Vol.2023-ARC-251 No.7
2023/1/10

整数命令のレイテンシ耐性を利用したALUの消費電力削減手法

黒川 陸[†1,a)]　安藤 秀樹[†1]

概要：現在のプロセッサが消費する電力は，冷却の限界にまで達している．このため，電力を増加させる結果となるクロック速度の向上は制限されており，性能向上が困難になっている．よって，プロセッサの消費電力の削減は非常に重要な課題である． 本研究では，ALU/アドレス生成回路（以下，まとめて ALU と称する）の消費電力を削減することを目的とする．ALU は，コアの消費電力の 10% 程度を占める回路である．ALU では一般に 1 サイクルでの計算完了が求められている．しかし，クリティカル・パス上にない整数命令については，レイテンシを伸ばしたとしてもプログラムの実行時間は必ずしも伸びるわけではない．そこで，本研究では，このようなレイテンシ耐性のある命令を低速・低消費電力 ALU に発行し，ALU の消費電力を削減することを提案する．レイテンシ耐性を動的に推定するため，古い命令ほどクリティカル・パス上にある可能性が高いという性質を利用する．同様のアプローチを採る既存の手法は，電力を多く消費する，または回路が複雑でクロック速度に悪影響を与えるという問題があった．これに対し本研究では，小型で単純な回路で古い命令を同定することができ，既存の手法における問題を解決した．評価の結果，性能低下率を平均 0.96%に抑えつつ，コアの消費エネルギーを平均 4.0%削減できることがわかった．本手法は，性能を犠牲にして電力を削減する一般的手法である動的電圧周波数スケーリングを上回る電力削減を達成している．

1. はじめに

現在のプロセッサが消費する電力は，冷却の限界にまで達している．このため，電力を増加させる結果となるクロック速度の向上は制限されており，性能向上が困難になっている．よって，プロセッサの消費電力の削減は非常に重要な課題である．本研究では，ALU/アドレス生成回路（以下，まとめて ALU と称する）の消費電力を削減することを目的とする．ALU は，コアの消費電力の 10% 程度を占める回路である [1], [2]．ALU の中心的回路は加算器である．加算器には様々な方式があるが，一般に，遅延と消費電力の間にはトレード・オフの関係がある．しかし，ALU はプロセッサのクリティカル・パスであるため，現在のハイエンド・プロセッサでは，高い消費電力を犠牲にして短い遅延の回路を使用している [3]．また，一般に，ALU のレイテンシは性能に大きく影響すると考えられており，ALU に長い遅延の加算器を使用しレイテンシを増加させ，これをパイプライン化させると，プログラムの実行時間を伸ばしてしまうと考えられている．このため，1 サイクルでの計算完了が求められている．しかし，すべての整数命令（ALU 命令およびメモリ命令）がデータフローのクリティカル・パス上にあるわけではなく，クリティカル・パス上にない整数命令については，ALU のレイテンシを伸ばしパイプライン化したとしてもプログラムの実行時間は必ずしも伸びるわけではない [4]．

そこで，本研究では，このような命令のレイテンシ耐性を利用して，ALU の消費電力を削減する方式について提案する．まず，従来の高速・高消費電力 ALU の一部を，低速・低消費電力 ALU に変更する．そして，レイテンシ耐性がない（つまり，データフローのクリティカル・パス上にある） 整数命令は従来の高速・高消費電 ALU に発行するが，レイテンシ耐性がある整数命令については，低速・低消費電力 ALU に発行する．これにより多くの整数命令が低電力で処理され，整数命令実行の電力を削減できる．レイテンシ耐性の推定を動的に行うため，基本的に，古い命令ほどクリティカル・パス上にある可能性が高いという性質 [5] を利用する．同様のアプローチを採る既存の手法は存在するが，電力を多く消費する [5]，または回路が複雑でクロック速度に悪影響を与えるという問題があった．これに対して，本研究では小型で単純な回路で命令の古さを推定する．

[†1] 現在，名古屋大学大学院工学研究科
Presently with Graduate School of Engineering, Nagoya University
[a)] kurokawa@ando.nuee.nagoya-u.ac.jp

提案手法ではまず，リオーダ・バッファ（ROB:reorder buffer）を複数の部分に論理的に分割する．分割された部分を ROB の**ウィンドウ**と呼ぶ．ウィンドウは以下で述べるように命令が古いかどうかを同定するために用いる．同定の機会を命令にできるだけ均等に与えるために，ウィンドウは部分的にオーバーラップさせる．命令を発行キュー（IQ: issue queue）にディスパッチするとき，割り当てられた ROB エントリが属しているウィンドウの番号を IQ に書き込む．命令が IQ から発行される際，このウィンドウ番号を読み出す．この番号が，後述する方法で得られる IQ 内の最も古い命令が割り当てられているウィンドウ番号と一致する場合，古い命令と判断する（そうでなければ，古い命令ではないと判断する）．古いと判断された命令は，高速・高消費電力 ALU に送る．そうでない命令は低速・低消費電力 ALU に送る．

IQ 内の最も古い命令が割り当てられている ROB のウィンドウ番号を得るために，**プログラム・オーダ・キュー**（PQ: program order queue）[6] と呼ぶキューを用意する．PQ は，プログラム順に命令が割り当てられる FIFO バッファである．PQ の各エントリは，IQ 内の命令に対応し，当該命令が割り当てられたウィンドウ番号を保持する．命令ディスパッチの際，当該命令が割り当てられた ROB のエントリが属するウィンドウ番号を PQ の末尾に書き込む．毎サイクル PQ の先頭のエントリを読み出すことにより，IQ 内の最も古い命令のウィンドウ番号を得ることができる．

SPEC2017 ベンチマークを用いて評価を行った結果，性能低下率を平均 0.96%に抑えつつ，ALU の消費エネルギーを平均 37.3%削減でき，この結果，コアの消費エネルギーを平均 4.0%削減できることがわかった．性能を犠牲にして電力を削減する一般的手法として，動的電圧周波数スケーリング（DVFS: dynamic voltage/frequency scaling）がある．DVFS では，1%の性能低下で 3%の消費エネルギーが削減される [7] ので，本手法は同一性能低下で DVFS を上回る電力削減を達成している．

以下本論文の構成について述べる．2 節では，提案手法に用いる加算器の構成について述べる．3 節では，提案手法である，レイテンシ耐性を推定し 2 種類の ALU へ分配する手法について述べる．4 節では，評価結果を述べる．5 節では関連研究を述べる．6 節で本研究をまとめる．

2. 提案手法に用いる加算器の構成

本節では，2.1 節で本研究で使用する高速・高消費電力 ALU に含まれる加算器について述べ，次に 2.2 節で低速・低消費電力 ALU に含まれる加算器について述べる．最後に 2.3 節でそれぞれの加算器の遅延・消費エネルギーの測定結果について述べる．

2.1 高速・高消費電力 ALU の加算器

高速・高消費電力 ALU の加算器として，Intel が発表しているスパース・ツリー加算器 [3] を仮定する．本節では，はじめにツリー加算器について簡単に述べ，その後，スパース・ツリー加算器について述べる．

加算器の高速化の鍵は，上位桁への桁上げをいかに速く計算するかである．このために，桁上げの計算回路をツリー上に構成する方法が多く提案されている．図 1 に，基本的な基数 2 のツリー加算器の桁上げ計算回路の構成を示す（8 ビットの例）．同図のセルはグループ PG セルであり，図のセル内の数字はグループの範囲を示す．グループ PG セルとは，グループが桁上げを生成することを示す G 信号と，グループが前方のグループからの桁上げを伝搬することを示す P 信号を計算するセルである．同セルは，2 つのグループ PG 信号から，1 つのグループ PG 信号を計算する．例えば，3 : 0 と示すセルでは，3 : 2 と 1 : 0 から，0 から 3 ビットのグループの PG 信号（3 : 0）を計算している．同図の回路の出力は桁上げである．第 n ビット目から第 (n+1) ビット目への桁上げ入力は，n : 0 の G 信号と等しい．

ツリー加算器では，まず 2 ビットのグループについて，グループ PG 信号を計算し，続いて 4 ビットのグループ，8 ビットのグループと計算を繰り返す．図に示していないが，最終的に，桁上げ計算回路によって求められた桁上げ入力から，和を計算する．このように，ツリー加算器ではツリー構造を用いて桁上げ入力を計算することで高速化を図っている．しかし，すべてのビットについて桁上げ入力を計算するため，長い配線が多い，ファンアウトが多いといった問題がある [8]．

スパース・ツリー加算器 [3] は，4 ビットのグループに対する桁上げのみを計算することでこれらの問題を解決している．スパース・ツリー加算器の桁上げ計算回路の構成を図 3 に示す．セルは，グループ PG セルであり， 4 ビットのグループに対する桁上げ計算であることを強調するためにセルに色をつけている．同図に示すように，出力は 4 : 7 ビット・グループに対する桁上げ（3 : 0），8 : 11 ビット・グループに対する桁上げ（7 : 0）のように，4 ビットのグループに対する桁上げのみであり，4 : 0，5 : 0，6 : 0 などの桁上げは計算していない．そのため，すべての桁上げを計算するツリー加算器に比べて桁上げ計算回路が単純である．

スパース・ツリー加算器では，4 ビットのグループに対する桁上げ入力の計算と並行して，それぞれのグループで桁上げ入力が 0 か 1 かを仮定して，桁上げ選択加算器で和をあらかじめ計算しておく．マルチプレクサで，桁上げ計算回路から得る桁上げ入力により，正しい和を選択する．桁上げ選択加算器を図 2 に示す．空白のセルはグループ PG セルである．XOR セルは桁上げ入力をもとに和を計算す

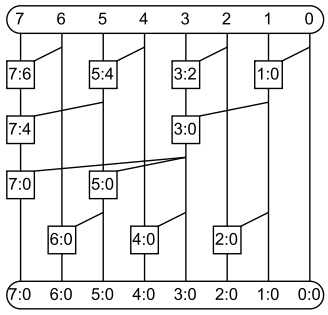

図 1 8 ビット ツリー加算器（基数 2）の桁上げ計算回路

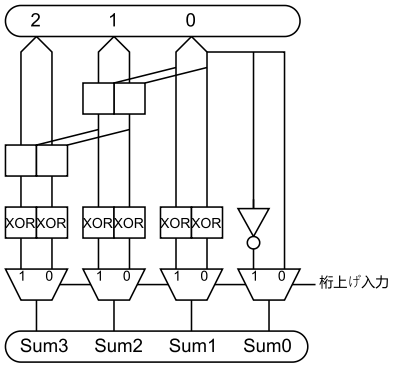

図 2 4 ビット 桁上げ選択加算器

る．桁上げ選択加算器では，短い順次桁上げ加算器を使って，グループへの桁上げ入力が 0 か 1 を仮定して和をあらかじめ計算する．その後，マルチプレクサで選択する．

2.2 低速・低消費電力 ALU の加算器

低速・低消費電力の加算器として，2 サイクルで動作する回路を用意することとする．表 1 に，種々の加算器のグループ PG セル数と桁上げ計算の深さを示す．深さは最長の桁上げ計算経路をいう．一般に，桁上げの深さと遅延には正の相関がある．同表から，スパース・ツリー加算器の桁上げ計算の深さは，6 である．したがって，桁上げ計算の深さが 12 近傍の加算器であれば，遅延がスパース・ツリー加算器の 2 倍程度となり，2 サイクルで実行可能となる可能性が高い．表 1 から，この条件を満たす加算器のうち，最もセル数が少ない加算器は，可変長桁上げインクリメント加算器 [9] と，ブレント・クン・ツリー加算器 [10] である．このうち，消費電力が小さいと予想される加算器を選択する．消費電力は，第 1 にセル数に正の相関を持つが，この他，配線の影響も受ける．配線の観点から，可変長桁上げインクリメント加算器は，ブレント・クン・ツリー加算器に比べて単純であり，消費電力が低くなると予想される．そのため，低速・低消費電力加算器として可変長桁上げインクリメント加算器を使用することとする．インクリメント加算器にはいくつか構成があるが [9]，本研究では，桁上げ計算の深さが最も小さい，可変長桁上げインクリメント加算器を選択した．

図 4 に，64 ビットの可変長桁上げインクリメント加算器の構成を示す．セルはグループ PG セルである．桁上げインクリメント加算器では，グループ内であらかじめグループ PG 信号を計算することで高速化している．前のグループからのグループ PG 信号が利用可能になると桁上げ入力を計算する．例えば，4 : 4，5 : 4，6 : 4 のグループ PG 信号をあらかじめ計算し，3 : 0 のグループ PG 信号が利用可能となると，4 : 0，5 : 0，6 : 0 の桁上げ入力（G 信号）が計算される．可変長桁上げインクリメント加算器ではグループ・サイズを可変長としている．これにより，前のグループの計算とグループ内の計算が同時期に終了するため，より高速に桁上げ入力を計算できる．

表 1 種々の加算器（64 ビット）のセル数と，桁上げ計算回路の深さ [8]

加算方式	セル数	深さ
順次桁上げ	64	63
桁上げ飛越	80	21
可変長桁上げインクリメント	128	11
ブレント・クン・ツリー	128	11
スクランスキ・ツリー	192	6
コギー・ストーン・ツリー	384	6
ハン・カールソン・ツリー	192	7
ラドナ・フィッシャ・ツリー	96	7
ノールズ・ツリー	384	6
スパース・ツリー	135	6

2.3 加算器の遅延・消費エネルギー

本節では，提案手法で使用する加算器の評価結果を示す．MOSIS デサイン・ルール [11] を仮定し，トランジスタ・レベルでレイアウトを描き，回路のサイズを求め，回路シュミレータ HSPICE を用いて遅延と消費エネルギーを評価した．トランジスタ・モデルとして，アリゾナ州立大学が開発した， 16nm Predictive Transistor Model [12] を使用している．遅延評価の結果，可変長桁上げインクリメント加算器の遅延はスパース・ツリー加算器の 1.91 倍であることがわかった．よって，可変長桁上げインクリメント加算器を使う ALU は，2 サイクルで動作可能と言える．また，図 5 にスパース・ツリー加算器の消費エネルギーで規格化した 2 つの加算器の消費エネルギーを示す．可変長桁上げインクリメント加算器の動的消費エネルギーは，スパース・ツリー加算器の消費エネルギーの 30.1%であり，低速・低消費電力 ALU で命令を実行することにより消費エネルギーを削減できると言える．

3. レイテンシ耐性の推定

本節では，はじめに 3.1 節で本提案手法と同様のアプローチを採る既存手法について説明し，本提案手法の利点を述べる．その後，3.2 節で本提案手法の概要を述べ．3.3，3.4 節で手法を詳述する．

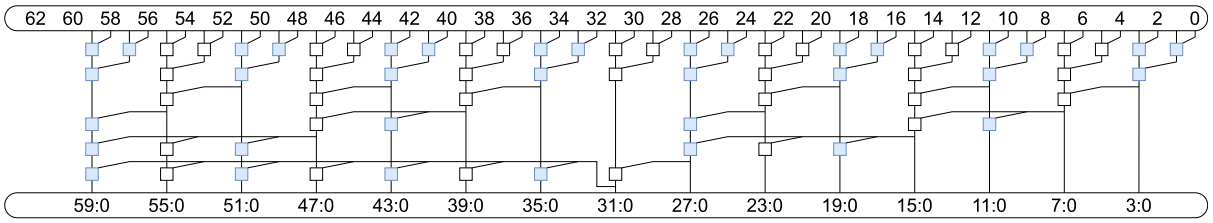

図 3　64 ビット スパース・ツリー加算器の桁上げ計算回路（一部の入力線のビット番号を省略）

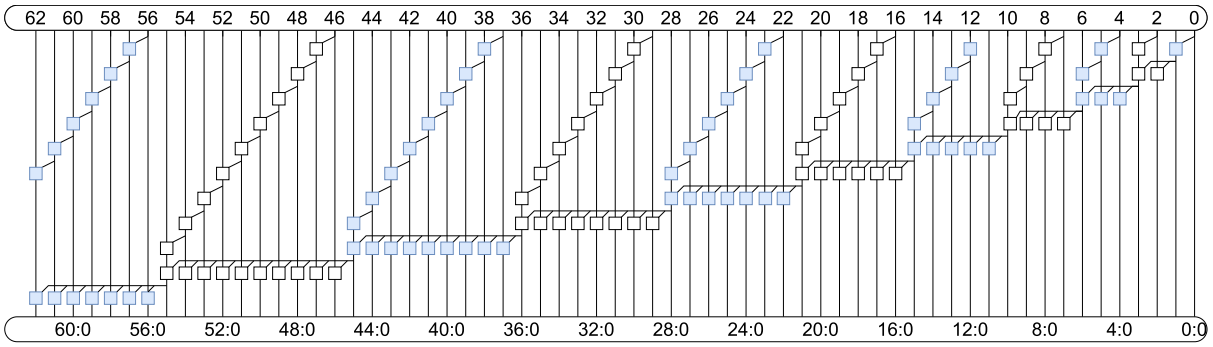

図 4　64 ビット 可変長桁上げインクリメント加算器の桁上げ計算回路（一部の入出力線のビット番号を省略）

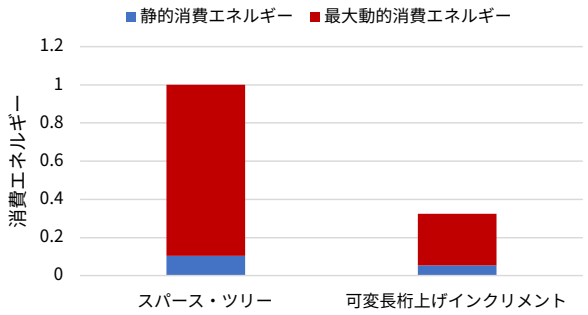

図 5　加算器の消費エネルギー（スパース・ツリー加算器で規格化）

3.1　既存手法

本方式は，レイテンシ耐性の推定を動的に行うため，基本的に，古い命令ほどクリティカル・パス上にある可能性が高いという性質 [5] を利用する．このようなアプローチは，本研究が初めてではなく，過去に同様のアプローチを採った方式が提案されている．最も直接的な方法は，ROB 上で IQ 内の最も古い命令と発行命令の距離に基づく方法である．この方式では，1）上記 2 つの命令が割り当てられた ROB エントリの番号の差を求め，2）その差が予め定めたしきい値以内にあるかを計算する．しきい値以内であれば，発行命令は古い命令と判断し，そうでなければ，古くない命令と判断する．この直接的な方法は，古さを判断する上記 2 つの計算の遅延がクロック・サイクル時間に悪影響を与えるという問題がある．

別な方法として，クリティカル・パス予測器 [5] を使う方法がある [2], [13]．クリティカル・パス予測器は，分岐予測器と同様にエントリに飽和カウンタを持つ予測テーブルを使用する．命令が IQ で最も古い命令となった場合，対応するカウンタの値を増加させる．そうでない場合にはカウンタの値を減少させる．予測の際には，命令が対応する予測テーブルのカウンタをアクセスし，しきい値より大きい場合，この命令はクリティカル・パス上にあると判断する．この方式は，最初に述べた ROB における距離に基づく方法と異なり，発行後に古さを判断するのではなく，フロントエンドで予め判断できるため，プロセッサのクリティカル・パスを伸ばすことがないという利点がある．しかし，十分な予測精度を得るためには，大きな予測テーブルを必要とし，これへのアクセスで消費する電力が大きく，低速・低電力 ALU による電力削減の効果を低減することである．彼らの評価では，予測テーブルには 8 ビット飽和カウンタをエントリに持つ 64K エントリのデーブルが必要としている．

これら既存手法に対して，本提案手法では小型で単純な回路で命令の古さを判定する．

3.2　提案手法の概要

提案手法ではまず，ROB を複数のウィンドウに論理的に分割する．ウィンドウは以下で述べるように IQ から発行する命令が古いかどうかを同定するために用いるが，同定の機会を命令にできるだけ均等に与えるために，ウィンドウは部分的にオーバーラップさせる．命令を IQ にディスパッチするとき，割り当てられた ROB エントリが属しているウィンドウの番号を IQ に書き込む．命令が IQ から発行される際，このウィンドウ番号を読み出す．この番号が IQ 内の最も古い命令が割り当てられているウィンドウ番号と一致する場合，古い命令と判断する（そうでなければ，古い命令ではないと判断する）．古いと判断された命令は，高速・高消費電力 ALU に送る．そうでない命令

は低速・低消費電力 ALU に送る．高速・高消費電力 ALU への発行を試みたが高速・高消費電力 ALU が不足していた場合は，低速・低消費電力 ALU へ発行する．逆に低速・低消費電力 ALU が不足していた場合は高速・高消費電力 ALU へ発行する．

IQ 内の最も古い命令が割り当てられている ROB のウィンドウ番号を得るために，PQ と呼ぶキューを用意する．PQ は，プログラム順に命令を割り当てる FIFO バッファで構成する．PQ の各エントリは，IQ 内の命令に対応し，当該命令が割り当てられたウィンドウ番号を保持する．このために，命令ディスパッチの際，当該命令が割り当てられた ROB のエントリが属するウィンドウ番号を PQ の末尾に書き込む．命令が発行されたら，対応する PQ のエントリを削除する．これにより，毎サイクル PQ の先頭のエントリを読み出すことにより，IQ 内の最も古い命令のウィンドウ番号を得ることができる．

3.3 ウィンドウ

はじめに，ウィンドウの配置と IQ にウィンドウ番号を書き込む動作について図 6 を使って説明する．同図の右側にオーバーラップさせたウィンドウの配置（ウィンドウ数が 4 の例）を示す．ROB を等分に分割し，分割した各部分をウィンドウと呼ぶ．ウィンドウは部分的に重ねる．ROB の各エントリには，そのエントリが属するウィンドウの番号を記録する．ウィンドウはオーバーラップしているので，一般には，ROB エントリは複数のウィンドウ番号を持つ場合がある．実装としては，ROB エントリ番号に対応するウィンドウ番号を保持する ROM を用意する．命令を IQ にディスパッチするとき，この ROM を参照し，割り当てられた ROM エントリが属しているウィンドウの番号を得て，IQ に書き込む．

続いて，オーバーラップによる利点について図 7 を使って説明する．ウィンドウをオーバーラップさせなければ（図 7 の左の図），たとえば，ウィンドウの下から 1/2 の場所に最も古い命令が位置する場合，有効なウィンドウ・サイズは最大ウィンドウ・サイズの 1/2 より小さくなる．最も古い命令より下に位置する命令は発行済みであるからである．しかし，たとえば，ウィンドウの中央から別のウィンドウ（このウィンドウは上記命令を含むとする）をオーバーラップさせると（図 7 の右の図），最も古い命令はこの別のウィンドウにも含まれる．そのため，有効なウィンドウ・サイズが最大ウィンドウ・サイズと同じ大きさに保たれる．一般に，より多くのエントリをオーバーラップさせるほど，有効なウィンドウ・サイズを最大のウィンドウ・サイズに近いサイズに保つことができる．一方，より多くのエントリをオーバーラップさせると，ウィンドウ数が増加し，ウィンドウ番号の IQ での記憶コストが増加するという欠点がある．

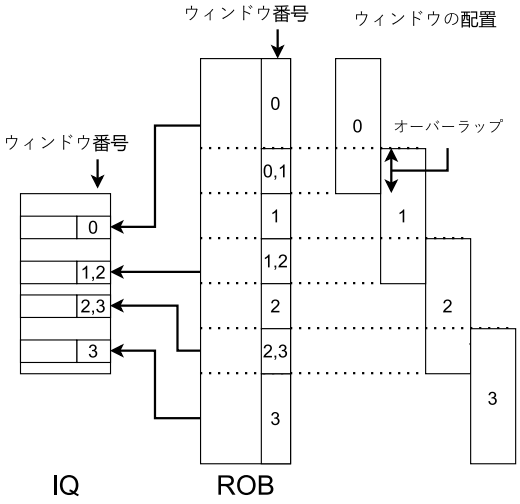

図 6　ウィンドウ番号とそれの IQ への書き込み

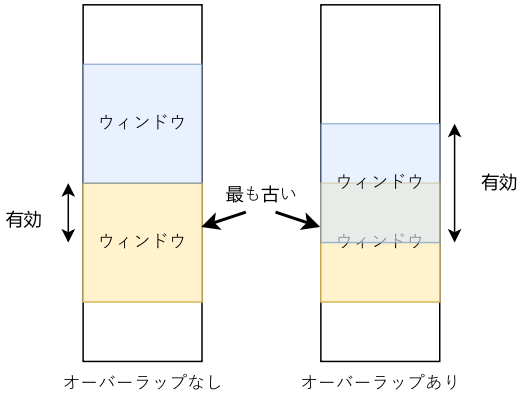

図 7　オーバーラップの利点

最後に，ウィンドウ番号の一致を判断する仕組みについて説明する．本方式では，発行する命令の番号を，PQ の先頭が保持する最も古い命令のウィンドウ番号と比較する必要がある．しかし，比較器を用いるとその遅延によるクロック・サイクルへの悪影響や消費電力の増加という問題が生じる．そこで，IQ の各エントリにウィンドウ番号をビット・ベクタで表すこととする．エントリが保持する命令が属するウィンドウ番号に対応するビットを 1，属さないウィンドウ番号に対応するビットを 0 とする．この方法では，命令が属するウィンドウ番号を比較器を用いることなく，AND ゲートと OR ゲートのみで最も古いウィンドウに属しているか判断することができる．ウィンドウ番号の記録方式と比較方法の概略図を図 8 に示す．同図は，発行される命令が最も古い命令ウィンドウに属している場合を示している．0 ビット目の AND 演算の結果が 1 となるため，OR 演算の結果が 1 となり，発行する命令は古い命令と判断されて高速・高消費電力 ALU へ発行される．なお，この演算は発行後のレジスタ読み出しと命令の機能ユニットへの分配のステージにおいて，レジスタ読み出しの時間に並行して行われるため，クロック・サイクル時間を伸ばすことはない．

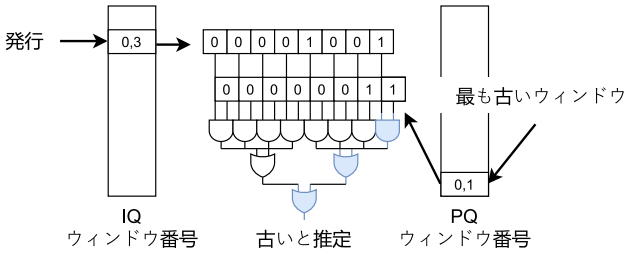

図 8　古い命令を判断する方法

3.4　PQ

PQ [6] は，プログラム順に命令が割り当てられる FIFO バッファである．PQ の各エントリは，IQ 内の命令に対応し，当該命令が割り当てられたウィンドウ番号を保持する．命令ディスパッチの際には，当該命令が割り当てられた ROB のエントリが属するウィンドウ番号を PQ の末尾に書き込む．命令が発行されたら，対応する PQ のエントリを削除する．これにより，毎サイクル PQ の先頭のエントリを読み出すことにより，IQ 内の最も古い命令のウィンドウ番号を得ることができる．

また，本研究では ALU に着目しているため，PQ には ALU を使用する命令[*1]のみを挿入する．ALU を使用する命令と，浮動小数点命令の間にはほとんどデータ依存がなく，これら 2 種類の命令は同じデータフロー・グラフには含まれない．そのため，浮動小数点命令を PQ に挿入すると，ALU を使用する命令の古さが正しく判断されなくなる懸念がある．

PQ の回路を単純に保つため，先頭と末尾の間のエントリが無効化されても圧縮しない. このため，PQ に穴が空き，容量効率が低下するので，PQ は IQ のサイズよりも大きくする必要がある．完全に不足しないようにするには ROB のサイズが必要である．PQ のサイズを ROB 以下とし，PQ に空きがなくディスパッチできないときは，2 通りの方法が考えられる. ひとつは PQ にディスパッチできるエントリができるまで，ディスパッチをストールさせる方法である. この方法では，IQ に命令が十分に供給できなくなり，性能が大きく低下すると思われる．もうひとつは PQ に空きがない場合は命令を PQ に挿入しないという方法である. この方法では，一部の命令が挿入されないことにより PQ が誤った命令を最も古い命令として選択する可能性がある．PQ のサイズや，PQ の容量不足時の対応による性能の違いは，4.3 節で評価する．

4.　評価

本節では，提案手法による性能と消費エネルギーの変化を評価する．はじめに 4.1 節で評価環境と評価方法について述べ，4.2 節で性能変化および高速・高消費電力 ALU への発行率について評価する．続いて，4.3，4.4，4.5，4.6 節で，それぞれ，PQ サイズ，ウィンドウ・サイズ，オーバーラップ・サイズ， 高速・高消費電力 ALU と低速・低消費電力 ALU の数を変化させた場合について評価する．最後に，4.7 節で消費エネルギーの削減率について述べる．

[*1] 繰り返しになるが，本論文では，ALU とアドレス計算回路をまとめて ALU と呼んでおり，ALU を使用する命令とは，ALU 命令とメモリ命令である．

表 2　ベース・プロセッサの基本構成

Fetch width	6-instruction wide
Decode width	6-instruction wide
Issue width	8-instruction wide
Commit width	8-instruction wide
ROB	512 entries
IQ	256 entries
Load/Store queue	256 entries
Physical registers	int and fp 512 registers each
Branch prediction	36-histry length, 512 entry perceptron predictor, 2K-set, 4-way BTB, 15-cycle misprediction penalty
Function unit	8 iALU, 2 iMULT, 2 Ld/St, 4 fpU
L1 I-cache	32KB, 8-way, 64B line,
L1 D-cache	32KB, 8-way, 64B line, 2 ports, 4-cycle hit latency
L2 cache	2MB, 16-way, 64B line 12-cycle hit latency
Main memory	300-cycle latency, 16B/cycle bandwidth
Data prefetcher	stream-based, 32-stream tracked, 16-line distance, 2-line degree, prefetch to L2 cache

表 3　最善構成の場合の本手法のパラメータ

Program order queue	384 entries, 1R/6W
ROB window	64 entries，32-entry overlap
Function unit	4 high-iALU, 4 low-iALU 2 iMULT, 2 Ld/St, 4 fpU

4.1　評価方法

ベース・プロセッサの構成を表 2 に示す．性能は，SimpleScalar [14] をベースにシミュレータを作成し，SPEC2017 ベンチマークを用いて評価した．消費エネルギーは，McPAT [15] を修正して評価した．シミュレータに追加した低速・低消費電力 ALU の消費エネルギーは 2.3 節の評価をもとに定めた．動的消費エネルギーは McPAT が仮定している ALU（高速・高消費電力 ALU）の 30.1%の消費エネルギーとし，静的消費エネルギーは McPAT が仮定している ALU の 27.4%の消費エネルギーとした．提案手法を導入したプロセッサは，低速・低消費電力 ALU と高速・高消費電力 ALU をそれぞれ 4 つとした．総数はベース・プロセッサの ALU 数と同じである．

本方式では，より大きな性能低下を許すと消費電力をより削減できるというトレードオフが存在するが，本研究で

は設計の方針として消費エネルギー削減より性能を優先する．具体的には，性能低下を最大 1%程度許すこととし，そのときの構成を最善の構成とする．表 3 に，最善の構成における本手法のパラメータを示す．以下の評価では，特に断らない限り，この構成をデフォルトとする．

4.2 提案手法による性能変化

本節では，最善の構成での性能変化及び高速・高消費電力 ALU への発行率を評価する．より多くの命令が低速・低消費電力 ALU へ発行されれば，消費エネルギー削減が期待できるため，高速・高消費電力 ALU への発行率は低いほうが良い．図 9 に全 ALU を低速・低消費電力 ALU とした場合のベースラインに対する性能低下率（IPC 低下率）を示す．図からわかるように，平均 9.0%性能が低下している．このことから，ALU として部分的に高速のものを用意する必要があることがわかる．

図 10 に最善の構成でのベースラインに対する性能低下率を示す．性能低下は平均 0.96%であり，目標の 1%以下を達成している．また， 全ての ALU を低速・低消費電力とした場合に比べ，大きく性能を回復している．平均（GM）を見ると，int ベンチマークでは性能低下が大きく，fp ベンチマークでは小さい．これは，int ベンチマークに含まれるプログラムは，整数命令の割合が多く，一方で，fp ベンチマークに含まれるプログラムは，多くの浮動小数点命令を含み，整数命令の割合が少ないからである（浮動小数点ユニットには低速・低消費電力ユニットはない）．

ALU へ発行される命令の内訳を図 11 に示す．青色の部分（high/old）は古い命令と判断され，実際に高速・高消費電力 ALU へ発行された命令であり，黄色の部分（low/new）は，新しい命令と判断され，低速・低消費電力 ALU へ発行された命令である．high/old と low/new を合わせ平均 87.8%の命令はレイテンシ耐性の予測に従って発行されている．一方，予測とは異なる ALU へ発行されている命令（赤色と緑色の部分）もある．赤色の部分（high/new）は，新しい命令と判断されたが，低速・低消費電力 ALU に空きがなかったため，高速・高消費電力 ALU へ発行された命令であり，消費エネルギー削減の機会を逃している可能性がある．緑色の部分（low/old）は，古い命令と判断されたが，高速・高消費電力 ALU に空きがなかったため，低速・低消費電力 ALU へ発行された命令であり，クリティカル・パスを伸ばし，性能に悪影響を与える可能性がある．いずれの場合も，一般には，このような期待しない発行の割合は少なく，性能，消費エネルギーともに与える影響は小さいと考える．例外として，xz が挙げられる．xz では，low/old の割合が 10%を超えており高い．このため，IPC 低下率が高くなったと考えられる．

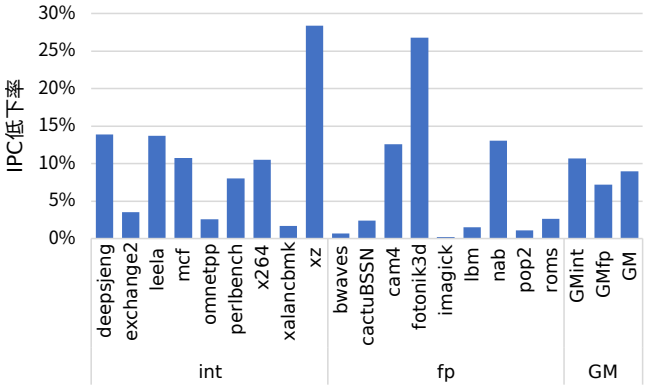

図 9 全ての ALU を低速・低消費電力とした場合の性能低下率

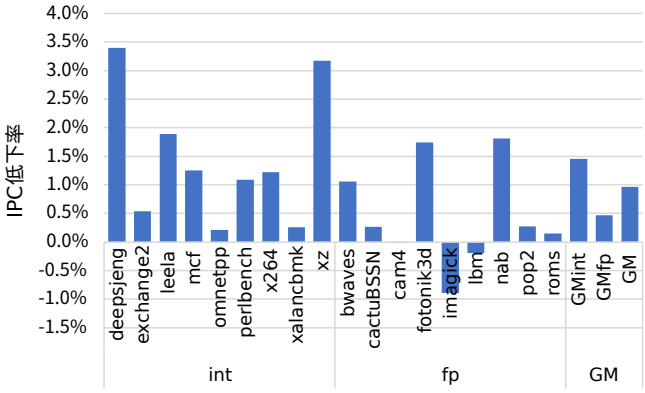

図 10 提案手法における性能低下率

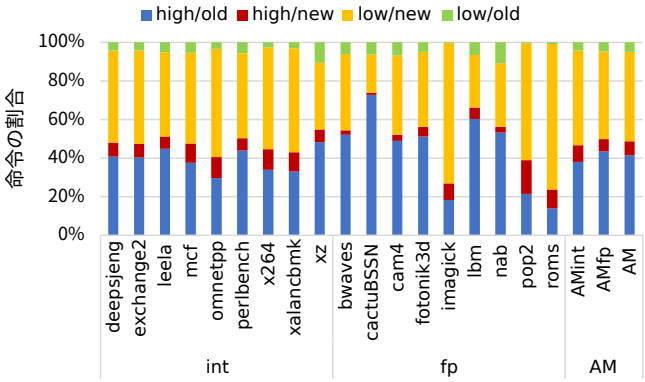

図 11 ALU への発行される命令の内訳

4.3 PQ サイズを変化させた場合の性能変化

本節では，PQ のサイズと PQ が容量不足となったときの対応を変更した時の IPC 低下率および PQ に入る命令の割合を評価する．評価結果を図 12 に示す．青色の棒グラフは PQ の容量不足時にディスパッチをストールさせるモデルの性能低下率（左軸）を表している．赤色の棒グラフと黄色の折れ線グラフは，それぞれ，PQ の容量不足時に命令を挿入しないモデルの性能低下率（左軸）と PQ 命令挿入率（右軸）を表している．同図からわかるように，PQ の容量不足時にストールするモデルでは，大幅な性能低下が生じている．一方，命令を挿入しないモデルでは，性能低下は緩やかであり，30%ほどの命令が PQ に非挿入となるモデル（PQ サイズ 128）でも性能低下を平均 1.3%

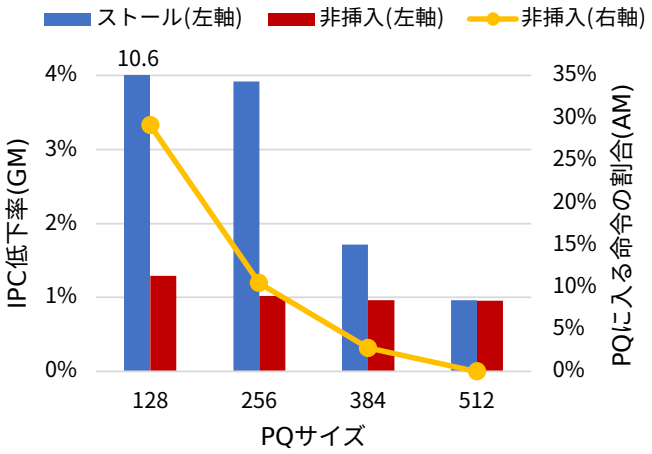

図 12　性能低下率と PQ に入る命令の割合

に抑えている．本研究では，性能低下を抑制できていることを重視し，平均 0.96%の性能低下である PQ サイズとして，384 エントリを採用する．このエントリのモデルでは，PQ に挿入されなかった命令は平均 3%以下であり，512 エントリのモデルと比較しての性能低下率は平均 0.01%以下である．

4.4　ウィンドウ・サイズを変化させた場合の性能変化

本節では，ウィンドウ・サイズを変化させた時の性能変化率および高速・高消費電力 ALU への発行率を評価する．オーバーラップ・サイズはウィンドウ・サイズの 1/2 とした．評価結果を図 13 に示す．青色の棒グラフは性能低下率である（左軸）．赤色の折れ線グラフは高速・高消費電力 ALU への発行率である（右軸）．ウィンドウ・サイズが 32 エントリのモデルでは高速・高消費電力 ALU への発行を平均 41.6%に抑えているものの，性能低下率は平均 1.4%と 1%を上回っている．64 エントリのモデルは性能低を平均 0.96%に抑え，平均 48.6%の命令を高速・高消費電力 ALU へ発行している．128 エントリのモデルは性能低下を平均 0.77%に抑えているものの，高速・高消費電力 ALU への発行率が平均 57.7%と高い．そのため，消費電力削減率が 64 エントリのモデルに比べて低いことが予想される．以上から，性能低下率が平均 1%以下であり高速・高消費電力 ALU への発行率が低い，64 エントリのモデルを採用する．

4.5　オーバーラップ・サイズを変化させた場合の性能変化

本節では，オーバーラップ・サイズを変化させた時の性能変化率および高速・高消費電力 ALU への発行率を評価する．評価結果を図 14 に示す．オーバーラップ・サイズは，ウィンドウ・サイズに対しての割合で示している．オーバーラップ・サイズが 0 エントリ（オーバーラップを行わない）のモデルでは，性能低下率が平均 1.4%と高く，1%を上回っている．一方で，ウィンドウ・サイズの 1/2 のモデルは性能低下率が平均 0.96%，ウィンドウ・サイズの

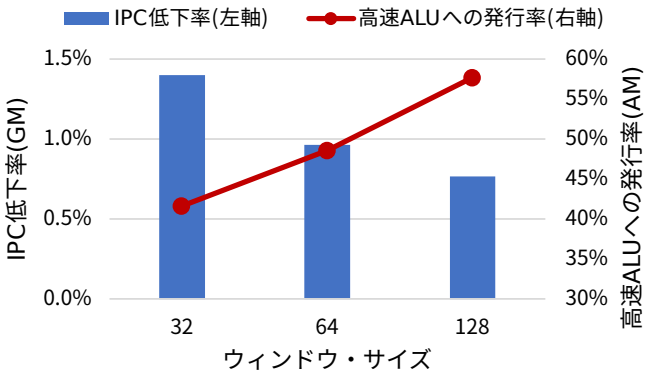

図 13　ウィンドウ・サイズを変化させたときの性能低下率（オーバーラップ・サイズはウィンドウ・サイズの 1/2）

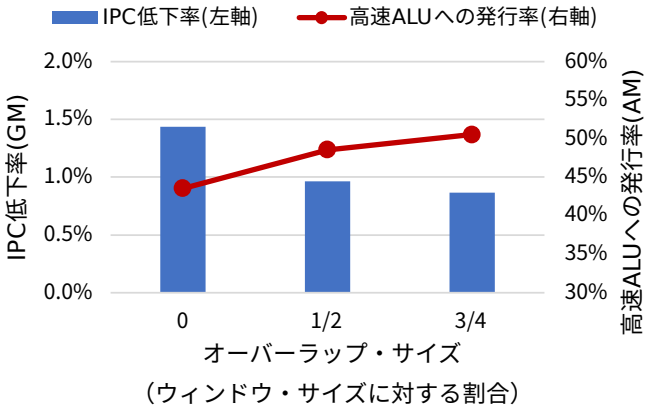

図 14　オーバーラップ・サイズ（ウィンドウ・サイズに対する割合）を変化させたときの性能低下率

3/4 のモデルは性能低下率が平均 0.87%である．オーバーラップを行うモデルでは性能低下を 1%以下に抑制できていることがわかる．オーバーラップの効果が示されたと言える．以上から，性能低下率が 1%以下であり高速・高消費電力 ALU への発行率が低いウィンドウ・サイズの 1/2 を採用する．ウィンドウ・サイズの 1/2 のモデルでは，ウィンドウ番号の情報は 16 ビットで表すことができる．

4.6　高速・高消費電力 ALU と低速・低消費電力 ALU 数を変化させた場合の性能変化

本節では，高速・高消費電力 ALU，低速・低消費電力 ALU の数を変化させた場合の性能変化率および高速・高消費電力 ALU への発行率を評価する．2 種類の ALU の数の合計はベースライン・プロセッサと同じ 8 である．評価結果を図 15 に示す．横軸は高速・高消費電力 ALU の数を表す．低速・低消費電力 ALU の数は 8 から高速・高消費電力 ALU の数を減じた値である．青色の棒グラフは性能低下率を表している（左軸）．赤色の折れ線グラフは高速・高消費電力 ALU への発行率を表している（右軸）．図からわかるように，性能低下率は高速・高消費電力 ALU の数が増加するとともに減少する．高速・高消費電力 ALU 数が 3 のモデルでは平均 1.4%であるが，4 つのモデルでは平

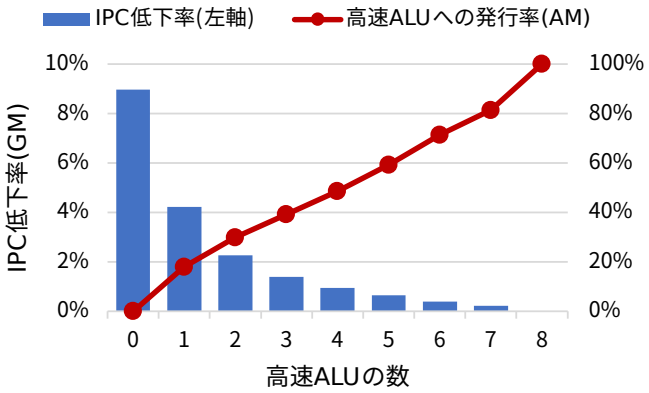

図 15　高速・高消費電力 ALU の数を変化させた時の性能低下率と高速・高消費電力 ALU への発行率

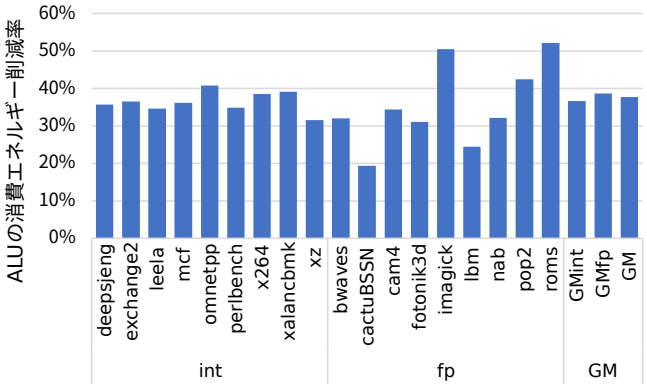

図 16　ALU の消費エネルギー削減率

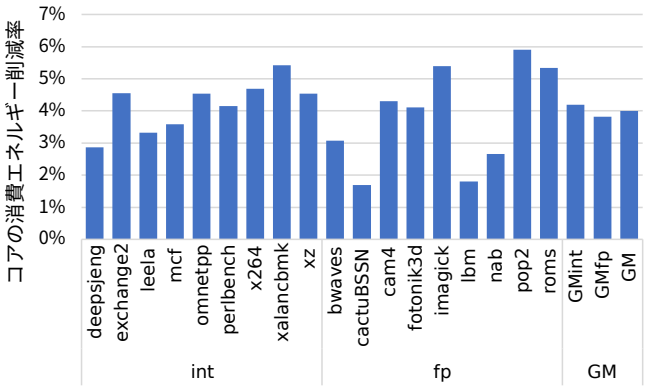

図 17　コアの消費エネルギー削減率

均 0.96%と，1%を下回っている．また，高速・高消費電力 ALU への発行率は高速・高消費電力 ALU 数の増加とともに増加する．性能低下率が 1%を下回っているモデルのうち，高速・高消費電力 ALU への発行率が最も低い高速・高消費電力 ALU が 4 つ，低速・低消費電力 ALU が 4 つのモデルが適していると言える．

4.7　提案手法による消費エネルギー変化

本節では，提案手法による消費エネルギー変化率を評価する．ALU の消費エネルギー削減率の評価結果を図 16 に示す．平均 37.6%削減できている．低速・低消費電力 ALU の動的消費エネルギーが高速・高消費電力 ALU の 1/3 倍程度である（2.3 節）ことと，高速・高消費電力 ALU へ発行される命令が 50%程度である（4.2 節）ことからは，ALU の消費エネルギー削減率は以下の式から 33%と予想される．

$$1-\left(\frac{1}{2}\times 1+\frac{1}{2}\times\frac{1}{3}\right)=\frac{1}{3}$$

実際にはより多くの消費エネルギーを削減することができた．これは，低速・低消費電力 ALU のトランジスタ数が高速・高消費電力 ALU の 40%程度と少なく，静的消費エネルギーの削減率が 27.4%となったためと考えられる．

コアの消費エネルギー削減率の評価結果を図 17 に示す．平均 4.0%の消費エネルギー削減率を達成した．浮動小数点ベンチマークは整数命令が少ないため，多くのプログラムで整数ベンチマークと比較して消費エネルギー削減率が低い．性能を犠牲にして電力を削減する一般的手法として， DVFS があるが，DVFS では，1%の性能低下で 3%の消費エネルギーが削減される [7] ので，本手法は同一性能低下で DVFS を上回る電力削減を達成している．

5.　関連研究

本節では，5.1 節で加算器に関する関連研究についてまとめる．次に 5.2 節で命令のレイテンシ耐性に関する関連研究についてまとめ，最後に 5.3 節で加算器の消費エネルギー削減に関する研究をまとめる．

5.1　加算器に関する研究

Morgan らは，4 ビットごとの桁上げを先に計算し，これを長い桁上げ計算を飛び越すことに使用することで，桁上げ計算の深さを $N/4+5$（N:加算桁数）に減少させる桁上げ飛越加算器 [16] を提案した．

Zimmermann らは，桁上げ計算のグループ毎にセル数を 1 つずつ増加させ，桁上げ計算の深さを，$\sqrt{2N}$ に減少させる可変長桁上げインクリメント加算器 [9] を提案した．

Kogge らは，2 ビットごとの桁上げを計算し，ツリー構造を用いて，桁上げ計算の深さを $\log_2 N$ に減少させるコギー・ストーン・ツリー加算器 [17] を提案した．

ツリー加算器には複数の方式があり，スクランスキ・ツリー加算器 [18]，ノールズ・ツリー加算器 [19] も桁上げ計算の深さが $\log_2 N$ である．また，ハン・カールソン・ツリー加算器 [20]，ラドナ・フィッシャー・ツリー加算器 [21] は桁上げ計算の深さが，$\log_2 N+1$ であり，ブレント・クン・ツリー加算器 [10] は深さが $2\log_2 N-1$ である．これらのツリー加算器は，配線の数，ファンアウトの大きさ，回路面積についてトレード・オフの関係にある [8]．

Mathew らは，4 ビット単位での桁上げ選択加算を行うことで，コギー・ストーン加算器の，配線数が多い問題を改良したスパース・ツリー加算器 [3] を提案した．

5.2 レイテンシ耐性に関する研究

Fields らは，スラックがある命令の割合について調査した [4]．ここで，スラックとは，プログラム全体の実行時間を伸ばすことなく命令のレイテンシを増加させることができるサイクル数のことである．調査の結果，平均で 75%の命令はレイテンシを 2 倍にしてもプログラム全体の実行時間を伸ばさないことがわかった．これは，平均で 75%の命令にレイテンシ耐性があることを意味している．

Tune らは，実行時にクリティカル・パス予測を行うクリティカル・パス予測器を提案した [5]．クリティカル・パス予測器は，クリティカル・パス上の命令は発行キューの最下部に繰り返し到達することに着目した手法である．彼らは，現在のプロセッサとは異なり，命令がプログラム順に並んでいる IQ を仮定しているため，最下部に到達することは，発行キュー内で最も古いことを意味する．クリティカル・パス予測器は，分岐予測器と同様に飽和カウンタ予測テーブルを利用し，過去の動作から将来のクリティカリティを予測するが．彼らの評価では，十分な予測精度を得るためには，予測テーブルに 64K エントリの 8 ビット飽和カウンタが要求され，大きな電力を消費する．

Pyreddy らは，クロック周波数の低い低速・低消費電力ユニットを用意し，クリティカル・パス上にない命令をこれらのユニットに発行することを提案した [2]．クリティカル・パスの予測は，クリティカル・パス予測器 [5] の手法をベースに，プロファイルを使用し静的に行っている．

Seng らは，クリティカル・パス予測器 [5] を使用し，動的にクリティカル・パス上にない命令を推定して低速・低消費電力 ALU で実行することで，消費電力あたりの性能が向上することを示している [13]．一方，予測テーブルのエントリ数を 32K にすると 2.5%性能が低下することが示されており，予測テーブルが大きな消費電力を消費するという課題が残る．

5.3 ALU の消費エネルギー削減に関する研究

ALU の消費エネルギー削減に関する研究では，主に一部の ALU を従来よりも消費電力の低い ALU に置き換えるものである．消費電力の低い ALU を作成するアイデアは複数あり，前節で述べたように，Pyreddy らは，周波数を下げることで ALU を含む低速・低消費電力ユニットを実現し，クリティカル・パス上にないと推定した命令を低速・低消費電力ユニットに発行している [2]．

また，Minana らは，オペランドのビット幅が最大データ幅（64 ビット）の半分より小さい命令は 32 ビットの低消費電力 ALU へ発行し，それ以外の命令は通常の高消費電力 ALU に発行する方式を提案した [22]．

6. まとめ

本研究では，レイテンシ耐性のある命令を低速・低消費電力 ALU に発行し，ALU の消費電力を削減することを提案した．レイテンシ耐性の推定を動的に行うため，古い命令ほどクリティカル・パス上にある可能性が高いという性質を利用する．この性質を利用した既存手法は，電力を多く消費する，または回路が複雑でクロック速度に悪影響を与えるという問題があった．これに対して，本手法では小型で単純な回路で，IQ 内の最も古い命令とプログラム順で近い整数命令をレイテンシ耐性のない命令と推定し，これらの問題を解決した．評価の結果，性能低下率を平均 0.96%に抑えつつ，コアの消費エネルギーを平均 4.0%削減できることがわかった．本手法は，性能を犠牲にして電力を削減する一般的手法である DVFS を上回る電力削減を達成している．

参考文献

[1] Gowan, M., Biro, L. and Jackson, D.: Power considerations in the design of the Alpha 21264 microprocessor, *Proceedings 1998 Design and Automation Conference*, pp. 726–731 (1998).
[2] Pyreddy, R. and Gary, T.: Evaluating Design Tradeoffs in Dual Speed Pipelines, *Proceedings of Workshop on Complexity-Effective Design*, pp. 10–15 (2001).
[3] Mathew, S., Anders, M., Bloechel, B., Nguyen, T., Krishnamurthy, R. and Borkar, S.: A 4-GHz 300-mW 64-bit integer execution ALU with dual supply voltages in 90-nm CMOS, *IEEE Journal of Solid-State Circuits*, Vol. 40, No. 1, pp. 44–51 (2005).
[4] Fields, B., Bodik, R. and Hill, M.: Slack: maximizing performance under technological constraints, *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pp. 47–58 (2002).
[5] Tune, E., Liang, D., Tullsen, D. and Calder, B.: Dynamic prediction of critical path instructions, *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, pp. 185–195 (2001).
[6] Matsuda, Y., Shioya, R. and Ando, H.: Reducing Energy Consumption of Wakeup Logic through Double-Stage Tag Comparison, *IEICE Transactions on Information and Systems*, Vol. E105.D, No. 2, pp. 320–332 (2022).
[7] Gochman, S., Ronen, R., Anati, I., Berkovits, A., Kurts, T., Naveh, A., A.Saeed, A., Sperber, Z. and Valentine, R. D.: The Intel Pentium M Processor : Microarchitecture and Performance, *Intel Technology Journal*, pp. 21–36 (2003).
[8] Weste, N. and Harris, D.: *CMOS VLSI Design: A Circuits and Systems Perspective*, Addison-Wesley Publishing Company, USA, 4th edition (2010).
[9] Zimmermann, R.: Non-Heuristic Optimization and Synthesis of Parallel-Prefix Adders, *Proceedings of the International Workshop on Logic and Architecture Synthesis*, pp. 123–132 (1996).
[10] Brent and Kung: A Regular Layout for Parallel Adders, *IEEE Transactions on Computers*, Vol. C-31, No. 3, pp. 260–264 (1982).
[11] https://www.mosis.com/.
[12] http://ptm.asu.edu/.
[13] Seng, J., Tune, E. and Tullsen, D.: Reducing power with dynamic critical path information, *Proceedings of*

the 34th ACM/IEEE International Symposium on Microarchitecture., pp. 114–123 (2001).
[14] http://www.simplescalar.com/.
[15] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M. and Jouppi, N. P.: McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures, *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 469–480 (2009).
[16] Morgan, L. P. and Jarvis, D. B.: Transistor logic using current switching and routing techniques and its application to a fast 'carry' propagation adder, *Proceedings of the the IEE - Part B: Electronic and Communication Engineering*, Vol. 106, No. 29, pp. 467–468 (1959).
[17] Kogge, P. M. and Stone, H. S.: A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations, *IEEE Transactions on Computers*, Vol. C-22, No. 8, pp. 786–793 (1973).
[18] Sklansky, J.: Conditional-Sum Addition Logic, *IRE Transactions on Electronic Computers*, Vol. EC-9, No. 2, pp. 226–231 (1960).
[19] Knowles, S.: A family of adders, *Proceedings of 15th IEEE Symposium on Computer Arithmetic*, pp. 277–281 (2001).
[20] Han, T. and Carlson, D. A.: Fast area-efficient VLSI adders, *Proceedings of 1987 IEEE 8th Symposium on Computer Arithmetic*, pp. 49–56 (1987).
[21] Ladner, R. E. and Fischer, M. J.: Parallel Prefix Computation, *Journal of ACM*, Vol. 27, No. 4, pp. 831–838 (1980).
[22] Minana, G., Garnica, O., Hidalgo, J., Lanchares, J. and Colmenar, J.: A Power-Aware Technique for Functional Units in High-Performance Processors, *Proceedings of the 9th EUROMICRO Conference on Digital System Design*, pp. 456–459 (2006).