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

2006－ＡＲＣ－168（３）
2006／6／8

依存情報を用いた命令グループ化による動的命令スケジューリング機構の電力削減手法

佐々木　広†　　近藤　正章†　　中村　宏†

† 東京大学 先端科学技術研究センター　〒153-8904 東京都目黒区駒場 4-6-1
E-mail: †{sasaki,kondo,nakamura}@hal.rcast.u-tokyo.ac.jp

あらまし　動的命令スケジューリング機構は，その複雑さから多命令同時発行，アウト・オブ・オーダー実行を行うマイクロプロセッサにおいて，主要な電力消費要素の一つである．近年，さらなる高速化のため，動的命令スケジューリング機構のポート数やエントリ数が増加傾向にある．しかし，これは動的命令スケジューリング機構のアクセス時間や消費電力の増大といった問題を引き起こす．そこで，本稿では依存情報を用いた命令のグループ化による動的命令スケジューリング機構の電力削減手法を提案する．提案手法はグループ化した命令を単一の発行単位として扱うことによって，複雑さを低減する．本稿は，提案手法のためのマイクロアーキテクチャの拡張，評価について述べる．
キーワード　動的命令スケジューリング，命令キュー，命令グループ化

Reducing Energy Consumption of the Dynamic Scheduling Logic by Instruction Grouping

Hiroshi SASAKI†, Masaaki KONDO†, and Hiroshi NAKAMURA†

† Research Center for Advanced Science and Technology The University of Tokyo
Komaba 4-6-1, Meguro-ku, Tokyo, 153-8904 Japan
E-mail: †{sasaki,kondo,nakamura}@hal.rcast.u-tokyo.ac.jp

Abstract　Dynamic instruction scheduling logic is a quite complex component and dissipates significant energy in microprocessors which support superscalar and out-of-order execution. We propose a novel microarchitectural technique to reduce the energy consumption of the dynamic instruction scheduling logic. The proposed method groups several instructions as a single issue unit and reduces the required number of ports and size of the structure. This paper describes the microarchitecture mechanisms and shows evaluation results on energy saving and performance.
Key words　Dynamic Instruction Scheduling, Instruction Queue, Instruction Grouping

1. はじめに

近年の汎用のマイクロプロセッサでは，多命令同時実行，アウト・オブ・オーダー実行が可能なものが多い．汎用マイクロプロセッサは，様々なアプリケーションを効率的に実行することが求められ，また旧来のバイナリコード資産を実行する場合もあることから，高速化のためには実行時に並列性を抽出することが必要となる．また，携帯電話をはじめとする携帯機器や組み込み機器の高機能化，また多様化を受け，今後はそれらに搭載される組込型プロセッサにおいても，汎用のプログラムを高速に実行することが必須となり，多命令同時実行やアウト・オブ・オーダー実行といった動的な命令レベル並列性抽出技術が必要になると考えられる．

しかし，多命令同時実行，アウト・オブ・オーダー実行を行う動的命令スケジューリング機構の問題点として，ハードウェアの複雑化による消費電力の増大が挙げられる．特に，バッテリ駆動の携帯機器では消費電力増大は許容できない問題である．また，現在ではハイエンドシステムにおいても消費電力増大にともなう発熱量の増大が深刻化しており，複雑な動的命令スケジューリング機構を用いることが難しくなっている．このため，多命令同時実行や，アウト・オブ・オーダー実行機構のハードウェアの簡単化，低消費電力化は，非常に重要な課題である．

動的命令スケジューリングを行う上で中心的な役割を果たす機構として，命令が発行可能になるまで命令の情報を保持する命令キュー (命令ウィンドウ)，およびキューの中から発行可能な命令を選択する命令スケジューリング機構がある．命令レベル並列度 (ILP: Instruction Level Parallelism) を可能な限り抽出して，プロセッサを高性能化するためには，大容量の命令

キュー，および多数の同時命令発行が必須である．しかし，一般的にキューサイズやポート数の増加により，消費電力は増大してしまう．そこで，従来より，命令キューや命令スケジューリング機構を対象に，複雑化を抑え，消費電力を削減する，あるいはサイクルタイムを短縮するための手法が数多く提案されている[1]～[10].

本稿では，命令キュー，および命令スケジューリング機構の消費電力削減手法の一つとして，依存情報を用いた命令グループ化による手法を提案する．本手法は，依存情報を用いて命令をグループ化し，そのグループを一つの命令発行単位として扱うことで，命令キューやスケジューリング機構のサイズ／ポート数の増加を抑えつつ，より多くの命令の保持，および発行を行うものである．本機構により，従来の動的命令スケジューリング機構よりも少ないハードウェア量で，ほぼ同等，あるいはそれ以上の性能が得られ，また消費電力を大きく削減できる．

本論文の構成は以下のとおりである．次節において，動的命令スケジューリング機構の概要を述べ，3 章にて提案手法のアイデア，およびマイクロアーキテクチャを示す．4 章では性能評価環境および評価条件について説明し，5 章で評価結果を示す．6 章で関連研究についてまとめ，7 章で本論文のまとめと，今後の課題について述べる．

2. 従来の動的命令スケジューリング機構

一般的な動的命令スケジューリング機構の概要を以下に述べる．命令はレジスタ・リネーミングステージを経て命令キューにイン・オーダーに格納 (ディスパッチ) され，それぞれのソース・オペランドが揃い次第，アウト・オブ・オーダーに発行されて演算が行われる．命令のスケジューリングは命令のウェイクアップと，セレクトという二つのフェーズからなる．ウェイクアップでは，すでに発行された命令のデスティネーションタグが命令キューにブロードキャストされ，キュー内の全ての待機している エントリに対して，ストアされているソース・レジスタのタグとの連想マッチが行われる．連想マッチングでは，タグが一致したソースはレディであるとマークされる．命令のソースが全てレディとマークされたら，命令はウェイクアップされセレクトの対象となる．セレクトでは，選択可能な命令 (最大で N) の中から W 命令を選ぶ．ここで，N は命令キューのエントリ数であり，W はプロセッサの発行幅である．

命令キューの構成はフル・アソシアティブであり，毎サイクル新しい命令を発行するためのウェイクアップとセレクトを行うことが潜在的にどのエントリに対しても可能である．したがって，命令キューへのアクセスは毎サイクル，命令のディスパッチ (命令の書き込み)，ウェイクアップのためのタグのブロードキャスト，セレクト・発行 (命令の読み出し) 時に起こり，プロセッサの主要な電力消費要素であると言われている．

図 1 に 4 命令同時発行可能なプロセッサにおける従来の動的命令スケジューリング機構の構造を示す．図のように 1 サイクルに各々最大で，ディスパッチ時に 4 命令書き込みを行い，4 命令セレクトし，4 命令発行することが可能である．つまり命令キューは命令の書き込み，読み出し用にポートをそれぞれ 4 つずつ，またセレクトロジックは 4 命令をセレクトする回路を備えている．

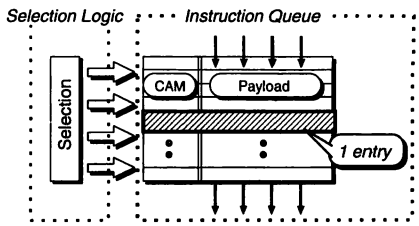

図 1　従来の動的命令スケジューリング機構

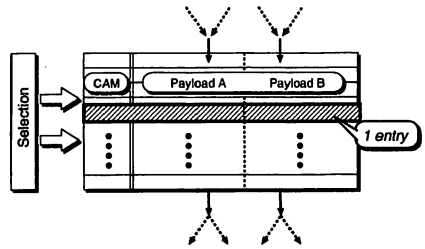

図 2　命令をグループ化した場合のイメージ図

3. 提案手法

本研究における目的は，動的命令スケジューリング機構の複雑さと消費電力を削減することである．本章では提案手法の概要およびマイクロアーキテクチャの実装について述べる．

3.1 概　　要

本稿では，複数の命令をグループ化し，1 つの発行単位として扱うことによって動的命令スケジューリング機構のサイズ／ポート数を低減する手法を提案する．ディスパッチステージにおいて複数の命令をグループ化し，発行時までのステージにおいてグループ化された命令を 1 命令として扱う．4 命令発行のプロセッサにおいて 2 命令をグループ化した場合のイメージ図を図 2 に示す．ペイロードエリアに 2 命令がグループ化して格納され，グループ化した命令を 1 命令としてウェイクアップ，セレクト，そして発行することが，実質 2 命令をウェイクアップ，セレクト，そして発行することになる．つまり，命令数に対して必要とされるポート数が半減できることになり，命令キューの複雑さは低減され，大幅な消費電力削減につながると考えられる．

しかしながら，従来の動的命令スケジューリング機構と同等のスループットを確保するためには，できる限り多くの命令をグループ化する必要がある．例えば，4 命令を同時に発行可能なプロセッサにおいて全くグループ化することができなかった場合，最大で 2 命令しか同一サイクルに発行することができなくなる．どのような命令をグループ化するか，およびグループ化された命令を 1 つの発行単位として扱うためのマイクロアーキテクチャの拡張について次節において述べる．

3.2 命令のグループ化

本手法では，一方の命令発行の次サイクルに確実に他方の命令が発行可能となる命令の組をグループ化する．したがって，2 命令を発行するためには先に発行される方の命令のみをウェイクアップ・セレクトし，この命令が発行された 1 サイクル後に

他方の命令を発行すればよい．グループ化する2命令を検出するためのハードウェアを簡素にするため，先行する命令には単一サイクルの実行レイテンシを持つ整数演算命令を対象とする．

3.2.1 グループ化可能な命令の条件

本手法においてグループ化可能な命令の組は以下の2つに大別される．

[1] 一方の発行が他方を発行する唯一のトリガー

[2] 同一サイクルに実行が可能

まず，[1] について説明する．下記の2命令において，命令2は右オペランドがレディであり，左オペランド r5 は命令1のデスティネーションとなっている．

$$\begin{cases} \text{命令 } 1 : add\ r5 \leftarrow r3, r2 \\ \text{命令 } 2 : add\ r4 \leftarrow r5, R \end{cases} \quad (1)$$

つまり，命令1が発行されるというただ一つの条件によって依存は解消され，命令2は次サイクルに発行可能となるため，この2命令はグループ化可能といえる．

同様に，以下の2命令も上記と同じく [1] の場合に相当する．

$$\begin{cases} \text{命令 } 1 : add\ r5 \leftarrow r3, r2 \\ \text{命令 } 2 : add\ r4 \leftarrow r5, r3 \end{cases} \quad (2)$$

命令2の左オペランドは命令1のデスティネーションで，前の例と同じである．また，命令2の右オペランド r3 は命令1の左オペランドでもあり，命令1が発行されるということは r3 はレディであるということを意味する．つまり，上記の2命令もグループ化が可能であると言える．

次に [2] について説明する．以下に示す2命令はどちらも両オペランドが揃っている，レディな命令である．

$$\begin{cases} \text{命令 } 1 : add\ r1 \leftarrow R, R \\ \text{命令 } 2 : add\ r2 \leftarrow R, R \end{cases} \quad (3)$$

この2命令には直接の依存関係はないが，どちらもすでにレディな状態にあるため，グループ化することが可能である．この場合，命令オーダーの若い命令が，先に発行される命令として格納される．どちらもレディな2命令をグループ化しなかった場合，同じサイクルに両命令を実行できることもあり，グループ化することによって性能的なペナルティが発生する場合もあると考えられる．しかし，本手法においては命令をなるべく多くグループ化し，スループットを確保することによって性能低下を防ぐ必要がある．また，レディな命令はキューにとどまっているサイクル数が短く次から次へと捌けていくためクリティカルな命令であることは少なく，実行が1サイクル遅れてもほとんど性能には影響がないと考えられる．上記のような理由から，本手法においてはこういったレディな2命令もグループ化の対象とする．

3.2.2 ディスパッチステージにおけるグループ化

十分なスループットを確保するためには3.2.1で述べた (1)-(3) の，3種類のグループ化対象となる命令の組をディスパッチステージにおいて発見し，なるべく多くの命令をグループ化する必要がある．そこで，まず同じサイクルにディスパッチされる命令 の中からグループ化可能な命令の組を探索する．アウト・オブ・オーダー，スーパースカラプロセッサは命令をディスパッチする前に，偽の依存性を解消するためのレジスタ・リネーミングを行う．その際，同じサイクルにディスパッチする命令についても各々の依存関係をチェックしリネーミングを行う必要がある．これは簡単な組み合わせ回路で実現されており，この回路に若干の改良を加えることで，3.2.1で説明したグループ化可能な命令の組を発見し，グループ化して命令キューに格納することが可能だと考えられる．

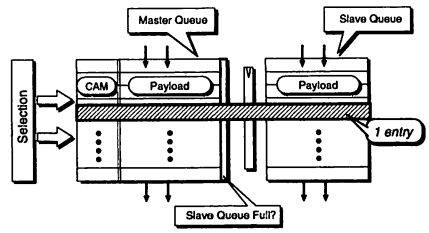

図3 提案する動的命令スケジューリング機構

さらに，より積極的にグループ化を行うために，ディスパッチする命令をすでに命令キューに格納されている命令とグループ化することを考える．以下では，そのための検出を行う手法について説明する．上記で述べた積極的なグループ化については，簡単なハードウェアで実装可能なように，3.2.1で挙げた例の内，(1) に当てはまるパターンを対象とする．つまり，(1) の命令2をディスパッチする際，命令キューに格納されている命令1を発見することが目的となる．プロセッサはディスパッチ時に物理レジスタの状態を知るためにステートテーブルにアクセスするが，このステートテーブルに改良を加えることで上記を実現する．一般的にレジスタは最低でも4状態あるため，レジスタの状態を示すフィールドが2ビットある．新たに，そのレジスタのプロデューサ命令の格納されている命令キューのエントリ番号が書き込まれるフィールドを追加する．以降，このフィールドをステートテーブルのキューフィールドと呼ぶ．新たに付け加えられたフィールドのビット幅は $log_2(size\ of\ IQ)$ である．グループ化対象の命令をディスパッチする際にはまずステートテーブルでソースレジスタの状態を確認する．この際，片方がレディで，もう一方のキューフィールドに値が書き込まれていたら，そのプロデューサとグループ化可能であることがわかる．また，整数演算命令をディスパッチする際にはソースレジスタの状態を確認するだけでなく，デスティネーションレジスタのキューフィールドにその命令が格納されるエントリ番号を書き込む．以上の改良によって，命令キューに格納されている命令とのグループ化も可能となる．

3.3 提案する動的命令スケジューリング機構

3.3.1 基本実装

2命令を1命令としてグループ化する場合における，提案手法の動的命令スケジューリング機構の概要を図3に示す．グループ化される2命令はそれぞれが図に示すマスターキュー (Master Queue) とスレーブキュー (Slave Queue) の同じエントリ番号のエントリに書き込まれ，以後，1エントリとして扱われる．マスターキューには対応するスレーブキューのエントリに命令が格納されているかどうかを示す1ビットのフラグが

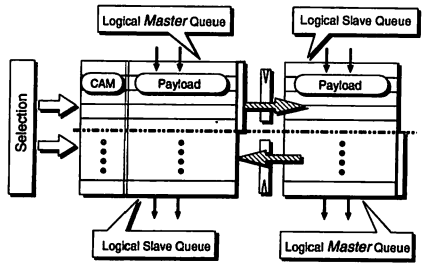

図 4　提案する動的命令スケジューリング機構の改良版

あり，スレーブキューに命令を書き込む際，同時にこのフラグを立てる．1 サイクル内に各々最大でマスター・スレーブキューはそれぞれ，ディスパッチ時に 2 命令ずつ書き込みを行うことができ，2 命令を発行可能である．また，セレクトロジックは 2 命令をマスターキューのエントリからセレクトする回路を備えている．マスターキューとスレーブキューの間にはラッチが設けられており，マスターキュー内の命令がセレクトされ，発行のためにアクセスされる際，スレーブキューに命令が格納されていることを示すフラグが立っていたら，ラッチを介して次サイクルにスレーブキュー内の命令がアクセスされ，発行される．したがって，提案する動的命令スケジューリング機構において，ウェイクアップ時に連想マッチを行うための CAM ロジックはマスターキューのみが有している．

図 3 に示す実装によって提案手法を実現することが可能であるが，マスター・スレーブキューそれぞれの書き込みポート数が 2 であるためにディスパッチの自由度が低くなっている．例えば，マスターキューに 3 命令，スレーブキューに 1 命令といった書き込みを行うことが出来ない．結果的にフロントエンドの周波数を低下させた場合と同様の振る舞いを示すことになる．フロントエンドは性能的にクリティカルであることがよく知られており，本実装では性能の低下が大きくなることが予想される．そこで，上記の問題を解決するために，図 3 の動的命令スケジューリング機構を改良する．

3.3.2　命令キューの改良

本節では図 3 に示す動的命令スケジューリング機構が内包する，ディスパッチ時の自由度が低いという根本的な問題を，物理的な改良によって解決する．

図 4 に改良した動的命令スケジューリング機構の概要を示す．図中，キューの中央に引かれている点線を境に，上部は図 3 の実装の様に左側がマスターキュー，右側がスレーブキューとなっているが，下部については逆に，左側がスレーブキュー，右側がマスターキューとなっている．図に示す改良によってディスパッチ時の自由度が増し，例えばマスターキューに 3 命令，スレーブキューに 1 命令を書き込むという要求にも上部左側のマスターキューに 2 命令，下部右側のマスターキューに 1 命令そして下部左側のスレーブキューに 1 命令を書き込むことによって応えることが可能となっている．

上記の実装によってディスパッチ時に命令をグループ化し，そのグループを一つの命令発行単位として扱うという提案手法の論理的動作がポート数などの制限によらず，実現可能となる．動的命令スケジューリング機構のサイズが比較的小さい場合には十分な ILP を抽出することができず，そのサイズが性能のボトルネックとなる．このような場合，従来の動的命令スケジューリング機構と比べて高い性能を達成することが期待できる．また，サイズが十分に大きい場合にはほぼ同等の性能を達成することが可能であると考えられる．消費電力に関しては，動的命令スケジューリング機構のサイズに関わらず，複雑さが低減したことによる大幅な削減が期待できる．

表 1　Processor Configuration

Fetch & Decode width	4
Branch prediction	Combined bimodal (4K-entry) gshare (4K-entry), selector(4K-entry)
BTB	1024 sets, 4-way
Mis-prediction penalty	3 cycles
Instruction queue size - floating-point	32
Issue width - integer - load/store - floating-point	4 2 2
Reorder buffer size	96
Commit width	4
L1 I-cache	32 KB, 32 B line, 2-way 1-cycle latency
L1 D-cache	32 KB, 32 B line, 2-way 2-cycle latency
L2 unified cache	512 KB, 64 B line, 8-way 10-cycle latency
Memory latency	100 cycles
Bus width	16 B
Bus clock	1/4 of processor core

4.　評　　価

4.1　評価環境

本稿において提案している動的命令スケジューリング機構による性能と消費電力への影響を調べるため，SimpleScalar Tool Set [11] を用いたシミュレーションにより評価を行う．なお，図 3 と図 4 に示す動的命令スケジューリング機構の評価を行うため，SimpleScalar のマイクロアーキテクチャに変更を加えている．また，消費電力の評価には，Wattch [12] を用いる．

評価プログラムは，SPEC CPU2000 [13] の整数ベンチマーク全て (*ref* インプットセット) および，MediaBench ベンチマーク群から mpeg2 エンコードのプログラムを用いる．SPEC CPU2000 についてはプログラムの最初の 2 億命令を fast-forward し，200 万命令をシミュレーションした．

4.2　評価の仮定

表 1 に，本評価におけるプロセッサの仮定を示す．また，命令キューは Alpha 21264 [6] に搭載されている，整数命令とロード・ストア命令がディスパッチされるものを仮定している．したがって，先行する整数演算命令とそれに依存のあるロード命令の組み合わせもグループ化することが可能である．評価では，命令キューのサイズを変化させ，従来の動的命令スケジューリング機構と比較する．

また，グループ化対象となる命令を検出するためのハードウェアの消費電力については無視できるものとし，評価には加えていない．

5.　評 価 結 果

5.1　性　　能

まず，従来型の動的命令スケジューリング機構および提案手法

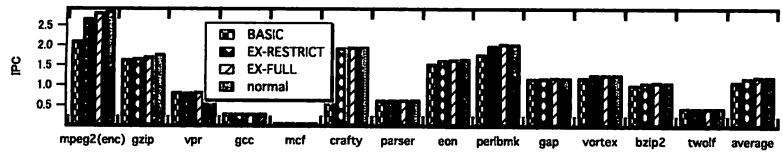

図 6　プログラム毎の IPC (命令キューのエントリサイズ: 48)

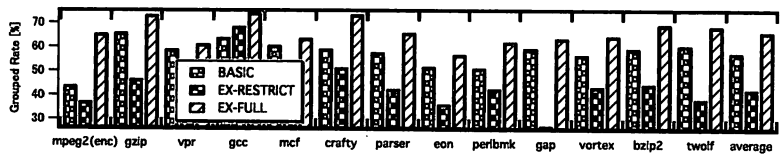

図 7　グループ化された命令の割合 (命令キューのエントリサイズ: 48)

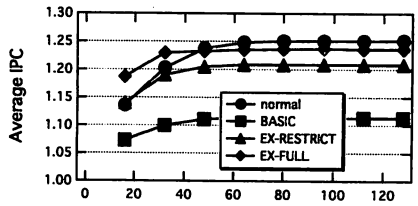

図 5　IPC (全プログラムの平均)

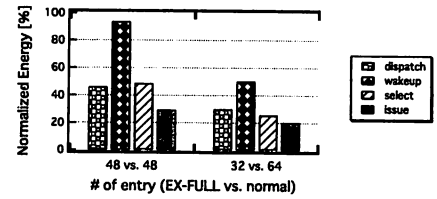

図 9　EX-FULL における消費電力削減率

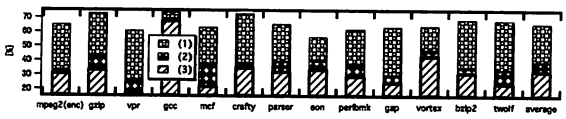

図 8　EX-FULL におけるグループ化された命令の内訳 (命令キューのエントリサイズ: 48)

において，命令キューのエントリサイズが性能に与える影響を調べるため，図 5 に命令キューのエントリサイズを変化させた場合の，評価に用いた全プログラムの平均 IPC を示す．図中，normal は従来型における結果を，BASIC は 3.3.1 に示す図 3 の基本的な実装における結果を示している．また，EX-RESTRICT と EX-FULL はどちらも図 4 に示す改良された実装における評価結果である．この両者の違いは，EX-RESTRICT はグループ化対象となる命令を同じサイクルにディスパッチされる命令の中のみから選択するのに対して，EX-FULL はそれに加えて 3.2.1 の例 (1) に該当する命令の組も対象とするという点である．BASIC も，EX-FULL と同様に 3.2.1 の例 (1) に当てはまる組もグループ化の対象としている．ここで，評価結果における命令キューのエントリサイズが従来型と提案手法で等しい場合，従来型の命令キューは 1 エントリに 1 命令を格納するが，提案手法では図 3 に示すように，1 エントリに 2 命令を格納する違いがあることに注意しなければならない．

図 5 より，全ての場合で，命令キューのエントリサイズが増加するにつれて IPC が向上しているのがわかる．これは，命令キューの保持できる命令数が増えるため，より ILP を抽出することができるためである．

エントリサイズが 64 以上の場合，BASIC，EX-RESTRICT，EX-FULL の normal に対する性能の低下率はそれぞれ 11.1%，3.5%，1.1%である．また一方，EX-FULL においてはエントリサイズが 16，32 と比較的小さめの場合，normal に対して高い性能を達成している．

以下，4 命令同時発行可能なスーパースカラプロセッサにとって命令キューのエントリサイズとして現実的な値である 48 エントリの場合について議論する．図 5 より，EX-FULL は normal とほぼ変わらない性能を達成していることがわかる．また，図 6 に，評価に用いた全プログラムにおける IPC を，図 7 にグループ化された命令の割合を示す．例えば，*mpeg2*, *gzip*, *crafty* などのプログラムにおいて BASIC と EX-RESTRICT は normal に対して性能が大幅に低下しているが，EX-FULL においてはほとんど性能低下がみられない．この理由は，図 7 に示すように EX-FULL は他の 2 つの場合と比べて十分な命令数をグループ化し，スループットを確保できたからと考えられる．

図 8 に，EX-FULL においてグループ化された命令の内訳を示す．図中，(1)-(3) はそれぞれ 3.2.1 で述べたグループ化対象となる命令の条件に相当する．この図から，*gzip*, *gcc*, *crafty*, *parser*, *eon*, *vortex*, *bzip2* においては 30%以上の命令がグループ化されたことがわかる．また一方，*mpeg2*, *vpr*, *crafty*, *gap*, *bzip2*, *twolf* のプログラムにおいては (1) と (2) の条件に相当する命令が 30%以上グループ化されている．(1), (2), (3) の条件に相当する命令がグループ化された割合の全プログラムにおける平均値はそれぞれ 20.6%，6.3%，32.7%である．このように，これら 3 種類の条件を適用することによって様々な特徴を持つプログラムにおいても多くの命令をグループ化でき，十分なスループットを得ることができたといえる．

5.2 消費電力

提案手法は，2 命令をグループ化し，動的命令スケジューリング機構内において 1 つの発行単位として扱うことによって，サイズ／ポート数を削減し，命令キューの消費電力を削減する．図 9 は EX-FULL において従来のプロセッサに対する，動

的命令スケジューリング機構の消費電力削減率を示したものである．それぞれの棒グラフは左からディスパッチ・ウェイクアップ・セレクト・発行の各ステージにおける削減率を示している．図中左側の 48 vs. 48 は命令キューのエントリ数 48 の normal に対するエントリ数 48 の EX-FULL の比較，右側の 32 vs. 64 はエントリ数 64 の normal に対するエントリ数 32 の EX-FULL (64 命令保持可能) の比較である．48 エントリ同士の比較から，EX-FULL は従来の動的命令スケジューリング機構に対してディスパッチ・ウェイクアップ・セレクト・発行の各ステージにおいてそれぞれ，約 54%，7%，52%，71%の消費電力を削減したことがわかる．ウェイクアップステージを除く 3 ステージにおいて大幅な電力削減を達成している．ウェイクアップステージの消費電力は，ブロードキャストされてきたタグとの連想マッチを行うための CAM の高さ (エントリ数) に支配されており，どちらもエントリ数が 48 のためほぼ同じ結果となっている．上記の評価結果から同じエントリ数同士の比較ではウェイクアップステージの消費電力はほぼ等しいことがわかる．しかし，動的命令スケジューリング機構の消費電力において，ウェイクアップステージの消費電力が支配的な構成の場合は，EX-FULL においてエントリ数を減らすことでその消費電力を削減することが可能である．エントリ数 64 の normal の性能とエントリ数 32 の EX-FULL を比較すると，EX-FULL が約 1.5%normal を下回っている．また，そのとき EX-FULL の normal に対する消費電力削減率はディスパッチ・ウェイクアップ・セレクト・発行の各ステージにおいてそれぞれ約 70%，50%，74%，80%となっており，同じエントリ数同士の比較よりもさらに大幅な消費電力削減を性能の低下率約 1.5%で達成している．

6. 関連研究

従来より，動的命令スケジューリング機構の大容量・多ポート化による複雑さや消費電力の増大の問題への対処を目的とした手法が多く提案されている．

文献 [3] では，命令キューにとどまっている命令の消費電力を考慮し，例えばキャッシュミスなどによるレイテンシの長い命令については，巨大な命令待ちバッファ (Waiting Instruction Buffer: WIB) に待避させる命令キューの設計を提案している．また，文献 [4] では，命令キューをサイズの小さなセグメントに分割し，パイプライン的に動作させることによって大幅な性能向上を達成するアーキテクチャを提案している．

また，本研究のように複数命令をグループ化するという視点から，命令を Macro-op と呼ばれる単位にグループ化し，ウェイクアップとセレクトのステージをパイプライン化することにより動的命令スケジューリング機構の複雑度を低減する研究が行われている [14]．命令をグループ化するという視点は同じであるが，実行レイテンシが単一サイクルである命令をなくすことによって依存のある命令のバックツーバックな実行を可能にすることを目的としている点が本研究とは異なっている．

本稿で提案する命令グループ化による動的命令スケジューリング機構の消費電力削減手法は，動的にグループ化可能な命令を検出することで，たとえば汎用プロセッサのように，さまざまな特徴を持つプログラムを実行するような場合にも対応できることも利点として挙げられる．

7. まとめと今後の課題

本稿では動的命令スケジューリング機構の消費電力削減を目的とした，命令グループ化による動的命令スケジューリング機構の消費電力削減手法を提案した．提案手法はグループ化した命令を単一の発行単位として扱うことによって，動的命令スケジューリング機構の複雑さを低減するものである．

提案手法を評価した結果，従来の動的命令スケジューリング機構を有するプロセッサと比較して，エントリ数が小さいときには高い性能を達成しつつ，また十分に大きいエントリ数に対してもほとんど性能を低下させることなく動的命令スケジューリング機構の消費電力を大幅に削減可能であることがわかった．今後，動的命令スケジューリングを行うステージだけでなく，他のパイプラインステージにおいても命令のグループ化を適用する方法を検討していく予定である．

謝辞 本研究の一部は，(株) 半導体理工学研究センターとの共同研究によるものである．

文　献

[1] P. Michaud and A. Seznec: "Data-flow prescheduling for large instruction windows in out-of-order processors.", HPCA, pp. 27–36 (2001).

[2] R. Canal and A. González: "Reducing the complexity of the issue logic.", ICS, pp. 312–320 (2001).

[3] A. R. Lebeck, T. Li, E. Rotenberg, J. Koppanalil and J. Patwardhan: "A large, fast instruction window for tolerating cache misses.", ISCA, pp. 59–70 (2002).

[4] S. E. Raasch, N. L. Binkert and S. K. Reinhardt: "A scalable instruction queue design using dependence chains.", ISCA, pp. 318– (2002).

[5] D. Folegnani and A. González: "Energy-effective issue logic.", ISCA, pp. 230–239 (2001).

[6] S. A. Taylor, M. Quinn, D. Brown, N. Dohm, S. Hildebrandt, J. Huggins and C. Ramey: "Functional verification of a multiple-issue, out-of-order, superscalar alpha processor - the dec alpha 21264 microprocessor.", DAC, pp. 638–643 (1998).

[7] M. Goshima, K. Nishino, T. Kitamura, Y. Nakashima, S. Tomita and S. ichiro Mori: "A high-speed dynamic instruction scheduling scheme for superscalar processors.", MICRO, pp. 225–236 (2001).

[8] A. Buyuktosunoglu, T. Karkhanis, D. H. Albonesi and P. Bose: "Energy efficient co-adaptive instruction fetch and issue.", ISCA, pp. 147–156 (2003).

[9] I. Kim and M. H. Lipasti: "Half-price architecture.", ISCA, pp. 28–38 (2003).

[10] J. J. Sharkey, D. V. Ponomarev, K. Ghose and O. Ergin: "Instruction packing: reducing power and delay of the dynamic scheduling logic.", ISLPED, pp. 30–35 (2005).

[11] T. M. Austin, E. Larson and D. Ernst: "Simplescalar: An infrastructure for computer system modeling.", IEEE Computer, **35**, 2, pp. 59–67 (2002).

[12] D. Brooks, V. Tiwari and M. Martonosi: "Wattch: a framework for architectural-level power analysis and optimizations.", ISCA, pp. 83–94 (2000).

[13] The Standard Performance Evaluation Corporation (SPEC). http://www.specbench.org.

[14] I. Kim and M. H. Lipasti: "Macro-op scheduling: Relaxing scheduling loop constraints.", MICRO, pp. 277–290 (2003).