Vol.2015-ARC-216 No.16
2015/8/5

ランダムバッファの発行キューにより生じる性能低下の抑制

酒井 信二[1]　塩谷 亮太[1]　安藤 秀樹[1]

概要：
発行キューはプロセッサの構成要素の中でも性能に大きな影響を与えるものの 1 つである．発行キューの代表的な構成方法として，シフトキューとサーキュラキューの 2 つがある．2 つのキューは共通して命令をプログラム順に保持する．しかし，シフトキューには電力消費が大きい，サーキュラキューには容量の無駄があるという欠点がある．これらとは違い，プログラム順に命令を格納しないランダムキューがある．ランダムキューは空きエントリに命令を挿入していくので，キューの容量は無駄にならない．しかし，命令がプログラム順に並ばないため，古い命令に高い優先度を与えることができず，プロセッサの性能が低下する．ランダムキューによる性能低下を解決するために，ランダムキューをメインキュー（MQ:main queue）とオールドキュー（OQ:old queue）と呼ぶ 2 つのキューに分割して発行を行う手法を提案する．フロントエンドからは，MQ にのみ命令を挿入し，OQ へは MQ の最も古い数命令を移動させる．2 つのキューで選択論理を共有し，OQ の命令を MQ の命令よりも優先して発行することで，データフローのクリティカルパス上にある命令に高い優先度を与え，プロセッサの性能低下を抑制する．SPECCPU2006 を用い評価した結果，ランダムキューの IPC は，シフトキューに対して幾何平均で 8.47%，最大で 21.57%低下していたが，提案手法を適用することでそれぞれ 1.41%，4.01%まで IPC 低下を削減できた．

1. はじめに

プロセッサの性能に大きな影響を与えるものの 1 つとして発行キュー（IQ:issue queue) が挙げられる．アウトオブオーダ実行では IQ 内の命令の順序にかかわらず，レディになった命令を発行する．一般に，レディな命令のうち，より古い命令を優先的に発行すれば性能はより高くなる．IQ 内で命令をプログラム順に並べておけば，レディな命令から実際に発行する命令を選ぶ選択論理としては，単純な回路で，古い命令に高い優先度を与えるよう実現できる [1][9][12].

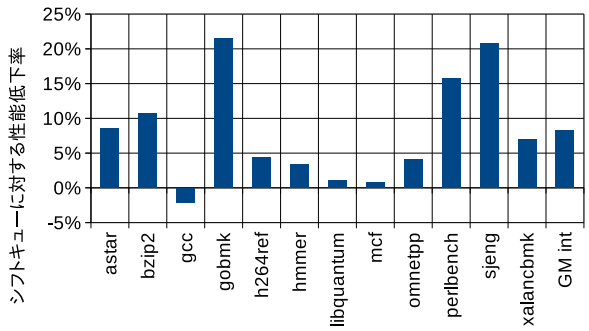
そのような IQ としては主に，シフトキューとサーキュラキューがある．シフトキューは，命令を発行したエントリの空きを詰めるコンパクションを行うことで，高い容量効率を達成できるが，コンパクションには大きな電力を要し，回路が複雑になる．一方，サーキュラキューはコンパクションをしないので電力消費が小さくなる．しかし，空いたエントリの分だけ実効的な容量が低下する欠点がある．

コンパクションをせず，実効容量を低下させない第 3 の方式としてランダムキューがある．ランダムキューは単に空いているエントリに新しい命令を挿入する．具体的には，空いているエントリの番号を保持するフリーリストを用意し，命令を挿入するときはそこからエントリ番号を得て，そのエントリに挿入する．ランダムキューには容量を無駄にすることがないという特徴がある．このためランダムキューは，例えば，Alpha 21464[8], AMD Bulldozer[13], IBM POWER8[14] といったプロセッサに用いられている．しかし，命令がプログラム順に並んでいないので誤った優先度を選択論理に与えることになる．これにより，レディな命令数が発行幅や機能ユニットの数よりも多く，実際に発行する命令を選択しなければならない状況で，より重要でない命令を選択することがある．このようにして，誤った優先度はプロセッサの性能を低下させる．

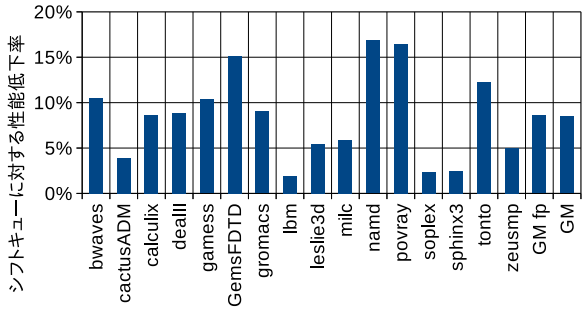
図 1 に，シフトキューとランダムキューの性能比較結果を示す．プロセッサの構成は 4 節の表 1 に示す．(a) が SPECCPU2006int，(b) が SPECCPU2006fp での評価を示す．横軸は各ベンチマーク，縦軸はシフトキューに対するランダムキューの IPC 低下率を表している．

ランダムキューの幾何平均における IPC 低下率は 8.47%であり，大きく低下している．IPC 低下率の最大値は gobmk での 21.57%と非常に大きい．また，ベンチマーク毎の IPC 低下率のばらつきが激しい．本論文では

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



(a) SPECCPU2006int



(b) SPECCPU2006fp

図 1 ランダムキューのベンチマーク毎の性能

ランダムキューに着目し，IPC 低下を抑制することを目的とする．

本論文では，ランダムキューをメインキュー（MQ:main queue）とオールドキュー（OQ:old queue）と呼ぶ 2 つのキューに分割する**再配置ランダムキュー (RRQ:rearranging random queue)** を提案する．MQ と OQ はランダムキューであり，どちらからでも命令を発行できる．一方で，命令ディスパッチは MQ にのみ行い，OQ へは MQ 内の最も古い数命令を移動させる．MQ と OQ で選択論理を共有し，OQ の命令を MQ の命令よりも優先して発行する．一般に古い命令がクリティカルパスに含まれ，プロセッサの性能に影響を与える確率が高い．これを利用し，古い命令が入っている OQ から優先的に発行することで，性能低下は小さくなる．

以下本論文の構成について述べる．2 節では発行論理について説明する．3 節で提案手法について述べ，4 節で性能評価を行う．5 節で関連研究を挙げ，最後に 6 節で本論文をまとめる．

2. 発行論理の構成と動作

本節では提案手法の前提となる発行論理の構成と動作について説明する．発行論理は，リネームされた命令を保持し，発行すべき命令を決定する．図 2 に示すように，発行論理はウェイクアップ論理，選択論理，タグ RAM，ペイロード RAM で構成される．

以下発行論理の動作について述べる．ウェイクアップ論理の各エントリは，対応する命令の 2 つのソースオペランドに対応するタグと，それらの状態を表すレディフラグを保持する．各ソースオペランドが利用できる状態ならば，対応するレディフラグがセットされる．2 つのフラグがセットされ，依存が解決したならば，発行要求信号を選択論理へ送る．選択論理は，資源制約などを考慮して，要求された命令の中から発行を許可する命令を選択し，発行許可信号を出力する．発行許可信号はペイロード RAM に送られ，機能ユニットに発行する命令の情報を送信する．発行許可信号はタグ RAM にも出力される．タグ RAM は各命令のデスティネーションタグを保持する．発行許可された命令のタグを読み出し，それをウェイクアップ論理へ放送して，レディフラグを更新する．

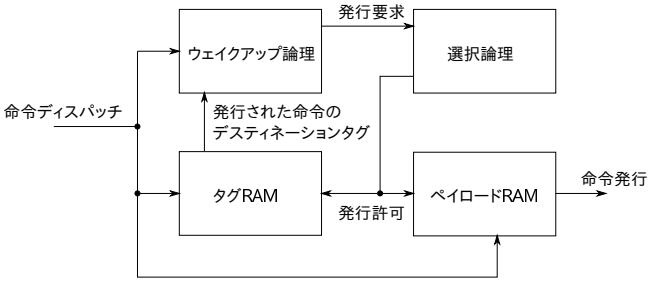


図 2 発行論理の概略図

IQ がランダムキューの場合は，これらとは別にフリーリストを用意する．フリーリストは IQ の空きエントリの番号を保存するバッファであり，FIFO で構成される．命令をディスパッチするときは，フリーリストから空きエントリの番号を得て，そのエントリに命令を挿入する．一方，命令を発行したら，自身のエントリ番号をフリーリストに返す．

3. 再配置ランダムキュー

本節では提案手法である再配置ランダムキューについて述べる．3.1 節では提案手法の概要について述べる．3.2 節では提案手法の発行論理の構成について述べる．3.3 節ではフロントエンドから命令をディスパッチされるメインキューについて述べる．3.4 節では古い命令を保持するオールドキューについて述べる．3.5 節ではメインキュー内の命令のプログラム順を管理するプログラムオーダーキューについて述べる．3.6 節では命令をメインキューからオールドキューへ移動させる際に発生しうる問題とその対処法について述べる．

3.1 概要

データフローのクリティカルパス上に命令があるならば，後続の多くの命令が直接，あるいは間接的にこれに依存している．このため，このような命令は発行キュー内に長時間残っている古い命令となる確率が高い．すなわち，古い命令はデータフローのクリティカルパス上にあると言

えるので，性能への影響が大きく，優先的に発行する必要がある．逆に新しい命令は，依存する命令の数が少ないので，発行の順番がプログラム順でなくとも，性能はあまり低下しないと言える．以上より，古い命令さえ優先的に発行できれば，新しい命令の発行優先度がランダムでも，プロセッサの性能は大きく低下しないと考えられる．

そこで, ランダムキューをメインキュー（MQ:main queue）とオールドキュー（OQ:old queue）に分割する再配置ランダムキュー (RRQ:rearranging random queue) を提案する．MQ と OQ はどちらもランダムキューであり，どちらからでも命令を発行できる．一方で，命令のディスパッチは MQ にのみ行い，OQ へは MQ の最も古い数命令を移動させる．選択論理は MQ と OQ とで共有し，OQ の命令を MQ の命令よりも優先的に発行する．以上のような構成により，通常のランダムキューと比較して，元々のプログラム順において一定数の古い命令に対して高い優先度を与えるので，プロセッサの性能低下を抑制できる．

3.2 再配置ランダムキューの構成

RRQ を実装した発行論理全体の概略図を図 3 に示す．命令の情報を保持するウェイクアップ論理，タグ RAM，ペイロード RAM は MQ と OQ に分割される．図中の PQ はプログラムオーダーキュー（program order queue）である．これは MQ 内の命令のプログラム順を記憶しており，OQ に移動する MQ 内の命令を決定する．MQ，OQ 共にランダムキューなので，命令を挿入するエントリを決定するためのフリーリストをそれぞれ個別に備えている．

MQ に命令をディスパッチするときは，フリーリストから空きエントリの番号を取得し，そのエントリに命令を挿入する．同時に PQ の末尾にそのエントリ番号を挿入する．

MQ の古い命令を OQ へ移動させるときは，移動させる命令のエントリ番号を PQ の先頭から読み出す．OQ は自身のフリーリストから空きエントリの番号を得て，該当するエントリに MQ から読みだした命令を書き込む．

MQ と OQ は通常の IQ と同じように命令の発行要求を選択論理に出し，発行許可を受ける．選択論理は両方のキューで共有している．選択論理は，図 3 における下の命令ほど高い優先度で選択するよう構成し，OQ 内の命令からの要求を MQ 内の命令のそれよりも優先する．この結果，通常のランダムキューと比べて，クリティカルパス上にある確率が高い命令に高い優先度を与えることができる．

3.3 メインキュー

MQ の最も古い数命令を OQ へ命令を移動させるためには，移動する命令のエントリ番号を PQ から受け取り，MQ の情報を OQ へ送る必要がある．そのために，MQ には発行幅と OQ のサイズのどちらか小さい方の数だけ読み出しポートを追加する．

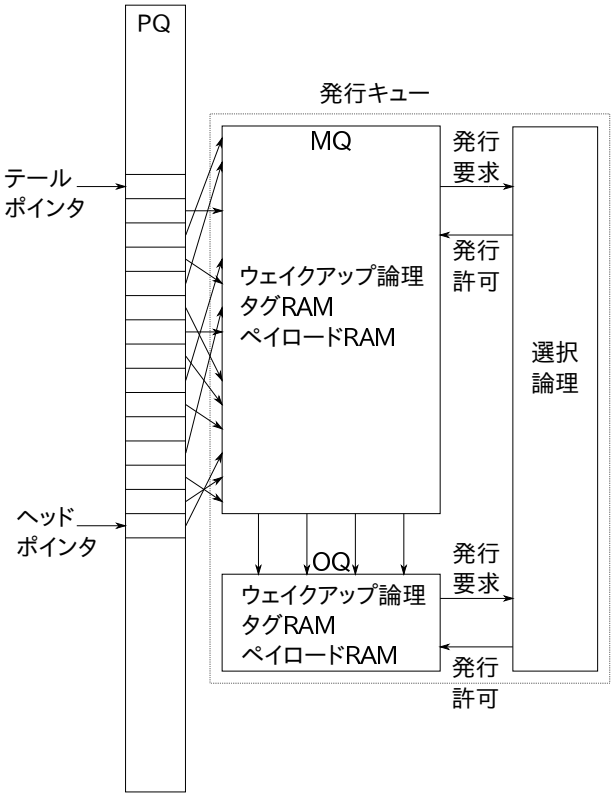


図 3 RRQ を実装した発行論理の概略図

3.4 オールドキュー

OQ には，MQ から移動する命令を書き込むため，MQ に追加した読み出しポート数と等しい数の書き込みポートが必要である．一方で，OQ には命令はディスパッチされないので，このためのポートは不要である．

3.5 プログラムオーダーキュー

PQ は MQ 内の命令のプログラム順を保存するキューである．PQ はサーキュラバッファであり，ヘッドポインタとテールポインタでデータの先頭と末尾を管理する．PQ の各エントリは MQ のエントリを指すポインタを持ち，これをプログラム順に保持する．

PQ の動作について述べる．MQ へ命令をディスパッチすると同時に，ディスパッチしたエントリ番号を PQ の末尾に挿入する．命令移動時の PQ の動作のタイミングを図 4 に示す．まず最初のサイクルで，PQ の先頭から MQ へのポインタを読み出す．次のサイクルでは，MQ の命令を読み出し，OQ へ書き込み，MQ の命令を無効化する．以上のように，PQ 読み出しから，OQ への移動までに合計 2 サイクルを要する．

PQ のサイズについて述べる．MQ は空きエントリのいずれかに命令を挿入できるが，PQ は，命令が挿入された MQ のエントリ番号を末尾にのみ挿入できる．このように，PQ は容量効率が悪いため，MQ より十分大きい必要がある．PQ はプロセッサ中の一部の命令のプログラム順を保持するので，PQ のサイズはインフライト命令数，つ

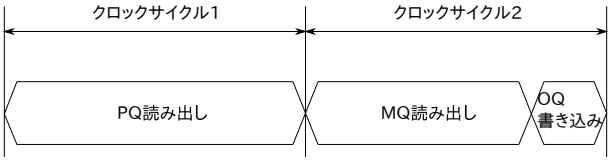


図 4　MQ から OQ へ命令移動

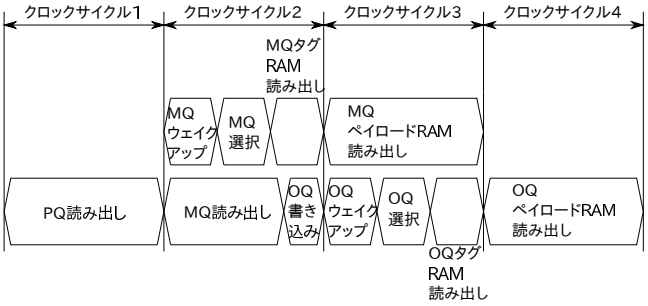


図 5　二重発行が発生する場合のタイミング図

まりリオーダバッファ（ROB:reorder buffer）のサイズだけあれば十分である．PQ のサイズをそれ未満にする場合は，エントリ不足でストールし，性能は低下する．

3.6　二重発行抑止

MQ から OQ への命令移動においては問題が発生する可能性がある．MQ のあるエントリにおいて，命令移動と発行許可が同一サイクルで起きると，MQ から命令が発行された後に，OQ から同一の命令が再び発行される二重発行という問題が生じる．連続したサイクルで同一命令が発行される場合のタイミングを図 5 に示す．図の 1 段目，2 段目はそれぞれ，MQ での命令発行，OQ に関連する動作の各タイミングを表す．ウェイクアップ，選択，タグ RAM 読み出しには，いずれも約 3 分の 1 サイクル時間を要する [12] ので，発行論理のタイミングは図のように描ける．

二重発行によって，ROB やロードストアキュー（LSQ:load store queue），レジスタなどに矛盾が生じ，多くの問題が引き起こされるため，二重発行が発生しない構成にする必要がある．以下，その方法として発行要求抑止，発行許可抑止，移動した命令の無効化の 3 方式を提案する．

3.6.1　発行要求抑止

エントリに有効ビットを設け，ウェイクアップ論理からの発行要求信号をそれによりゲーティングすることで二重発行を防ぐ．これを実現する回路を図 6 に示す．あるエントリに命令がディスパッチにより書き込まれたら有効ビットをセットする．一方，命令移動のための読み出しワード線がアサートされたら，対応するエントリの有効ビットをクリアする．これにより，有効ビットによって，選択論理へ送られる発行要求信号の出力は抑止され，二重発行を防ぐことができる．この時のタイミングを図 7 に示す．発行要求抑止により図の赤いハッチの部分の動作が行われない．その結果，ウェイクアップは行われるが，選択以降の

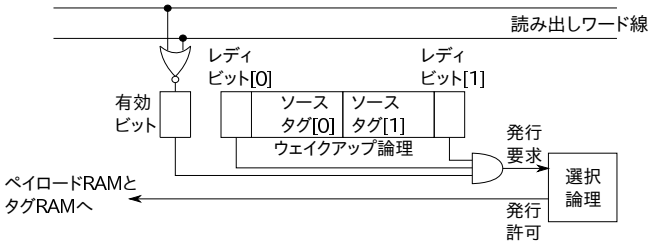


図 6　発行要求信号をゲーティングする回路

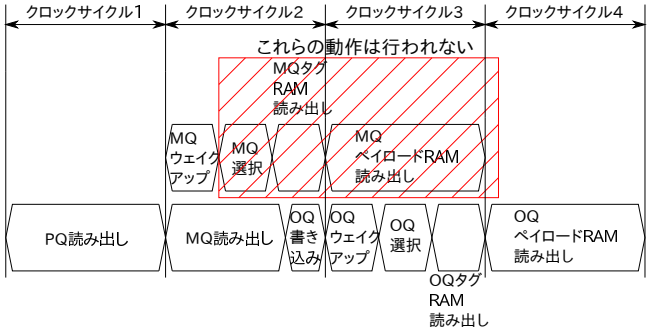


図 7　発行要求抑止のタイミング図

動作は行われず，MQ からの発行を抑止できる．

この方式には，有効ビットの無効化は，レディビットがセットされるウェイクアップの終了時点までに行われなければならない，というタイミング制約がある．

発行抑止された命令は，後に OQ から発行される．このタイミングは，MQ 内の命令が移動せずに発行されたサイクルより，少なくとも 1 サイクル遅れる．これによりプロセッサの性能は低下する．

3.6.2　発行許可抑止

前節で述べたように，命令のウェイクアップが終了するまでに，移動元のエントリが無効化されなければ，発行要求抑止を採用することができない．このタイミングが満足されない場合であって，選択が終了するまでに，移動元のエントリが無効化できるのであるならば，発行許可を抑止することにより二重発行を抑止できる．この方式では，発行要求信号の代わりに発行許可信号をゲーティングする．

これを実現する回路を図 8 に示す．エントリの有効ビットにより，選択論理から各エントリに送られてくる発行許可信号をゲーティングする．タイミングを図 9 に示す．発行許可抑止により図の赤いハッチの部分の動作が無効になる．その結果，ウェイクアップ，選択は行われるが，タグ RAM とペイロード RAM の読み出しが行われず，MQ からの発行を抑止できる．

この方式は発行要求抑止よりもゆるいタイミングで動作可能であるが，選択論理の動作は終了したあとなので，発行幅を無駄に消費してしまい，性能が低下する．また，発行要求信号を抑止する場合と同様に，MQ からの命令発行を無効化するので，命令発行が少なくとも 1 サイクル遅れ，さらに性能が低下するという欠点がある．

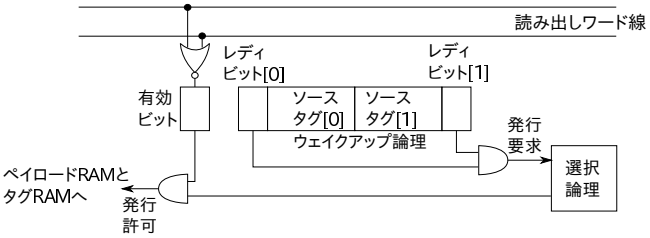


図 8　発行許可信号をゲーティングする回路

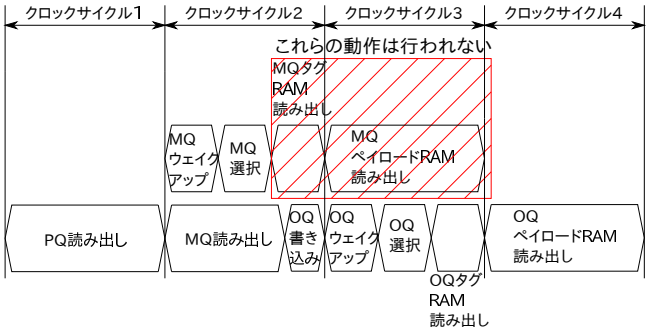


図 9　発行許可抑止のタイミング図

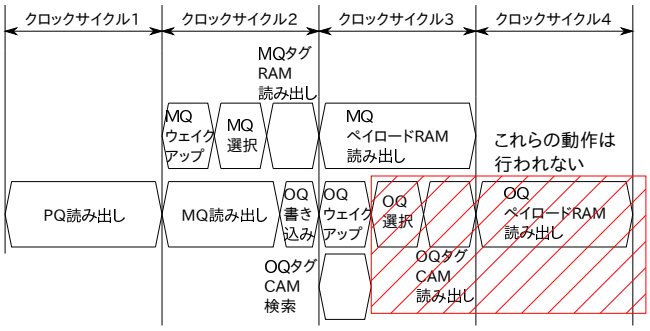


図 10　移動した命令を無効化する場合のタイミング図

3.6.3　移動した命令の無効化

MQ の移動元のエントリの無効化が，発行許可信号の出力より遅い場合は，MQ からの発行を無効にできない．そこで，二重発行を防ぐ方法として，OQ に移動した命令を無効化する方法を提案する．このタイミングを図 10 に示す．

MQ から発行する命令と同一の命令を OQ 内から探すために，その命令のデスティネーションタグで OQ を連想検索する．連想検索を可能にするために OQ のタグ RAM を CAM で構成する．連想検索の結果一致するエントリがあれば，そのエントリを無効化する．後述するように，タイミング的には，ウェイクアップ論理から発行要求信号が出力される前に無効化することは可能である．このため，図 10 の赤いハッチの部分の動作が行われず，OQ からの発行を抑止できる．

しかし，CAM を用いてデスティネーションタグで連想検索する場合，アドレス計算命令や分岐命令などのデスティネーションタグを持たない命令を区別できず，連想検索できない．その対策として，従来のプロセッサでは使用しないデスティネーションタグの値を使い，全ての命令を区別できるようにすることで連想検索を可能にする．

図 10 に示すタイミングの妥当性について述べる．ウェイクアップに要する時間の内，タグ線駆動に費やされる時間は長く，文献 [1] の測定では，64 エントリの IQ で 20%を占める．後に 4.1 節で述べるように，OQ は IQ より非常に小さい 4 エントリなので，タグ線駆動に要する時間は非常に小さい．よって，ウェイクアップと並行して OQ のタグ比較を行えば，発行要求信号が出力される前に無効化する OQ の命令を検出することができるので，図 10 のタイミングで命令の無効化が可能である．

以上のような機構を用いて，同一命令をデスティネーションタグで連想検索し無効化することで，OQ へ移動する命令を持つ MQ のエントリの無効化がどれだけ遅れたとしても，二重発行を防ぐことが可能である．また，発行要求抑止や発行許可抑止と違い，MQ ではなく OQ 内の命令を無効化するので，命令発行の遅れによる性能低下が生じない．

しかし，この実装の場合，OQ のタグ RAM を CAM で構築する必要がある．また，プロセッサの構成によっては，タグのビット幅を 1 ビット増やす必要がある．これらにより回路が複雑になってしまう．

4.　評価

SimpleScalar Tool Set Version 3.0a[16] をベースに RRQ を実装したシミュレータを作成し，性能を評価した．命令セットは Alpha ISA である．ベンチマークプログラムには，現在のところシミュレータが正しく動作しない wrf を除いた SPECCPU2006 の 28 本を採用した．プログラムは gcc ver.4.5.3 でオプション-O3 を用いコンパイルした．プログラムへの入力には ref データ・セットを用いた．各ベンチマークの特徴を反映するために，SimPoint[10] を用いて適切な 100M 命令区間を求め，その範囲でシミュレーションを行った．

プロセッサの構成を表 1 に示す．1 節で述べたように，命令がランダムに並ぶことによる性能低下は，レディな命令数が機能ユニットの数を上回った場合に生じる．このため，機能ユニットの数は評価において重要なパラメータとなる．本評価においては，機能ユニットの数を，最新のスマートフォンで使用される Cortex-A72[15] に準拠させた．

4.1　オールドキューのサイズを変えた場合の評価

以下の 5 つのモデルを評価した．

- **Shift**:IQ がシフトキューで構成されたモデル．
- **Random**:IQ が従来のランダムキューで構成されたモデル．
- **Request**:IQ が RRQ で構成され，発行要求抑止により二重発行を抑止するモデル．
- **Grant**:IQ が RRQ で構成され，発行許可抑止により

Pipeline width	4-instruction wide for each of fetch, decode, issue, and commit
ROB	128 entries
IQ	64 entries
LSQ	64 entries
PQ	128 entries
Physical Registers	128(int) + 128(fp)
Branch prediction	16-bit history 4K-entry PHT gshare, 2K-set 4-way BTB, 10-cycle misprediction penalty
Function unit	2 iALU, 1 iMULT/DIV, 2 Ld/St, 2 fpALU/MULT/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, 8B/cycle bandwidth
Data prefetcher	stride-based, 4K-entry, 4-way table, 16-data prefetch to L2 cache on miss

表 1 プロセッサの基本構成

二重発行を抑止するモデル．

- **OQ_Invalidate**:IQ が RRQ で構成され，移動した命令を無効化することにより二重発行を抑止するモデル．

この評価においては，PQ のサイズは十分に大きなもの（ROB と同サイズ）としている．PQ のサイズが性能に与える影響については，4.2 節で評価する．

Request，Grant，OQ_Invalidate の各モデルにおいて，OQ のサイズを変えながら各ベンチマークでシミュレーションを行い，IPC を求めた．得られた IPC を Shift モデルのものと比べた結果を図 11 に示す．横軸はモデル，縦軸は Shift に対する IPC 低下率を示している．値が小さいほど性能が良いことを表している．箱ひげグラフの縦線の上下の末端はそれぞれ IPC 低下率の最大値と最小値を表す．箱の上端と下端はそれぞれ第 3 四分位点と第 1 四分位点を表す．緑の十字は IPC の幾何平均における低下率を表す．各モデルにおける 6 つのグラフは異なる OQ のサイズでの測定値である．MQ と OQ のサイズの和は，常に表 1 の IQ のサイズとなっている．OQ のサイズが 0 の時は，OQ が存在しない，つまり従来のランダムキューである Random に等価である．ゆえにどのモデルも同じ IPC を示す．

RRQ を実装することで，IPC 低下率の幾何平均値が減少し，いずれのモデルも OQ のサイズが 4 の時に最小値を得る．Random では Shift に対して IPC が 8.47%低下したが，Request では 1.77%，Grant では 2.42%，OQ_Invalidate では 1.41%まで IPC 低下を抑制できた．

OQ のサイズが 0〜4 の場合は，サイズが大きくなるほど

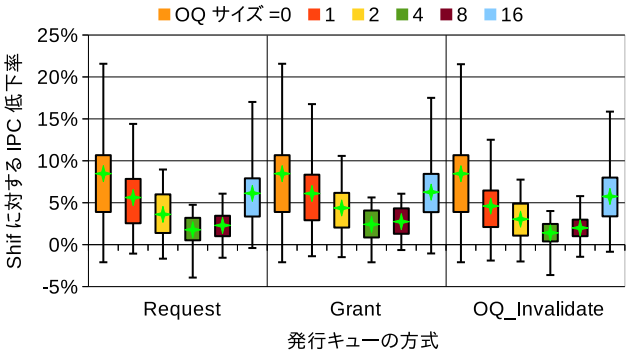


図 11 RRQ を実装したモデルの IPC 低下率

グラフの箱の高さが小さくなるので，IPC 低下率のばらつきが小さくなることが分かる．また，IPC 低下率の最大値も小さくなっているので，結果が最も悪いベンチマークにおいても性能向上していることが分かる．以上から OQ のサイズが 4 以下の場合は全体の性能が向上し，提案手法は有効と分かる．

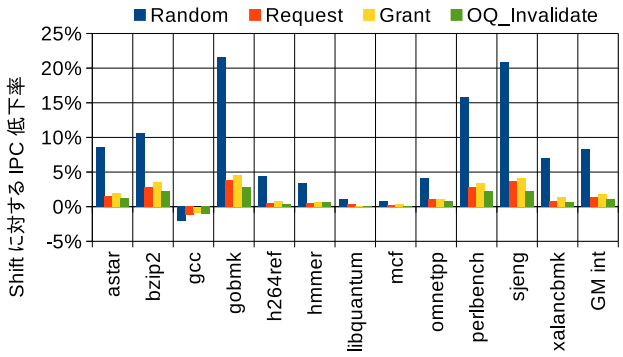
各モデルを比較すると，OQ_Invalidate，Request，Grant の順に性能が良い．OQ_Invalidate が他の 2 つのモデルよりも優れている理由は，3.6.1 節〜3.6.3 節で述べたように，他の 2 つのモデルには存在する二重発行の抑止による命令発行の遅延が存在しないためである．Grant の性能が最も悪い原因は，命令発行の遅延に加えて，発行幅の無駄な消費があるからである．

これ以降では，IPC の幾何平均値が最も高い，OQ のサイズが 4 の場合のみ評価する．図 12 に各ベンチマークにおける IPC 低下率を示す．(a) は SPECCPU2006int，(b) は SPECCPU2006fp での評価を表している．縦軸は Shift に対する IPC 低下率を表す．横軸は各ベンチマークであり，右端は int，fp それぞれでの幾何平均値を表している．

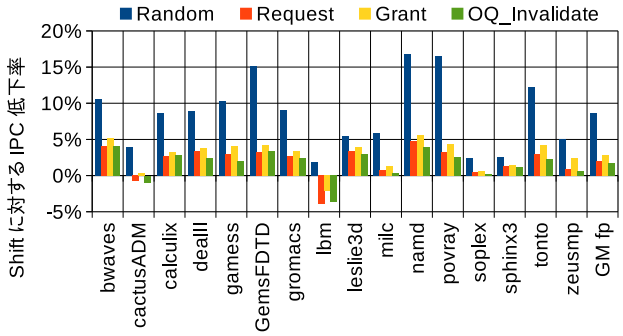
gcc を除く全てのベンチマークにおいて，RRQ を実装することで Random と比べて IPC 低下率が減少している．gcc では逆に増加しているが，Shift よりは優れている．IPC 低下率の最大値は，Request では 4.75%，Grant では 5.63%，OQ_Invalidate では 4.01%であった．これは RRQ を実装する前の 21.57%を大きく下回っている．幾何平均の IPC 低下率で見ても，OQ_Invalidate では Random に対して 7%幅以上減少しているので，提案手法によりランダムキューの性能低下を抑制できたと言える．

4.2 プログラムオーダーキューのサイズを変えた場合の評価

本節では必要な PQ のサイズを求める．IQ のサイズである 64 から ROB のサイズである 128 まで，PQ のサイズを変えながらシミュレーションを行い，IPC を測定した．モデルとしては，OQ のサイズが 4 の OQ_Invalidate を使用した．結果を図 13 に示す．横軸は PQ のサイズ，縦軸



(a) SPECCPU2006int



(b) SPECCPU2006fp

図 12 4 エントリの OQ でのベンチマーク毎の性能

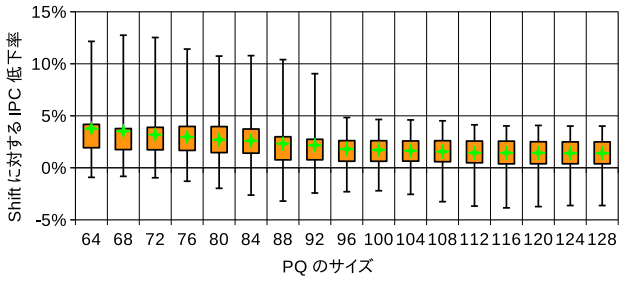


図 13 PQ のサイズを変えた場合の性能

は Shift に対する IPC の低下率を表している．値が小さいものほど性能が良い．グラフは，図 11 と同じく箱ひげグラフで表している．

PQ のサイズを小さくすると徐々に IPC 低下率が増加する．特に 92 以下になると，低下率の最大値が 4.84%から 9.05%まで 2 倍近くに増加する．このことから，提案手法を実装する場合は，PQ のサイズは 96 以上にすることが適切であると言える．96 の時の IPC の幾何平均の低下率は 1.80%である．また，最大値は 4.84%である．128 の時の IPC の幾何平均と最大値の低下率はそれぞれ 1.41%，4.01%である．これらの差は小さいので，PQ のサイズは 96 とするのが妥当である．

5. 関連研究

シフトキュー シフトキューを実装したプロセッサとして Alpha 21264 がある．文献 [2] にその回路の詳細が述べられている．1 節で述べたように，シフトキューは，十分に容量を有効利用できる他，コンパクションにより命令がプログラム順に並んでいるので，選択論理に正しい発行優先度を与えることができるという利点がある．しかし，コンパクションを行うために，複雑な回路が必要になり，電力を大きく消費する．

サーキュラキュー Henry らは，サーキュラキューのラップアラウンドに対応する回路を提案した [3]．キューの最後のエントリと最初のエントリをつなぎ，論理的にループさせる．これにより，キューの先頭と末尾の位置がどの位置にあっても，命令の発行優先度が正しくなる．しかし，クリティカルパス上にキューの全体を横断するような配線が加わるため，選択回路の遅延が非常に大きくなる．

発行論理の最適化 Palacharla らは IQ を複数の FIFO で構成する手法を提案した [1]．命令をディスパッチするときは依存する命令を末尾に保持する FIFO に挿入する．そのような FIFO がなければ空の FIFO に挿入する．空の FIFO もなければ命令ディスパッチを停止する．発行する命令の選択は，FIFO の先頭にある命令のみを対象に行えばよいので，実装が単純になり，複雑さが減る．しかし，FIFO の先頭の命令のソースオペランドの使用可否をチェックするために，多ポートのスコアボードが必要になるという欠点がある．

Stark らは IQ をパイプライン化する手法を提案した [4]．通常 1 サイクルで行われるウェイクアップと選択を 2 ステージにわけて行う．これにより複雑さが減り，大きな IQ の実装が可能になる．しかし，依存する命令を連続して発行するには，データフローグラフにおいて 2 つ前の命令が選択された時点で投機的にウェイクアップする必要があり，投機失敗からの回復を行う複雑な回路が必要となる．

五島らはウェイクアップ論理を CAM ではなく，依存行列と呼ぶ RAM で構成する手法を提案した [5]．依存行列は IQ 内の命令の依存関係を表す．これを用いることで比較器を使わずに依存する命令をウェイクアップできる．しかし，CAM の IQ と同じく，高い容量効率を達成しつつ，正しい発行優先度を命令に与える簡単な方法は存在しない．

Brown らは選択論理を省略した IQ を提案した [6]．ウェイクアップと選択からなるクリティカルループから選択を排除することでループを短くでき，IQ の遅延を短くできる．しかし，この手法では，発行要求した命令は常に発行が許可されるとし，投機発行を行うので，投機失敗からの回復を行う複雑な回路が必要になる．

Michaud らは命令を IQ に挿入する前に，実行レイテンシを予測し，あらかじめスケジューリングすることで，発行キューの複雑さを軽減する手法を提案した [7]．しかし，キャッシュミスが起きると，IQ がミスした命令に依存する命令で満たされ，他の命令が発行できなくなる．また，

レイテンシを正確に予測できなければ，正しいスケジューリングを行えず，命令の発行が遅れる欠点がある．

Sassone らは，行列を用いて選択論理を構成する手法を提案した [11]．各命令がどの命令より古いのかを表す行列を用いることで，最も古い命令を単純な回路で選択できる．行列を用いる選択論理は Alpha 21464[8] と IBM POWER8[14] で使用されているが，最も古い 1 命令しか同定できないので，完全に正しい優先度を与えられない．

6. まとめ

発行キューはプロセッサの構成要素の中でも性能に大きな影響を与えるものの 1 つである．発行キューの実装方法の 1 つとして，プログラム順に命令を格納しないランダムキューがある．これは空きエントリに命令を挿入していくので，命令の並びはランダムとなる．キューの容量効率は良いが，古い命令に高い優先度を与えることができず，プロセッサの性能が低下する．

ランダムキューによる性能低下を解決するために，ランダムキューをメインキュー（MQ:main queue）とオールドキュー（OQ:old queue）と呼ぶ 2 つのキューに分割して発行を行う手法を提案した．フロントエンドからは MQ にのみ命令を挿入するが，OQ へは MQ の最も古い数命令を移動させる．OQ の命令を MQ の命令よりも優先して発行することで，データフローのクリティカルパス上の命令に高い優先度を与え，プロセッサの性能低下を抑制する．

SPECCPU2006 を用い評価した結果，ランダムキューの IPC は，命令がプログラム順に格納される発行キューと比べて，幾何平均で 8.47%，最大で 21.57%低下していたが，提案手法を適用することでそれぞれ 1.41%，4.01%まで IPC 低下を抑制することができた．

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 25330057），及び科学研究費補助金 若手研究 (A)（課題番号 24680005）による補助のもとで行われた．

参考文献

[1] Palacharla, S., Jouppi, N. P. and Smith, J. E.: Complexity-Effective Superscalar Processors, *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pp. 206–218 (1997).

[2] Farrell, J. A. and Fischer, T. C.: Issue Logic for a 600-MHz Out-of-Order Execution Microprocessor, *Journal of Solid-State Circuits*, Vol. 33, No. 5, pp. 707–712 (1998).

[3] Henry, D. S., Kuszmaul, B. C., Loh, G. H. and Sami, R.: Circuits for Wide-Window Superscalar Processors, *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp. 236–247 (2000).

[4] Stark, J., Brown, M. D. and Patt, Y. N.: On Pipelining Dynamic Instruction Scheduling Logic, *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, pp. 57–66 (2000).

[5] Goshima, M., Nishino, K., Kitamura, T., Nakashima, Y., Tomita, S. and Mori, S.: A High-Speed Dynamic Instruction Scheduling Scheme for Superscalar Processors, *Proceedings of the 34th Annual International Symposium on Microarchitecture*, pp. 225–236 (2001).

[6] Brown, M. D., Stark, J. and Patt, Y. N.: Select-Free Instruction Scheduling Logic, *Proceedings of the 34th Annual International Symposium on Microarchitecture*, pp. 204–213 (2001).

[7] Michaud, P. and Seznec, A.: Data-Flow Prescheduling for Large Instruction Windows in Out-of-Order Processors, *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pp. 27–36 (2001).

[8] Preston, R. P., Badeau, R. W., Bailey, D. W., Bell, S. L., Biro, L. L., Bowhill, W. J., Dever, D. E., Felix, S., Gammack, R., Germini, V., Gowan, M. K., Gronowski, P., Jackson, D. B., Mehta, S., Morton, S. V., Pickholtz, J. D., Reilly, M. H., Smith, M. J. : Design of an 8-wide Superscalar RISC Microprocessor with Simultaneous Multithreading, *2002 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pp. 334–472 (2002).

[9] Goshima, M.: Research on High-Speed Instruction Scheduling Logic for Out-of-Order ILP Processor, PhD Thesis, Kyoto University (2004).

[10] Hamerly, G., Perelman, E., Lau, J. and Calder, B.: SimPoint 3.0: Faster and More Flexible Program Analysis, *Journal of Instruction-Level Parallelism*, Vol. 7, pp. 1–28 (2005).

[11] Sassone, P. G., Rupley II, J., Brekelbaum, E., Loh, G. H. and Black, B.: Matrix Scheduler Reloaded, *Proceedings of the 34th Annual International Symposium on Computer Architecture*, pp. 335–346 (2007).

[12] Yamaguchi, K., Kora, Y. and Ando, H.: Evaluation of Issue Queue Delay: Banking Tag RAM and Identifying Correct Critical Path, *Proceedings of the 29th International Conference on Computer Design*, pp. 313–319 (2011).

[13] M. Golden, S. Arekapudi, J. V.: 40-Entry Unified Out-of-Order Scheduler and Integer Execution Unit for the AMD Bulldozer x86-64 Core, *2011 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pp. 80–82 (2011).

[14] Sinharoy, B., Norstrand, J. A. V., Eickemeyer, R. J., Le, H. Q., Leenstra, J., Nguyen, D. Q., Konigsburg, B., Ward, K., Brown, M. D., Moreira, J. E., Levitan, D., Tung, S., Hrusecky, D., Bishop, J. W., Gschwind, M., Boersma, M., Kroener, M., Kaltenbacha, M., Karkhanis, T. and Fernsler, K. M.: IBM POWER8 Processor Core Microarchitecture, *IBM Journal of Research and Development*, pp. 2:1 – 2:21 (2015).

[15] Gwennap, L.: ARM Optimizes Cortex-A72 for Phones, *Microprocessor Report* (2015).

[16] http://www.simplescalar.com/.