Vol.2011-ARC-196 No.16
2011/7/28

仮想リオーダ・バッファ方式における選択的先行実行による低消費電力化

加藤里奈[†1]　安藤秀樹[†1]

近年のコンピュータでは，クロック速度とメモリ・アクセス速度に大きな差があり，これが性能を大きく制限している．この問題を解決する手法にデータ・プリフェッチがある．基本的なプリフェッチ手法は，規則的なメモリ・アクセス・パターンにのみ有効なものであるが，不規則なパターンにも有効なプリフェッチ手法として，命令の先行実行がある．単一スレッド環境において先行実行を行う手法として，我々はこれまでに仮想 ROB（VROB: virtual reorder buffer）方式を提案した．従来の VROB は，有効にプリフェッチを行い，性能を大きく改善することができるものの，できるだけ多くの命令を先行実行するという方式であり，先行実行の効果がない命令も先行実行し，無駄な電力を消費していた．そこで本論文では，頻繁にキャッシュ・ミスを起こすロード命令とそれが直接的あるいは間接的に依存する命令のみを先行実行することにより有効な先行実行のみを行う選択的先行実行方式を VROB 方式に適用し，評価を行った．SPECfp2000 ベンチマークを用いて評価を行った結果，従来の VROB プロセッサと比較して，やや上回る性能を示しつつ，大きな電力増加を引き起こす先行実行される命令の発行キューへのディスパッチ命令数を 72%削減できることを確認した．

A Low-Power Virtual Reorder Buffer Scheme through Selective Pre-Execution

Rina Kato[†1] and Hideki Ando[†1]

In recent computers, there is a large speed discrepancy between processors and main memory, and this significantly limits the performance. Data prefetch is one of methods for solving this problem. While basic prefetch schemes are effective only for regular memory access patterns, instruction pre-execution is a scheme that are effective for irregular access patterns as well. We previously proposed a pre-execution scheme in a single threaded environment, called virtual reorder buffer (VROB). Although the previous VROB improves the performance significantly by prefetch, even instructions that do not contribute to the performance improvement ware pre-executed, because it attempted to pre-execute instructions as much as possible. This turns out to be a waste of power. This paper applies a selective pre-execution to the VROB, which pre-executes only loads that often cause L2 cache misses and instructions on which the loads depend directly or indirectly. Our evaluation results using SPECfp2000 benchmark show that our scheme reduces the number of dispatched instructions to the issue queue, which are to be pre-executed and cause extra large power consumption, by 72% with even slightly better performance, compared with the original VROB.

1. はじめに

プロセッサとメモリの間の速度差は非常に大きく，コンピュータの性能を大きく制限している．この問題を解決する手法として，データ・プリフェッチがある．データ・プリフェッチ手法の中でも，不規則なアクセスに対応可能な方法として，命令の先行実行がある．これは，本来の実行に先駆けて，事前に命令を実行する手法である．しかし，従来の先行実行手法のほとんどはマルチスレッド環境を必要としている[2),9)]．

これに対し，我々は単一スレッド環境で命令の先行実行を実現する**仮想リオーダ・バッファ**（VROB: virtual reorder buffer）方式[11)] を提案した．VROB 方式は，リオーダ・バッファ（ROB: reorder buffer）に空きがなく割り当てることができない時，割り当てないまま命令を発行キューへ挿入し，先行実行を行う手法である．従来の研究では，VROB 方式を定期要することにより，ロード・レイテンシを削減し，大きな性能向上を得られることがわかったが，できるだけ多くの命令を先行実行するため，性能向上に寄与しない命令も実行され，電力効率が悪いという問題があった．

そこで本論文では，VROB 方式に **delinquent ロード**に着目した選択的先行実行方式[6)] を導入し，評価を行う．一般に，L2 キャッシュ・ミス（本論文では，L2 キャッシュをキャッシュの最下層とする）の多くは，少数の静的ロードによって引き起こされていることが知られている[3)]．そのようなロードを delinquent ロードと呼ぶ．この性質を利用して，delinquent ロードを動的に検出し，それが直接あるいは間接的に依存する命令のみを先行実行する．これによって，不必要な命令の先行実行を削減し，消費電力を抑える．

本論文は次のような構成となっている．まず 2 節で本研究の関連研究について述べる．次に 3 節で VROB 方式，4 節で先行実行命令候補の抽出方法について説明する．5 節で評価

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

を行い，6 節で本論文をまとめる．

2. 関 連 研 究

2.1 先 行 実 行

本論文で用いる VROB 方式は，先行実行に基づくものであるが，これまでにも多くの先行実行を利用したプリフェッチ手法が提案されている[2),9)]．しかし，これらの手法の多くは別スレッドを生成して先行実行するというもので，マルチスレッド環境を必要とする．

単一スレッド環境において先行実行を行う手法として，Mutlu らは runahead 実行を提案した[8)]．この手法は L2 キャッシュ・ミスが生じると，プロセッサ状態をチェックポイントし，ミスが解決するまで runahead モードと呼ぶ特別のモードに移る．このモードでは，ミスした命令に依存していない命令を実行する．この時，ロードがキャッシュ・ミスを起こせば，データがプリフェッチされる．この手法の欠点は，runahead 実行中にはプロセッサ状態を更新する本来の命令実行を並行して行えないという点である．

2.2 先行実行方式における省電力化

先行実行を行う場合での電力の高効率化について，Mutlu らは runahead 実行方式における省電力化手法を提案している[7)]．これは過去の runahead 実行を学習し，効果の小さい命令の先行実行を抑制することで全体の実行命令数を削減する．

3. 仮想リオーダ・バッファ方式による命令の先行実行

図 1 に VROB 方式を実装したプロセッサの構成を示す．通常の構成要素に加え，①再フェッチ用 PC（**RPC**: refetch PC），②ディスパッチ・ステージから RPC へ再フェッチの開始・停止を指示する信号，③ROB から先行実行用発行キューへ ROB に空きエントリが生じたことを伝える信号，及び④先行実行用の FIFO 発行キューが追加されている．

3.1 先行実行・本実行

従来のプロセッサでは，命令に ROB を割り当てることができない場合，命令はストールする．これに対し，VROB 方式では ROB が不足している場合には，ROB 及び物理レジスタを割り当てないまま命令を発行キューへ挿入する．これを**先行ディスパッチ**と呼ぶ．先行ディスパッチされた命令は，ソース・オペランドが揃えば発行され，先行実行を行う．

先行実行は物理レジスタを割り当てられていないため，結果を保持することはできないが，バイパス論理を経由して後続命令に受け渡すことはできる．ただし，バイパス論理による結果の受け渡しは実行後 1 サイクルしか有効でない．この制約を緩和するために，フォワーディング・バッファ（FB: forwarding buffer）[1)] を用いる．FB はオペランド・タグで連想検索可能な小さなバッファであり，最近の先行実行の結果を保持している．バイパス論理による実行結果の受け渡しに失敗した場合でも，FB にその結果があれば，後続の依存命令を先行実行できる．FB からも結果値を得られなかった場合は，これらの命令は発行できず，後に本節の冒頭で示した信号③によって発行キューから削除される（3.2 節で詳述）．

命令の先行ディスパッチを開始したら，直ちにそれらの命令の再フェッチを開始し，本実行に備える．再フェッチは，再フェッチ用の PC である RPC を用いて行う．図 1②に示すとおり，RPC は先行ディスパッチを開始した際に，その最初の先行ディスパッチ命令の PC で初期化する．再フェッチした命令は，**図 2** に示すとおり再フェッチ・キュー（**RFQ**: refetch queue）と呼ぶ一時バッファへ格納する．一方，PC によってフェッチされた命令は，フェッ

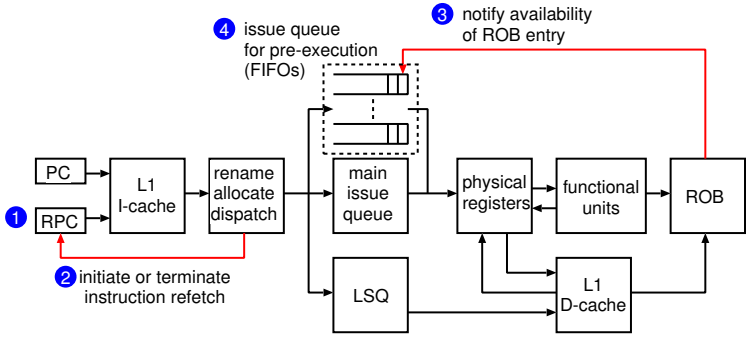

図 1 VROB プロセッサの構成
Fig. 1 Organization of processor with VROB.

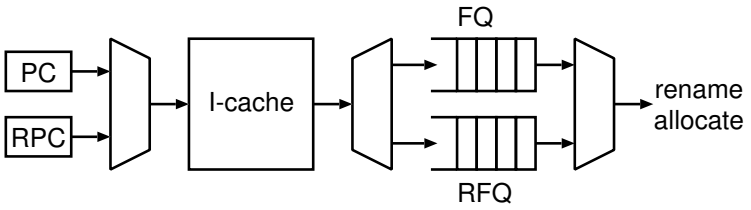

図 2 命令フェッチの構成
Fig. 2 Organization of instruction fetch.

チ・キュー（FQ）と呼ぶ別のバッファへ格納される．

再フェッチは，先行ディスパッチされた命令を全て本実行するまで継続する．再フェッチを終了するタイミングを検出するため，**先行ディスパッチ・カウンタ**と呼ぶカウンタを用意する．このカウンタは本実行されるべき命令数を表し，命令を先行ディスパッチした際にインクリメントする．一方，再フェッチした命令を発行キューへ挿入した際にデクリメントする．カウンタ値が 0 となった場合，必要な本実行は全て行われることが確定するため，再フェッチを終了し，RFQ をフラッシュする．

命令フェッチ及び再フェッチは時分割で行う．再フェッチを優先して行い，RFQ が満杯となった場合に PC によるフェッチを行う．これは本実行のスループットの方が，先行実行よりも性能において重要となるからである．また，リネーム・ステージにおける FQ または RFQ からの読み出しも，同様に時分割で行う．まず RFQ の先頭の命令について，資源割り当てが可能かを確認し，可能であれば RFQ から命令を読み出す．不可能であれば FQ から読み出す．

3.2 先行ディスパッチ命令の削除

3.2.1 概　　要

先行ディスパッチされた命令は，以下の場合においては発行される前に発行キューから削除されなければならない．

(1) 先行実行する前に，本実行に必要な資源が利用可能となった場合．この場合，命令は本実行可能となるため，もはや先行実行を行う必要はない．

(2) バイパス論理及び FB による先行実行結果の受け渡しに失敗した場合．この場合，後続の依存命令は発行不能となり，発行キューに取り残される．

(1) の場合に対応するため，次のようにして資源の利用可能性を発行キュー内の命令に伝達する．ROB から命令がコミットされ空きエントリが生じたら，そのエントリの ID 番号（3.3 節で述べる仮想エントリ番号）を発行キューへ放送する．発行キューでは，先行ディスパッチされた命令について，そのエントリが，もし ROB に空きがあれば自身が割り当てられたはずのエントリかどうかを判断する．もしそうであれば，その命令を発行キューから削除する．削除された命令は必要な資源を割り当てられた上で，RFQ から発行キューへ再ディスパッチされる．なお，厳密には ROB が利用可能であっても，その他の資源が割り当て可能であるとは限らず，直ちに再ディスパッチ可能であることは保証されない．しかし，すべての資源のバランスがとれた設計においては，ほぼ良い近似を示すと考えられる．

この方法の欠点としては，(2) の場合の命令の削除としてはタイミングが遅いことが挙げられる．この場合においては本来，命令は先行実行結果の受け渡しに失敗した時点で削除されるべきである．しかし，実際には結果受け渡しの成功率は非常に高く，(2) の状態が生じることは稀である．従って，これによる性能への影響は小さい．

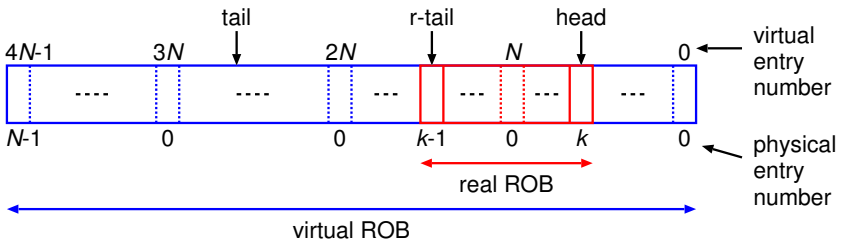

図 3　仮想 ROB（$M = 4$）
Fig. 3　Virtual reorder buffer（$M = 4$）.

3.3 ROB の利用可能性の伝達

リネーム時に ROB が満杯でエントリを割り当てることができなければ，もしも空いていたとするなら割り当てられたはずの ROB のエントリを命令に割り当てる．これを**先行割り当て**と呼ぶ．これは概念的には ROB を仮想的に拡大したことに相当し，ROB に関する資源制約を緩和し先行実行を可能とする．

図 3 に仮想的に拡大された ROB の概念図を示す．この図では，実エントリ数 N の ROB を $M = 4$ 倍に拡大した場合を例示している．図において，ROB の上部の数字は仮想的に拡大された ROB 全体の仮想エントリ番号を表し，下部の数字は ROB を循環バッファで実装した時の物理エントリ番号を表している．(仮想エントリ番号 mod N) が物理エントリ番号となる．従って，1 つの物理エントリには 1 つの実エントリと $(M-1)$ 個の仮想エントリがマッピングされる．以後，仮想的に拡大された ROB 全体を**仮想 ROB**，実在の ROB を**実 ROB** と呼ぶ．

仮想 ROB の先頭と末尾を，それぞれ head，tail ポインタが指す．一方，実 ROB の先頭は仮想 ROB と同一であり head ポインタが指すが，末尾は別途 **r-tail**（real tail）と呼ぶポインタが指す（これらのポインタは全て仮想エントリ番号を持つ）．リネーム・ステージにおいて ROB が満杯の場合，命令には仮想エントリを割り当て，tail ポインタのみを更新する．これが前述した先行割り当てである．

図 4 に，先行ディスパッチされる命令の発行キューへの挿入及び削除の様子を表す．命令は，先行割り当てされた ROB エントリの仮想エントリ番号と共に発行キューへ挿入される

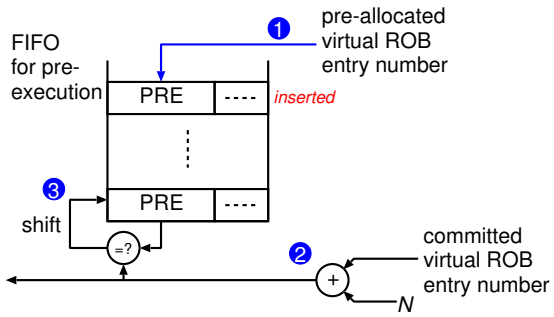

図 4 発行キューへの挿入及び削除（N は実 ROB サイズ）
Fig. 4 Insert and removal of instructions in issue queue.

（①）．仮想エントリ番号を保持するために，発行キューの各エントリに **PRE**（pre-allocated ROB entry）フィールドを追加する．資源の利用可能性を伝達するため，ROB から命令がコミットされたら，その ((仮想エントリ番号 + N) mod ($N \times M$)) が発行キューへ放送される（②，図 1③も参照）．この値は，解放された物理エントリが次に割り当てられる仮想エントリの番号を表している．発行キューでは，先行ディスパッチされた命令について，その PRE フィールドが保持している仮想エントリ番号と放送されてきたエントリ番号とを比較する．一致すれば，そのエントリに割り当てられた命令に先行割り当てされた ROB のエントリが利用可能となったことを意味する．この場合，その命令を発行キューから削除する（③）．なお，3.1 節で述べたように，先行ディスパッチされた命令は即座に再フェッチされるため，削除された命令は多くの場合，すでに RFQ の先頭で待ち合わせている．

4. 先行実行命令候補の抽出

本節では，先行実行命令候補の抽出方法[6] について説明する．本手法における先行実行命令は，delinquent ロードとそれが直接，あるいは間接的に依存する命令（**TP 命令**: transitive producer）のみとする．それぞれの抽出の方式について順に説明する．

4.1 delinquent ロードの検出

第 1 節で述べたとおり，delinquent ロードとは，頻繁に L2 キャッシュ・ミスを起すロードである．そのため，ロードが delinquent かどうかを判断するには，一定期間での L2 キャッシュ・ミス回数を計測すればよい．delinquent ロード検出のため，ロードの命令アドレスをインデクスとする **MCT**（miss count table）と呼ぶ表を用意する．この表の各エントリは，以下の内容を保持する：

- 有効ビット（V フラグ）
- L2 キャッシュ・ミス回数を数えるカウンタ
- ロード命令の PC の一部（タグ）
- 4.2 節で述べる TP 探索がすでに行われたかどうかを示すビット（C フラグ）

L2 キャッシュ・ミスを起したロードはコミットされる際，MCT の対応するエントリを参照する．エントリにヒットしたら，そのエントリのカウンタをアップする．ミスの場合，カウンタを 1 にセットし，C フラグをリセットする．L2 キャッシュ・ミスの頻度を測るために，一定間隔で全有効ビットをクリアし，MCT をリセットする．ロードがカウンタを増加させる際，カウンタ値があらかじめ定めた閾値に到達し，C フラグがセットされていなければ，そのロードを delinquent と判定する．delinquent ロードは，次回実行時に，4.2 節で述べる TP 探索を起動する．TP 探索が終了したら，C フラグをセットし，以後，現在のインターバルにおいては，再び TP 探索しないよう制御する．

4.2 TP 命令の動的探索

delinquent ロードの TP 命令の探索のため，delinquent ロードに先行して実行される命令を解析する．そのため，コミットされた命令を順に蓄える **RIB**（retired instruction buffer）と呼ぶ FIFO のバッファを用意する．RIB には，命令の PC と論理デスティネーション・レジスタ番号および論理ソース・レジスタ番号を保持する．RIB はコミットされた順に命令を保持するため，プログラム順で命令が並ぶ．

TP 命令の探索は 2 段階で行われる．第 1 段階は，RIB を起動し，コミットされた命令を格納する段階である．RIB は，通常，低電力モード（電源電圧降下もしくは電源供給停止）にあり，トリガ信号により起動し，コミットされた命令を格納していく．そして，第 2 段階の TP 命令抽出が終了したら低電力モードに遷移する．このように必要に応じて起動することにより RIB による無駄な電力消費を抑える．

まず，MCT によって delinquent ロードが検出されたら，その命令をトリガとしてマークする．これは，命令キャッシュにフラグ（**TG フラグ**）を用意して記憶する．その後，トリガ命令がフェッチされたら MCT を参照する．C フラグがセットされていなければ，TP 探索は行われていない．トリガ命令のデコード時に RIB を起動し，以後，トリガ命令がコミットされるまで，先行してコミットされる命令を RIB に書き込む．

第 2 段階は，トリガ命令がコミットされるタイミングで開始される．この段階では，RIB を末尾から先頭に向かって読み出し，データフロー解析を行い，TP 命令を抽出する．この

```
1:  LIVE := sreg of delinquent load;
2:  foreach inst ∈ RIB (from tail to head) {
3:    if (LIVE = φ) break;
4:    if (dreg o inst ∈ LIVE) {
5:      mark inst as "TP";
6:      LIVE := LIVE − dreg of inst;
7:      LIVE := LIVE ∪ sreg(s) of inst;
8:    }
9:  }
```

図 5 TP 命令抽出アルゴリズム
Fig. 5 TP instruction extraction lgorithm.

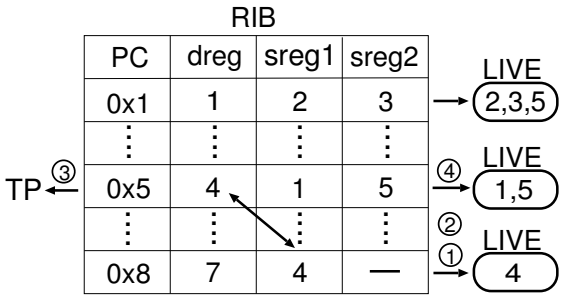

図 6 TP 命令抽出時の動作
Fig. 6 Example of TP instruction extraction.

解析では，データフロー・グラフを delinquent ロード（トリガ命令）を始点としてエッジを逆方向に遡り，至ったノードに対応する命令を TP 命令としてマークする．具体的には，**図 5** に示すアルゴリズムで実現する．また，この時の動作を**図 6** に示す．

まず，図 6 の①に示すように delinquent ロードのソース・レジスタ（*sreg*）を集合 *LIVE* の初期値とする（図 5,1 行目）．次に，RIB を末尾から先頭に向かって順番に読み出し，*LIVE* が空になるまで，以下の動作を行う．②に示すように読み出した命令のデスティネーション・レジスタ（*dreg*）が *LIVE* に属しているなら，③に示すようにその命令を TP としてマークする（5 行目）．そして，*LIVE* から *dreg* を除き（6 行目），⑤に示すように *sreg* を *LIVE* に加える（7 行目）．

集合 *LIVE* はビット・ベクタで表す．つまり，レジスタ i が *LIVE* に属していれば，第 i 番目のビットを 1 とする．こうすれば，集合演算はビット毎の論理演算という単純なハードウェアで行うことができる．また，TP としてのマークは，命令キャッシュにそれを保持するビット（**TP フラグ**）を用意して記憶する．以上の TP 探索が終了すると，MCT の C ビットをセットする．また，RIB を低電力モードとし電力消費を抑える．

以後，フェッチされた命令が TG 命令または TP 命令とマークされている場合，それらの命令は先行実行の対象となる．ROB が不足し，先行実行を行う状態になったら，TG または TP 命令のみを先行ディスパッチする．

ある delinquent ロードが RIB を起動した後で，他の delinquent ロード（まだ TP 探索されていないもの）がフェッチされた場合，RIB は既に稼働中である．TP 探索は，RIB を起動させた delinquent ロードが行わなければならないので，後者の delinquent ロードは RIB を起動することができない．そのような場合には TP 探索は行わず，次回の実行以降に延期される．

TP フラグおよび TG フラグは，MCT リセットの際に，全てクリアする．これは，プログラムの実行のフェーズ移行に追従するためである．追従しなければ，TP 命令集合が不必要に大きくなり，不要な先行実行が生じる．また，これらのフラグは，命令キャッシュに保持するため，キャッシュが書き換えられる際に消去されるが，命令キャッシュ・ミスが起こることは稀であり，消去された後に再度必要となる可能性は低いものと考えられる．

4.3 選択的先行実行の VROB 方式への適応

VROB 方式では ROB が不足していた場合，ROB 及び物理レジスタを割り当てないまま，命令を先行ディスパッチする．この先行ディスパッチされる命令を，TP 探索によって TP フラグがセットされた命令のみに限定することで，選択的先行実行を VROB 方式に適応する．ただし，TP フラグがセットされておらず，先行ディスパッチがスキップされた命令についても，後に RFQ から再フェッチされ，本実行される必要がある．

RIB を起動するタイミングは，delinquent ロードを本実行ディスパッチするタイミングとする．ここで本実行ディスパッチとは，命令に ROB や物理レジスタなどを割り当てた後，本実行を行うためにディスパッチすることを表す．この場合，TP 探索を行う命令数は，最大でも実 ROB のエントリ数と同数の命令のみとなる．

表 1　キャッシュの MPKI 及びメモリ・アクセス率
Table 1　Statistics of cache and memory accesses.

program	MPKI		memory	memory
	L1 data	L2	access rate	intensive?
ammp	21.8	0.7	0.3	no
applu	24.0	17.0	6.1	yes
apsi	4.0	1.0	0.4	no
art	108.3	10.1	5.1	yes
equake	70.1	26.4	6.4	yes
facerec	3.8	1.6	0.7	no
fma3d	15.6	12.1	4.3	yes
galgel	19.0	1.1	0.3	no
lucas	26.3	21.9	9.4	yes
mesa	1.3	0.6	0.2	no
mgrid	19.4	6.6	2.1	moderately
sixtrack	0.9	0.3	0.2	no
swim	39.7	20.1	9.4	yes
wupwise	4.6	2.3	1.2	moderately

表 2　ベース・プロセッサの構成
Table 2　Configration of base processor.

Pipeline width	4-instruction wide for each of fetch, decode, issue, and commit
Real ROB	128 entries
Fetch queue	16 entries
Issue queue	128 entries
LSQ	128 entries
Physical regsiter	128 for int and fp
Function unit	4 iALU, 2 iMULT/DIV, 2 Ld/St, 4 fpALU, 2 fpMULT/DIV/SQRT
L1 I-cache	64KB, 2-way, 32B line
L1 D-cache	64KB, 2-way, 32B line, 2 ports, 2-cycle hit latency, non-blocking
L2 cache	2MB, 4-way, 64B line, 12-cycle hit latency
Main memory	300-cycle min. latency, 2B/cycle bandwidth
Branch prediction	16-bit history gshare, 64K-entry PHT, 2048-entry 4-way set associative BTB, 10-cycle misprediction penalty

5. 評　　価

5.1 評価環境

評価には，SimpleScalar Tool Set Version 3.0a[10] をベースに提案手法を実装したシミュレータを用いた．命令セットには Compaq Alpha ISA を用いた．ベンチマーク・プログラムとして，数値計算プログラムからなる SPECfp2000 を使用した．**表 1** に使用したベンチマーク・プログラム及びロードの L1 データ・キャッシュ・ミス率（MPKI: misses per kilo-instructions），L2 キャッシュ・ミス率，主記憶アクセス率を示す．バイナリは，Compaq C 及び Fortran コンパイラを用いて-fast -O4 のオプションでコンパイルした．入力には ref 入力を用い，SimPoint[5] によって選択した 100M 命令を実行した．評価におけるベース・プロセッサの構成を表**表 2** に示す．

評価では以下の 2 つモデルについて評価を行った．

- **full**: $M = 8$ に仮想 ROB を拡大した VROB 方式において，可能な限りすべての命令の先行実行を行うモデル
- **low-power**: $M = 8$ に仮想 ROB を拡大した VROB 方式において，選択的先行実行を行うモデル

full 及び low-power モデル固有のプロセッサ構成を**表 3** に示す．本論文では，先行実行用の FIFO 発行キューを本実行用発行キューに統合し，1 つの CAM の発行キューとした．発行キュー及び LSQ は十分に存在すると仮定し，エントリ数を仮想 ROB エントリ数と同一とした．また，この 2 つのモデルにおいては 8 エントリの FB を用いた．FB の容量を有効に利用するため，エントリの置換ポリシとして non-bypass caching[4] を用いた．このポリシでは，バイパス経由で読み出されなかった結果のみを FB に置く．2 回以上参照されるオペランドは少ないため，読み出されることのないオペランドによる FB の容量の浪費を抑制できる．

5.2 先行実行命令削減率

図 7 に，コミット命令数に対する先行ディスパッチ命令数の割合を示す．各棒グラフは 2 つの部分にわかれている．「先行実行」は実際に実行された命令であり，「削除」は先行ディスパッチされたが，発行キュー内で削除された命令の割合を表している．「先行実行」と「削除」を合わせたものが，コミット命令数に対する先行ディスパッチ命令の割合となる．また図 7 の右端の「A.M.」は，14 本の算術平均である．

先行実行による消費電力は，先行ディスパッチされた命令数と先行実行された命令数に

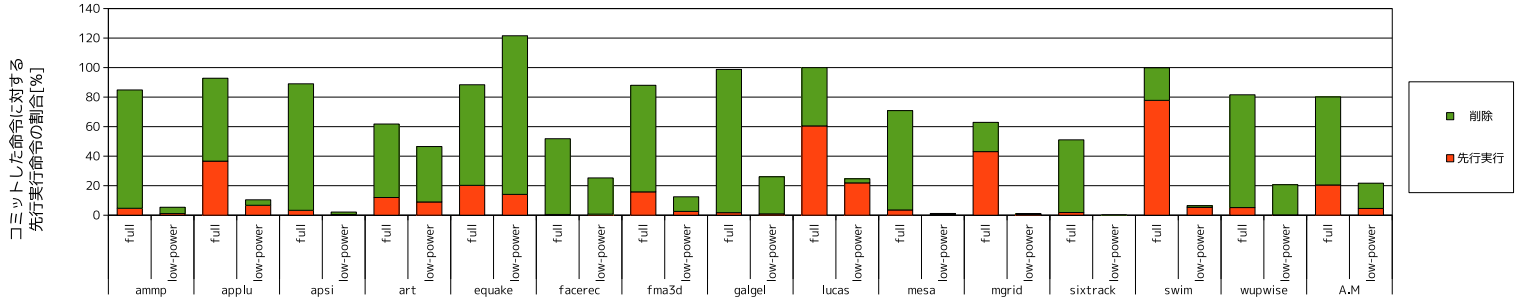

図 7 コミット命令に対する先行ディスパッチ命令の割合
Fig. 7 Ratio of pre-dispatched instructions to commited instructions.

表 3 full, low-power モデルの構成
Table 3 Configrations of full and low-power models.

common	1024-entry LSQ 1024-entry issue queue 8-entry forwarding buffer, non-bypass caching
low-power model	1024-entry MCT 128-entry RIB Threshold of miss count to identify load is 8. Reset interval of MCT and TP, TG flags is 1M cycles. TP search penalty is 1 cycle per each RIB entry.

よって変化する．先行ディスパッチされた命令は，発行キューの消費電力を増加させる．また先行実行された命令は，機能ユニットやキャッシュの消費電力を増加させる．図からわかるように，選択的先行実行を行うことで，ほとんどのプログラムにおいて，先行ディスパッチ命令数，先行実行命令数を大きく削減することができている．

コミット命令数に対する先行ディスパッチ命令数の割合は full モデルでは平均 81%であったが，low-power モデルでは平均 22%と大きく削減できた．またコミット命令数に対する先行実行命令の割合も full モデルでは平均 20%であったが，low-power モデルでは平均 5%と大きく削減できた．また，先行ディスパッチ命令と先行実行命令の削減率の平均は，それぞれ，72%，77%である．

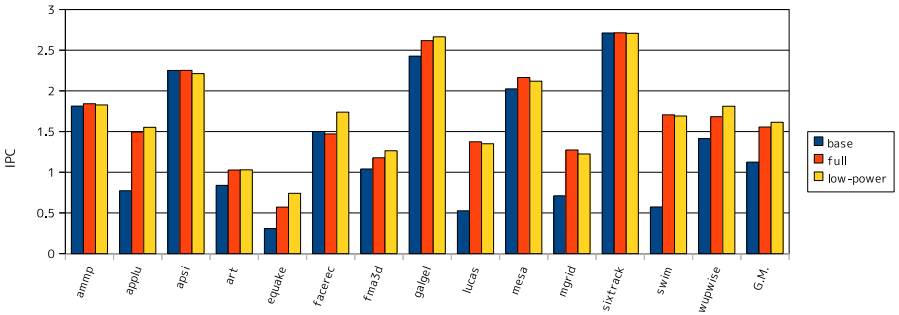

図 8 IPC
Fig. 8 IPC.

5.3 性能評価

図 8 に base モデル，full モデル，low-power モデルの IPC を示す．また，**図 9** に，コミットされたロード命令のレイテンシを示す．

low-power モデルでは，先行ディスパッチを delinquent ロード，TP 命令に限定しているため，full モデルよりもプリフェッチ機会が減少し，一般にレイテンシは若干増加すると予想された．そのため性能も低下すると予想された．しかし，実際には low-power モデルのレイテンシは full モデルとほぼ同等か，削減されており，平均では，21%削減されている．

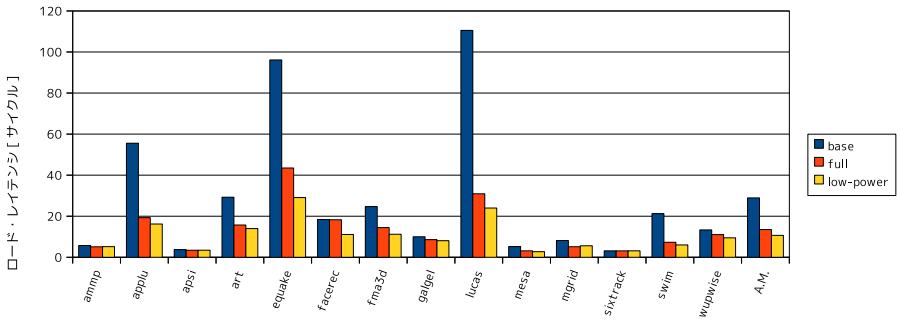

図 9 ロード・レイテンシ
Fig. 9 Load latency.

この結果，性能もすべてのプログラムで同等か上回るようになり，平均で 3%向上している．理由として次の 2 点があげられる: 1) delinquent ロードがほぼすべての L2 ミスをカバーしていると思われる．2) 先行実行する命令を削減したことによって，実行における資源競合が減少し，発行キューでの待ち合わせ時間が減少し，その結果，プリフェッチのタイミングが改善されたと思われる．性能が full モデルとほぼ同等で，図 7 に示す通り先行実行命令が削減できたので，本論文提案の手法は効果的であると言える．

6. ま と め

データ・プリフェッチを実現する方法に命令の先行実行がある．単一スレッド環境において先行実行を行う手法として，我々はこれまでに仮想 ROB（VROB : virtual reorder buffer）方式を提案した．しかし従来の VROB 方式では，先行実行が有効でない命令も先行実行し，無駄な電力を消費していた．そこで本論文では，頻繁にキャッシュ・ミスを起こすロード命令とそれが直接あるいは間接的に依存する命令のみを先行実行することで，有効な先行実行のみを行う選択的先行実行方式を VROB 方式に適用し，評価を行った．SPECfp2000 ベンチマークを用いて評価を行った結果，通常の VROB プロセッサをやや上回る性能で，先行ディスパッチ命令数を 72%削減できることを確認した．

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 22500045）による補助のもとで行われた．

参 考 文 献

1) Borch, E., Manne, S., Emer, J. and Tune, E.: Loose Loops Sink Chips, *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, pp.299–310 (2002).
2) Collins, J.D., Tullsen, D.M., Wang, H. and Shen, J.P.: Dynamic Speculative Precomputation, *Proceedings of the 34th International Symposium on Microarchitecture*, pp.306–317 (2001).
3) Collins, J.D., Wang, H., Tullsen, D.M., Hughes, C., Lee, Y.-F., Lavery, D. and Shen, J. P.: Speculative Precomputation: Long-range Prefetching of Delinquent Loads, *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pp.14–25 (2001).
4) Cruz, J.-L., González, A., Valero, M. and Topham, N.P.: Multiple-Banked Register File Architectures, *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp.316–325 (2000).
5) Hamerly, G., Perelman, E., Lau, J. and Calder, B.: SimPoint 3.0: Faster and More Flexible Program Phase Analysis, *Journal of Instruction-Level Parallelism*, Vol.7, pp.1–28 (2005).
6) Hyodo, K., Iwamoto, K. and Ando, H.: Energy-Efficient Pre-Execution Techniques in Two-Step Physical Register Deallocation, *IEICE Transactions on Information and Systems*, Vol.E92-D, No.11, pp.2186–2195 (2009).
7) Mutlu, O., Kim, H. and Patt, Y.N.: Techniques for Efficient Processing in Runahead Execution Engines, *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pp.370–381 (2005).
8) Mutlu, O., Stark, J., Wilkerson, C. and Patt, Y.N.: Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors, *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, pp.129–140 (2003).
9) Roth, A. and Sohi, G.S.: Speculative Data-Driven Multithreading, *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, pp. 37–48 (2001).
10) simplescalar: `http://www.simplescalar.com/`.
11) 市原敬吾，田中雄介，安藤秀樹：仮想化により拡大したリオーダ・バッファによる先行実行，2011 年先進的計算基盤システムシンポジウム SACSIS 2011，pp.64–71 (2011).