PC は例外からの再開のために用いられるだけなので，例外を生じる可能性のない命令については保持する必要がない．また，例外処理においては, 最も古い命令の例外が処理され, 再開可能な場合，それに後続する命令は再実行される．よって，すでに古い命令が例外を生じていたなら，それに後続する命令の PC と例外原因コードを保持する必要はない．以上のような事実から，PC と例外原因コードを必要最小限の記憶領域で最短の期間保持し，ROB のハードウェアを小さくすることを提案する．

本論文の残りの節は，以下のように構成される．2 節で関連研究について述べる．3 節で ROB の基本的な構成と機能の説明を行い，4 節で提案手法について述べる．5 節で評価結果を示し，6 節で本論文をまとめる．

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

2. 関 連 研 究

チェックポイント回復方式は，ROB を用いずに正確な例外を実現することができる点で，ROB のハードウェア量，消費電力削減の問題に対する解決法として有望である．しかし，単純に適用しただけでは, ROB 削除の代わりにチェックポイントにおける状態保持のためのハードウェア量が増加してしまう．また，チェックポイントの作成には，構造 (たとえばマップ表) 間の高いバンド幅が必要であり，ハードウェアが複雑化する．さらに，分岐予測ミス時に，チェックポイントからの命令の再実行が必要であり，電力を多く消費する．

ハードウェア量の問題に対して，Akkary らは, 選択的にチェックポイントを作成することを提案した[2]．従来，分岐ごとにチェックポイントを作成していたが，これを信頼度の低い分岐に対してのみ作成する．これによりサポートするインフライト命令数に対するチェックポイントのハードウェア量は削減される．しかし，分岐予測の信頼度推定器が必要であり，その分ハードウェア量は増加する．また，複雑度，電力の問題は解決されない．

Cristal らは，ROB とチェック・ポイント回復方式をハイブリッド化することにより，分岐予測ミス時の再実行による電力問題を緩和した[3)–5)]．実行における最近の命令について，pROB(psuedo ROB) と呼ばれる ROB に格納し，その中にある分岐について予測ミスが生じたときは，チェックポイントに戻るのではなく，pROB を用いて状態の回復を行う．しかし，pROB が保持する情報は従来の ROB と同じであり，加えてチェックポイントを保持しなければならないから，ハードウェア量の削減は難しい．また，チェックポイント作成の複雑度は回避されない．

3. ROB 入門

ROB は，アウト・オブ・オーダ実行を許しながらも，正確な例外を実現するための機構である[1)]．ROB は FIFO で構成され，フロントエンドで命令は ROB のエントリをプログラム順で割り当てられる．命令が実行され例外が生じた場合，この時点では処理されず，対応する ROB エントリに例外原因コードが記録される．命令は ROB への格納順に，すなわちプログラム順でコミットされる．その際，当該命令の実行中に例外を生じていたなら，それ以降の命令は無効化される．そして，例外が処理され，必要であれば，例外命令から再フェッチされ，実行が継続される．

3.1 ROB が保持する情報

ROB が保持する情報は，レジスタ・リネーミングの手法によって異なる．本論文では，MIPS R10000 タイプのリネーミング手法[6)] を仮定する．すなわち，コミットされた値と実行が終了したがまだコミットされていない命令の結果の両方を保持するレジスタ・ファイルを持つプロセッサにおけるレジスタ・リネーミング手法を仮定する．このタイプの ROB の各エントリは以下のフィールドを持つ:

- PC: プログラム・カウンタ
- ecode: 例外原因コード
- dreg: 論理デスティネーション・レジスタ番号
- ppreg : dreg に以前に割り当てられていた物理レジスタ番号
- R: 実行が終了したことを示すフラグ

これらは全て正確な例外を実現するために必要なものである．

なお，本タイプ以外に，ROB にコミット待ちの実行結果を保持するタイプがあるが，本論文で提案する手法は適用可能である．

3.2 例 外 処 理

例外処理は次のようにして行われる．ROB の先頭の命令の R ビットがセットされていれば，ecode を参照し，実行中に例外を生じたかどうかをチェックする．生じていなければ，先頭のエントリを削除し，次のエントリをチェックする．生じていれば，例外命令以後の命令を無効化し，マップ表を例外命令がリネームされた直前の状態に復元する．これは，ROB の末尾から先頭に向かって，dreg と ppreg を読み出し，マップ表を修復することによって行われる．その後，OS に制御を移し，例外処理が行われる．例外処理が終わると, 再開可能であれば例外命令から実行が再開される．

なお，上述のマップ表の修復には時間がかかるが，これが性能に悪影響を与える場合，具体的には，分岐予測ミスからの回復には，別の方法がとられることがある．この方法では，従来のマップ表以外にコミット時に更新するマップ表 (**リタイアメント・マップ表**) を用意し，例外検出時には，リタイアメント・マップ表を従来のマップ表にコピーする．この方法においては，マップ表の修復時間が短縮され，ROB のエントリの ppreg が不要となるが，代わりに，フロントエンドで割り当てた物理レジスタ番号を保持しなければならない．本論文では，リタイアメント・マップ表は仮定しない．

4. ROB のハードウェア削減方式

3 節で述べたように, 従来の ROB は全てのエントリに PC と例外原因コードを保持していた．しかし、これらは必ずしも必要ない．なぜなら、まず第 1 に、命令の中には，例外を

生じる可能性のない命令があるから，PC と例外原因コードを保持する領域は必要に応じて用意すれば良い．次に、例外処理においては, 最も古い命令の例外が処理され, 再開可能な例外においては，それに後続する命令は再実行されるから，PC と例外原因コードは最も古い命令についてのみ保持すれば良い．以上のような事実から，PC と例外原因コードを必要最小限の記憶領域で最短の期間保持し，ROB のハードウェアを小さくすることを提案する．

これらの情報を保持しなければならない期間を考える．まず，命令を ROB に割り当てる際，その命令が例外を生じる可能性があるなら，例外からの再開に備えて PC を記憶する必要がある．実行終了時に，例外が生じたかどうかが判明するが，例外を生じていなければ，記憶した PC は破棄することができる．例外を生じていた場合，

(1) より古い命令で未処理の例外が存在するなら，当該命令は再実行されるから，保持していた PC を破棄することができる．

(2) そうでなければ，本例外が最も古い例外であり，コミット時に処理されなければならない可能性があるから，PC を保持しつづけ，例外原因を記憶する．

(2) の場合において，当該命令より新しい命令が過去に例外を生じ，その結果，PC と例外原因をすでに記憶しているなら，当該命令のそれらで置き換える．

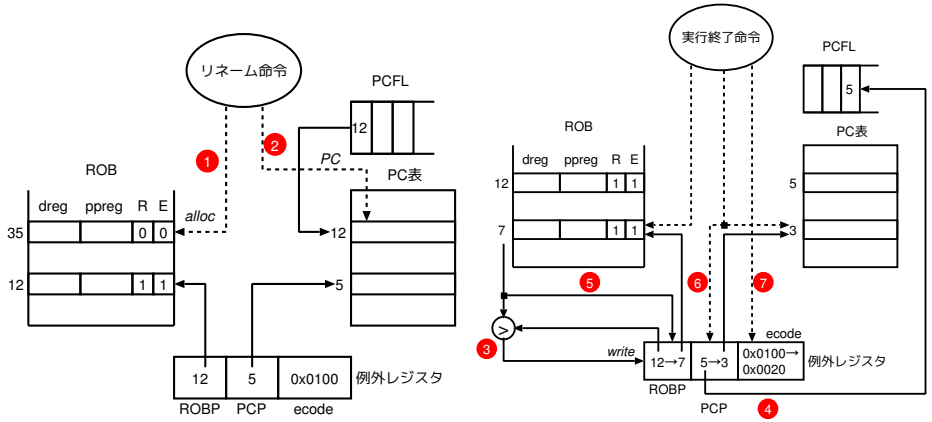
4.1 実　　装

上述の議論からわかるように，PC と例外原因を保持する記憶領域は ROB とは別に用意すればよい．PC は少なくとも ROB 割り当てから実行終了まで保持しなければならないので，同時に複数の PC を保持しなければならない．そこで，PC を保持する **PC 表**と呼ぶ表と，その空きエントリを記憶する **PCFL** と呼ぶフリー・リストを用意する．

一方，例外原因は最も古い命令について保持すれば良いから 1 つのレジスタが必要なだけである．これに加えて，例外処理時に例外命令の PC が必要であり，その PC は PC 表にあるから，記憶されている PC 表のエントリ番号を保持するレジスタを用意する．また，命令が例外を生じたときに，過去に例外を生じていたなら，それとの「年齢」の上下を判断しなければならないから，記憶している例外命令の年齢情報が必要である．これを ROB のエントリ番号で表すこととする．以上 3 つの情報を記憶するレジスタを 1 つにまとめ，**例外レジスタ**と呼ぶこととする．まとめると，例外レジスタは以下のフィールドからなる:

- ecode: 例外原因コード
- PCP: 例外命令の PC を保持する PC 表のエントリ番号
- ROBP: 例外命令が対応する ROB のエントリ番号

以上により，ROB のエントリから PC と例外原因コードのフィールドを除去できるが，



(a) フロントエンドでの動作　　(b) 実行終了時の動作

図 1　動作

命令がコミットされる際に例外を生じたかどうかは知る必要がある．このため，ROB のエントリに，例外を生じたことを示す 1 ビットのフラグ E を新たに用意する．

4.2 動　　作

図 1 を用いて動作を説明する．命令がリネームされたら，ROB のエントリを割り当て，dreg，ppreg を初期化し，R，E をクリアする (図 1(a) の (1))．さらに，命令が例外を生じる可能性のある命令の場合，PCFL より PC 表の空きエントリを得，そのエントリに当該命令の PC を書き込む (同図 (a) の (2))．

実行が終了したら，以下を行う．例外が生じていなければ，割り当てられている PC 表のエントリを解放する．すなわち，そのエントリ番号を PCFL に追加する。例外を生じていれば，例外レジスタを参照し，すでに例外が記録されているかをチェックする．その結果，

(1) 記録されていれば，ROBP と当該命令が割り当てられている ROB エントリの番号を比較し，どちらが古いかを判断する (図 1(b) の (3))．その結果，

- 実行終了命令の方が古ければ，例外レジスタの PCP を PCFL に追加し (同図 (b) の (4))，現在記憶されている例外命令に割り当てられている PC 表のエントリを開放する．そして，ROBP に当該命令が割り当てられている ROB のエントリ番号を書き込む (同図 (b) の (5))．さらに，PCP フィールドに当該命令の PC が記憶

されている PC 表のエントリ番号を書き込む (同図 (b) の (6))．そして，例外レジスタの ecode フィールドに例外原因コードを書き込む (同図 (b) の (7))．最後に，ROB の割り当てられたエントリの E ビットをセットする．

- そうでなければ，実行を終了した当該命令に割り当てられている PC 表のエントリを解放する．すなわち，そのエントリ番号を PCFL に追加する．

(2) 記録されていなければ，例外レジスタの ROBP フィールドに当該命令が割り当てられている ROB のエントリ番号を，PCP フィールドに当該命令の PC が記憶されている PC 表のエントリ番号を，ecode フィールドに例外原因コードを書き込む．

例外を生じた命令がコミットされたら，つまり，E がセットされた ROB のエントリをコミットしたら，例外レジスタの ecode と PCP を参照して，例外原因と例外命令の PC を得て例外処理を開始する．

5. 評　　価

提案手法を評価するために，SimpleScalar をベースとしたシミュレータを作成した．命令セットは，MIPS R10000 の ISA の拡張セットである SimpleScalar/PISA である．ベンチマーク・プログラムとして，SPECint2000，SPECfp2000 からそれぞれ 8 本を使った．プログラムは，gcc ver. 2.7.2.3，コンパイル・オプション -O6, funroll-loops でコンパイルした．**表 1** に，各ベンチマーク・プログラムにおけるロードのメモリ・アクセス率と分岐予測ミス率 (1K 命令に対するミスの回数) を示す．

消費電力については，次のようにして評価を行った．ある部品で消費する電力は，その部品の 1 回の動作によって消費される電力にその動作回数を乗じて得られる．動作回数は，前述した SimpleScalar ベースのシミュレータによって得られる．アレイと CAM の構造を持つ部品における 1 回の動作で消費される電力は，我々が修正した CACTI[7] を用いて得た．修正箇所は，配線に関するパラメータであり，元の CACTI では，0.8μm のパラメータを，仮定する LSI テクノロジに合わせてスケーリングしているが，我々はそうではなく，そのテクノロジにおける実際のパラメータ[8] を使うようにした．この修正は，正確な電力測定をする上においては重要である．一方，組合せ論理からなる部品については，Wattch[9] で使われている電力モデルをスケーリングして用いた．35nm テクノロジ，電源電圧 0.9V，クロック速度 3GHz を仮定した．なお，測定した電力は，クロックを除く動的電力である．

以下のモデルについて評価を行った．

- BASE：従来の ROB を用いたモデル
- RROB：本提案手法を用い, ROB のハードウェア量を削減したモデル

両モデルの基本となるプロセッサ構成を**表 2** に示す．

表 1 ベンチマーク・プログラムのメモリ・アクセス率と分岐予測ミス率

プログラム	メモリ・アクセス率 (%)	分岐予測ミス率 (ミス回数/K 命令)
bzip	0.5	11.4
gzip	0.1	14.4
mcf	16.2	26.3
perlbmk	0.0	11.3
vpr	0.8	11.6
gcc	0.1	3.2
parser	0.0	8.8
vortex	0.1	1.3
ammp	3.8	2.3
applu	2.9	2.8
apsi	0.2	1.8
art	27.6	1.1
equake	2.6	7.2
mesa	0.5	4.4
mgrid	0.9	0.1
swim	5.4	0.0

5.1 PC 表のサイズによる性能変化

PC 表のサイズに対して，性能がどのように変化するかを測定した．PC 表が不十分であると，その不足によってパイプラインがストールし，性能が低下する．測定結果を**図 2** に示す．ここで, 横軸は PC 表エントリ数, 縦軸は各ベンチマークの IPC を BASE モデルの IPC で正規化した値を表す．

同図より, SPECint2000, SPECfp2000 両ベンチマークとも, PC 表は 32 エントリあれば, 十分であることがわかる．このときの BASE モデルに対しての性能低下は 0.2%である．

5.2 ハードウェア量

ROB の機能を実現するために必要となるハードウェア量を計算した．構成する部品の構成 (エントリ数とビット数) を**表 3** に示す．

表 3 から計算したハードウェア量と，BASE に対する RROB のハードウェア量削減率を**表 4** に示す．RROB モデルにおける性能低下がほとんど生じない PC 表エントリ数が 32 のとき, ROB のハードウェアを 60%削減できることがわかる．

表 2 ベース・プロセッサの構成

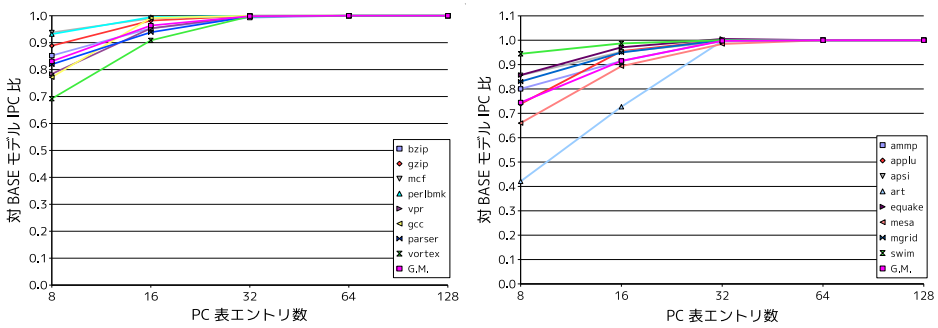
命令フェッチ幅	最大 8 命令
命令デコード幅	最大 8 命令
命令発行幅	最大 8 命令
命令コミット幅	最大 8 命令
命令フェッチ・キュー	32 エントリ
ROB	128 エントリ
命令ウィンドウ	64 エントリ
LSQ	64 エントリ
機能ユニット	8iALU, 4iMULT/DIV, 4Ld/St, 6fpALU, 4fpMULT/DIV/SQRT
命令キャッシュ	64KB, 2 ウェイ・セット・アソシアティブ, ライン長 32 バイト, 1 ポート, ヒット・レイテンシ 2 サイクル
データキャッシュ	64KB, 2 ウェイ・セット・アソシアティブ, ライン長 32 バイト, 4 ポート, ヒット・レイテンシ 2 サイクル
2 次キャッシュ	統合, 2MB, 4 ウェイ・セット・アソシアティブ, ライン長 64 バイト, ヒット・レイテンシ 12 サイクル
メインメモリ	最小レイテンシ 300 サイクル, バースト転送間隔 4 サイクル, メモリ・バス幅 8 バイト
分岐予測機構	gshare 6 ビット履歴, 8K エントリ PHT, 512 エントリ 4 ウェイ BTB, 予測ミス・ペナルティ10 サイクル
物理レジスタ	32 ビット幅 int 128 個, fp 128 個

5.3 消 費 電 力

5.3.1 ROB の電力

ROB の機能を実現するために必要となるハードウェアの消費電力を測定した．測定結果を**図 3** に示す．縦軸は，BASE モデルの ROB の消費電力で正規化した RROB の消費電力を示す．各棒グラフは，RROB を構成する部品の消費電力で内訳されている．

RROB では，ROB の機能のために要する電力が大幅に減少していることがわかる．BASE モデルと同等の性能を示す 32 エントリの PC 表の場合の電力削減は，53%である．



(a) SPECint2000　　(b) SPECfp2000

図 2 PC 表エントリ数と BASE モデルで正規化した各ベンチマークの IPC 比

表 3 各構成部品のエントリ数とデータ幅

	BASE	RROB
ROB エントリ数	128	128
PC	32 ビット	0 ビット
ecode	16 ビット	0
dreg	$\log_2$(論理レジスタ数) = 6 ビット	6 ビット
ppreg	$\log_2$(物理レジスタ数) = 8 ビット	8 ビット
R	1 ビット	1 ビット
E	0 ビット	1 ビット
ROB 計ビット数	63 ビット	16 ビット
PC 表エントリ数	0	8 〜 128
PC	0 ビット	32 ビット
PCFL エントリ数	0	8 〜 128
PCP	0 ビット	$\log_2$(PC 表エントリ数) ビット
例外レジスタ・エントリ数	0	1
ROBP	0 ビット	$\log_2$(ROB エントリ数) = 7 ビット
PCP	0 ビット	$\log_2$(PC 表エントリ数) ビット
ecode	0 ビット	16 ビット

提案手法における消費電力の大半は，ROB によって消費されていることがわかる．この理由は次のようなことである:

- 表 4 からわかるように，ハードウェア量のかなりの量を ROB が費している (PC 表 32 エントリでは，63%).
- ROB のエントリ数に対し，PC 表，PCFL のエントリ数は非常に少なく，デコーダと

表 4 各モデルの ROB，PC 表，PCFL，例外レジスタのハードウェア量 (バイト)

		RROB				
		PC 表エントリ数				
	BASE	8	16	32	64	128
ROB	1008	256				
PC 表	0	32.0	64.0	128.0	256.0	512.0
PCFL	0	3.0	8.0	20.0	48.0	112.0
例外レジスタ	0	3.3	3.4	3.5	3.6	3.8
合計	1008	294.3	331.4	407.5	563.5	883.8
削減率	N/A	70.8%	67.1%	59.6%	44.1%	12.3%

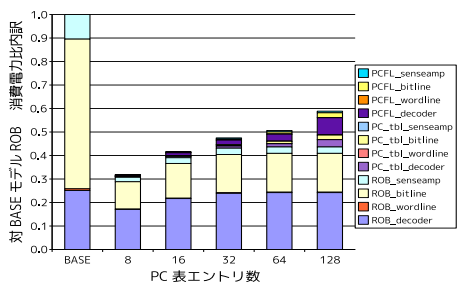


図 3 BASE モデルの ROB を基準とした消費電力比の内訳

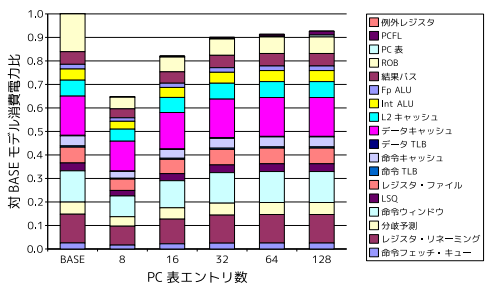


図 4 対 BASE モデル消費電力比の内訳

ビット線の電力は大幅に少なくなる．

- ROB はフェッチされた全ての命令について書き込まれ，コミットされた全ての命令について読み出される．一方，PC 表の動作のほとんどは，フェッチされた命令の内，例外を生じる可能性のある命令について書き込みが起こるのみである．

提案手法により ROB のエントリ数は減少しないが，ROB の消費電力は大きく減少している．これは，ROB のデータ幅が縮小していることによる．具体的には，ビット線が 63 本から 16 本に大幅に減っている．このため，ビット線，センス・アンプの消費電力が激減している．一方で，エントリ数が変わらないため，デコーダの電力はほどんと変化がない．

最後に，PC 表 8，16 エントリの場合の電力について注意を述べておく．これらのエントリ数での電力は，ハードウェア量の減少により削減されていることに加え，図 2 で示したように，性能が低下しており，実行時間が延びただけ電力は減少している．

5.3.2 プロセッサの電力

図 4 にプロセッサ全体の電力を示す．RROB モデルの場合では，PC 表のエントリ数を変化させた測定結果を示している．各棒グラフは，プロセッサを構成する部品の電力で内訳されている．

同図よりわかるように，性能低下がほとんど生じない PC 表エントリ数が 32 の時, RROB モデルは BASE モデルに対して 10%の消費電力削減となっている．BASE モデルでは，ROB はプロセッサの全電力の 16%という大きな部分を占めていたが，提案手法によりそれが大きく削減されたためである．なお，PC 表のエントリ数が 8, 16 の時, 消費電力が大幅に小さくなっているのは，前節で述べたように，性能が低下していることが大きく影響している．

5.4 ROB 同一ハードウェア量での理想的な性能向上

これまで，ROB を実現するために必要なハードウェア量を削減した場合について評価を行ってきたが，本節では逆に，従来の ROB と同量のハードウェアを投資したとき，提案手法で理想的にどこまで性能を向上させることができるかを評価する．この評価においては，ROB 以外の資源不足でパイプラインがストールしないよう命令ウィンドウ，LSQ，レジスタ・ファイルを十分に大きいものと仮定した．資源拡大によるクロック速度への悪影響は考えない．

表 4 に示したように，従来の ROB のハードウェア量は，128 エントリの場合で 1008 バイトであるから，およそそのハードウェア量で，RROB モデルにおいて，ベンチマーク平均で最高の性能を示す最適構成を実験により求めた．その結果，ROB が 288 エントリで，PC 表が 80 エントリが最適であることがわかった．このときのハードウェア量は 1042 バイトである．

図 5 に，BASE モデルに対する性能向上率を示す．同図よりわかかるように，整数系ではほとんど性能向上がないが (平均で 4.7%)，浮動小数点系では非常に大きな性能向上 (平均で 32.3%) を示すことがわかる．これは，表 1 からわかるように、浮動小数点系の *ammp, applu, art, swim* はメモリ・インテンシブであり，提案手法によるインフライト命令数の増加により，MLP がより多く利用できたことによると考えられる．整数系でも *mcf* は，メモリ・インテンシブであるが，分岐予測精度が低いため，プロセッサ内に十分な数のインフライト命令が蓄積されず，MLP が十分に利用されない．その他の整数系プログラムはメモリ・アクセス率が低く，かつ，分岐予測精度が低いので，インフライト命令増加の効果は低い．

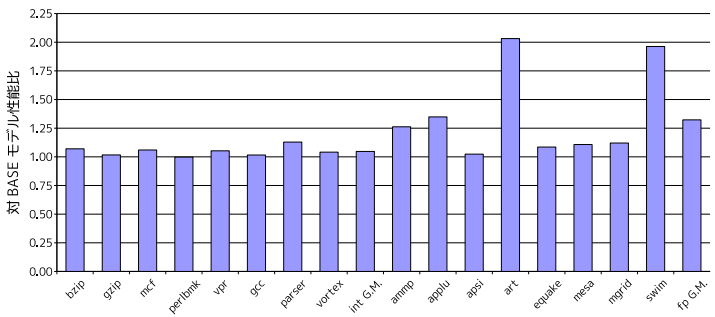


図 5 ROB 同一ハードウェア量での理想的な性能向上率

6. ま と め

本論文では, PC と例外原因コードを必要な期間だけ別に記憶し，ROB のハードウェア量の削減を行う手法を提案した. 評価の結果，ほどんど性能を低下させることなく，従来の ROB に対し 60%ハードウェア量を削減できた．また，消費電力 (クロックを除く動的電力) においては，ROB だけでは 53%，プロセッサ全体では 10%削減されることを確認した.

謝辞 本研究の一部は，日本学術振興会 科学研究費補助金基盤研究 (C)（課題番号 19500041）による補助のもとで行われた.

参 考 文 献

1) Smith, J.E. and Pleszkun, A.R.: IMPLEMENTATION OF PRECISE INTERRUPTS IN PIPLINED PROCESSORS, *Proceeding of the 12th Annual International Symposium on Computer Architecture*, pp.291–299 (1985).
2) Akkary, H., Rajwar, R. and Srinivasan, S.T.: Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors, *Proceedings of the 36th Annual International Symposium on Microarchitecture*, pp.423–434 (2003).
3) Cristal, A., Santana, O.J., Valero, M. and Martínez, J.F.: Toward Kilo-instruction Processors, *ACM Transactions on Architecture and Code Optimization*, Vol.1, pp. 389–417 (2004).
4) Cristal, A., Santana, O.J., Cazorla, F., Galluzzi, M., Ramírez, T., Pericás, M. and Valero, M.: Kiro-Instruction Processors: Overcoming the Memory Wall, *IEEE MICRO*, Vol.25, No.3, pp.48–57 (2005).
5) Cristal, A., Ortega, D., Llosa, J. and Valero, M.: Out-of-Order Commit Processors, *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, pp.48–59 (2004).
6) Yeager, K.C.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, Vol.16, No.2, pp.pp.28–40 (1996).
7) Shivakumar, P. and Jouppi, N.P.: CACTI 3.0: An Integrated Cache Timing, Power, and Area Model, *WRL Research Report 2001/2* (2001).
8) Agarwal, V., Hrishikesh, M.S., Keckler, S.W. and Burger, D.: Clock Rate versus IPC: The End of the Road for Conventional Microarchitecture, *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pp.248–259 (2000).
9) Brooks, D., Tiwari, V. and Martonosi, M.: Wattch: a framework for architectural-level power analysis and optimizations, *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp.83–94 (2000).