名古屋大学大学院工学研究科博士前期課程
修士学位論文

ソース・タグ値に基づくセグメント化による発行キューの電力削減

令和　3年 3月
情報・通信工学専攻

森　健一郎

概要

発行キューは電力密度の大きいホット・スポットとして知られている．ホット・スポットは，デバイスの摩耗故障を引き起こし，誤動作やタイミング・エラーを引き起こす．発行キューが大きな電力を消費する原因は，ウェイクアップ論理のタグ比較回路にある．この回路は CAM で構成されており，全てのデスティネーション・タグと発行キュー内の全てのソース・タグとの多数の比較を一斉に行うため，非常に大きな電力を消費する．そこで本論文では，大容量 CAM の研究分野で提案されている手法を応用し，タグ比較による消費電力を削減する手法を提案する．本手法では，発行キューを複数のセグメントに分割する．命令は，ソース・タグの下位ビットがセグメント番号と一致するセグメントにディスパッチする．そして，ウェイクアップ時には，デスティネーション・タグの下位ビットが一致するセグメントにあるタグ比較器のみを動作させる．一致しないセグメントの比較器は動作しないため，タグ比較器の動作回数を削減できる．

本手法では，命令がディスパッチされるセグメントに空きがない場合，他のセグメントに空きがあってもディスパッチできないためストールする．この結果，発行キューの容量効率が低下するという問題が生じる．この問題は，発行キューの容量効率が重要なプログラムにおいて性能低下を引き起こす．そこで本論文では，容量効率を重視したディスパッチ・アルゴリズムと，タグ比較の積極的な削減を重視したディスパッチ・アルゴリズムを動的に切り替える手法を提案する．本手法は，発行キューの容量効率が重要な場合は容量効率の低下による性能低下を抑制し，そうでない場合は積極的にタグ比較器の動作回数を削減することを可能とする．提案手法を SPEC CPU 2017 を用いて評価を行った．結果，性能低下を最大で 5% 以下に抑えつつ，タグ比較器の動作回数を平均で 82% 削減できることを確認した．

目 次

第1章　はじめに

現在のプロセッサは，非常に微細な LSI 技術で製造される．このような LSI の微細化に伴い，デバイスの信頼性低下の問題が深刻になっている [1]．微細化は，経年劣化や摩耗故障を加速し，その結果，タイミング・エラーや誤動作を引き起こし，デバイスの寿命を縮める．経年劣化や摩耗故障は温度に関して指数関数的に加速し [2, 3, 4]，温度 10〜15 ℃の上昇でデバイスの寿命は半分以下になる [5]．

プロセッサ・チップ上には，ホット・スポットと呼ばれる単位面積あたりの電力が大きい場所が存在する．ホット・スポットは，そうでない場所と比べて温度上昇が激しいため，上述した故障を引き起こす確率が高くなる．従って，ホット・スポットを生成する回路の消費電力を低下させる必要がある．

ホット・スポットを生成する回路の 1 つに，発行キューがある．発行キューのサイズはプロセッサの世代が進むごとに大きくなっており，より深刻なホット・スポットとなっている．従って，発行キューの電力削減に対する要求は非常に大きい．

発行キューの中で最も電力を消費する回路は，タグ比較の回路である．タグ比較は，発行幅分のデスティネーション・タグとすべてのソース・タグとの間で行われるため，非常に多くの電力を消費する．そこで本論文では，タグ比較器が動作する回数を削減する以下のような手法を提案する．

- 発行キューを複数の**セグメント**に分割する．命令を発行キューにディスパッチする際，第 1 ソース・タグの下位ビットが n である命令は，第 n 番目のセグメントに書き込む．タグ比較時には，デスティネーション・タグの下位ビットがセグメント番号と一致するセグメントでのみ，第 1 ソース・タグの比較を行う．一致しないセグメントでは比較が行われない．これによりタグ比較回数が削減される．

- 上記の方法では，第 2 ソース・タグの比較回数は削減されない．そこで提案手法では**スワップ**と**サブ・セグメント**と呼ぶ 2 つの方法を導入し，第 2 ソース・タグの比較回数も削減する．スワップは，ディスパッチ時に第 1 ソース・オペランドがレディで，第 2 ソース・オペランドがレディでない命令において，第 1 ソース・タグと第 2 ソース・タグを格納するフィールドを交換し，第 2 ソース・タグの下位ビットを用いてディスパッチするセグメントを決定する手法である．サブ・セグメントは，各セグメントを第 2 ソース・タグに基づきさらに分割する手法である．

- セグメント化によりディスパッチできるエントリが制限されるため，発行キューの容量効率が低下し，容量に敏感なプログラムにおいて性能が低下するという問題が存在する．この問題に対応するため，本論文では **SWITCH** という手法を提案する．SWITCH では，容量効率を重視したディスパッチ・アルゴリズムと，タグ比較回数の削減を重視したディスパッチ・アルゴリズムを，容量効率の重要性に応じて切り替えて使用することにより，性能低下を抑制する．

提案手法を SPEC CPU 2017 ベンチマークを用いて評価し，性能低下を 最大でも 5% 以下に抑えつつ，タグ比較の回数を平均で 82% 削減できることを確認した．

本論文の残りの構成は次の通りである．まず，第 2 章で関連研究を説明し， 第 3 章で発行キューの基本的な事項を説明する．そして，第 4 章で提案手法の基本となるアイデアに関して説明したあと，第 5 章で提案手法の問題点である発行キューの容量効率の低下とその対策方法を説明する．第 6 章で評価を行い，第 7 章でまとめる．

第2章　関連研究

本章では，発行キュー（IQ：Issue Queue) に関連する研究について述べる．2.1 節で IQ に関する一般的な関連研究に関して説明し，2.2 節 で IQ の研究のうち，電力に関係する研究を述べる．

2.1　IQ に関する関連研究

Palacharla らは，命令発行幅と IQ のサイズを変化させた時の，ウェイクアップ論理と選択論理の遅延を評価した [6]．また，遅延を小さくするために，IQ を複数の FIFO バッファで構成し，依存する命令を同じ FIFO バッファに割り当てる依存ベースの IQ を提案した．この手法では，各バッファの先頭の命令のみ発行可能かチェックすれば良いので，回路が単純化され遅延が減少する．

Stark らは，IPC をほとんど低下させずに，ウェイクアップ論理と選択論理をパイプライン化する手法を提案した [7]．この手法では，投機的にウェイクアップを行うことで，依存する命令を連続するサイクルで発行できるようにした．

五島らは，ウェイクアップ論理を従来の 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 [8]．これによって比較器を用いずに依存する命令をウェイクアップすることが可能で，ウェイクアップの遅延を短縮できる．

Sassone らは，依存行列の遅延と電力をより小さくするための手法を提案した [9]．具体的には，従来はすべての命令について，その古さを完全に追跡していたのに対して，命令をグループ化してグループ単位で古いものを選択する．これにより，性能低下を最小限に抑えながら，回路の規模を小さくできる．

Lebeck らは．キャッシュ・ミスするロードのような長いレイテンシの命令に依存する命

令を，IQ とは別の待機用バッファに入れ，その長いレイテンシの処理が完了するまで IQ に挿入しないという方式を提案した [10]．これによって，IQ が待機する命令で埋ることによって起こるストールの頻度が減り，性能が向上する．

Raasch らは，IQ をいくつかのセグメントに分割する方式を提案した [11]．この方式では，各命令の依存命令チェーンのレイテンシを元に割り当てるセグメントが決定される．そして，発行可能になる直前に最下位セグメントである発行バッファに命令を移動する．この発行バッファでのみ発行を行うことで，すべてのエントリから発行できる通常の IQ と比較して遅延を短縮できる．

Kim らは，レイテンシが互いに 1 サイクルの依存関係のある 2 つの命令をグループ化し，1 つの命令として IQ のエントリでスケジューリングすることで，依存グラフのエッジのレイテンシ短縮とキューの容量効率を上げる手法を提案した [12]．

Gibson らは，依存する命令をポインタでつなぎ，ポインタをたどることでウェイクアップを行う手法を提案した [13]．この方式により CAM が不要になり，電力を削減できる．

安藤は，実行プログラムの命令レベル並列性（ILP：instruction-level parallelism）とメモリ・レベル並列性（MLP：memory-level parallelism）に応じて IQ の方式を切り替える手法を提案した [14]．ILP と MLP のいずれかが高い場合は IQ の容量効率が重要であるため，IQ をランダム・キューに構成する．ILP と MLP のどちらも低い場合には，容量効率よりも正しい発行優先度が重要であるため IQ をサーキュラー・キューに構成する．

甲良らは，実行プログラムの ILP と MLP に応じて IQ のサイズを変化させる手法を提案した [15]．本手法では，MLP が高い場合には，IQ の容量が重要となるため IQ のエントリ数を増加しパイプライン化する．MLP が低い場合には IQ のエントリ数を減少させ，パイプライン化を解除する．

2.2 IQ の電力削減に関する関連研究

Folegnani らは，空のエントリの比較器や既にレディなオペランドを持つ比較器など，タグを比較する必要がない比較器を動作させないことで，消費エネルギーを削減する手法を提案した [16].

Ponomarev らは，リソース要求に応じて IQ のサイズをリサイズすることにより，消費エネルギーを削減する手法を提案した [17] .

Ernst らは，IQ に入ってくる命令のうちのほとんどが，はじめから少なくとも 1 つのソース・オペランドがレディであると指摘した [18]．そして IQ に，2 つのソース・オペランドを保持できるエントリに加えて，1 つのソース・オペランドのみ保持できるエントリと，ソース・オペランドを保持しないエントリを用意し，レディでないソース・オペランドの数に応じていずれかにディスパッチする手法を提案した．さらにこの手法を実現するために，命令の 2 つのオペランドの内，あとにレディになるオペランドを予測する手法も提案した．

Sembrant らは，クリティカル・パス上にない命令を IQ とは別のバッファに入れ，ディスパッチを遅延させることによって，性能を低下させずに IQ のサイズを小さくする手法を提案した [19].

Homayoun らは，キャッシュ・ミスの処理中に発行幅を半減させることで，IQ の消費電力を削減する手法を提案した [20]．発行幅半減中に元の発行幅の半分以上の命令が発行される場合，一時的にその命令を小さなバッファに移動させることで対応している．

小林，松田らは，ウェイクアップ時のタグ比較を 2 段階に分割することによりエネルギー削減を行う方法を提案した [21, 22]．この方法では，タグの比較を高位ビットと低位ビットに分割し，低位ビットの比較を最初のサイクルで行う．そして低位ビットが一致していた場合のみ，次のサイクルで高位ビットの比較を行う．低位ビットの比較で一致しない場合，高位ビットの比較は行われず，エネルギーを削減することができる．また，タグの 2 段階比較には，ウェイクアップに 2 サイクル必要であるため性能が低下するという欠点が存在する．これに対しこの手法では，クリティカル・パス上にあると推測される命令のみ 1 サ

イクルで比較を行い性能低下を軽減する.

第3章　発行キュー（IQ：Issue Queue）

本章では，本研究の研究対象である，IQ に関して説明する．まず，IQ の概要と動作を 3.1 節で説明したあと，IQ の回路構成を 3.2 節で述べる．その後，3.3 節で IQ の方式に関して説明する．

3.1　概要と動作

IQ はアウト・オブ・オーダ実行のプロセッサにおいて，リネームされた命令を保持し，実行順序をスケジューリングして，機能ユニットへ発行する回路である．IQ は，ディスパッチ，発行，ウェイクアップと呼ばれる 3 種類の動作を行う．以下でそれぞれの動作に関して説明する．

- ディスパッチ：リネームされた命令は，IQ にエントリが割り当てられ，命令の情報が格納される．この動作をディスパッチと呼ぶ．ディスパッチの動作は，IQ の方式により異なる．IQ の方式に関しては，3.3 節で詳しく説明する．

- 発行：IQ 内の命令のうち，ソース・オペランドが両方共レディとなった命令は，依存関係が解消し，実行が可能となる．このような命令を実行ユニットに送出する動作を発行と呼ぶ．なお，発行可能な命令が機能ユニットの数を超える場合は，各命令の発行優先度に基づき命令を選択して発行する．発行された命令のエントリは IQ より削除される．

- ウェイクアップ：命令が発行されると，その命令のデスティネーション・オペランドのタグと IQ 内にある全命令のソース・オペランドのタグの比較が行われる．比較が一致した場合には，対応するソース・オペランドのレディ・ビットをセットする．こ

の動作をウェイクアップと呼ぶ．両方のオペランドがレディとなった命令は，依存が解消したため発行可能となる．

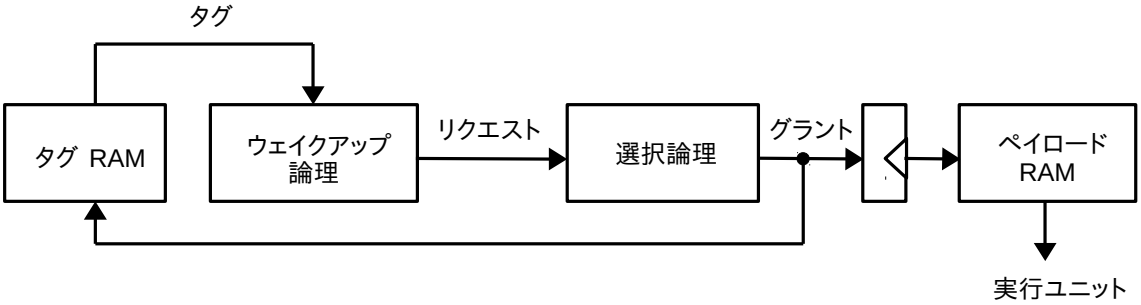

図 3.1: IQ の回路構成

3.2　回路構成

図 3.1 に IQ の回路構成を示す．IQ はウェイクアップ論理，選択論理，タグ RAM，ペイロード RAM と呼ばれる 4 つの回路より構成される．以下で各回路に関して説明する．また，IQ の回路のうちウェイクアップ論理は提案手法に関わる重要な回路であるため，3.2.1 節にて詳細に説明する．

- ウェイクアップ論理 : 命令間の依存関係を管理し，他の命令との依存関係が解消された命令について発行要求（リクエスト信号）を出す．
- 選択論理 : 資源制約を考慮して，発行を要求された命令の中からそれを許可する命令を選択し，発行許可信号（グラント信号）を出力する．この選択においては，回路の単純化のために IQ の先頭のエントリの命令をより優先する．

- タグ RAM : 発行待機中の命令のデスティネーション・タグを保持する回路で，選択論理から発行許可信号が送られると，対応する命令のタグが読み出され，ウェイクアップ論理へ送られる．

- ペイロード RAM : 発行待機中の命令のコードを保持する．選択論理から発行許可信号が送られると，対応する命令のコードを実行ユニットに送出する．

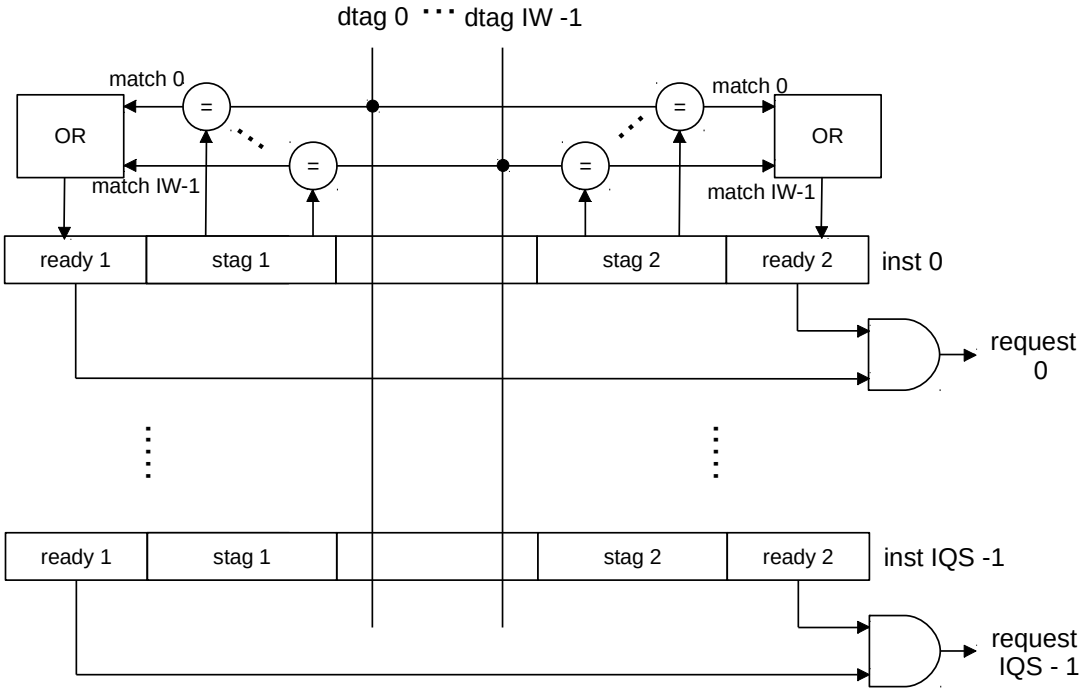

図 3.2: ウェイクアップ論理

3.2.1 ウェイクアップ論理

図 3.2 に，ウェイクアップ論理の回路を示す．図中の IW は発行幅を，IQS は IQ のエントリ数を表す．ウェイクアップ論理では，IW 個のデスティネーション・タグ（dtag）が

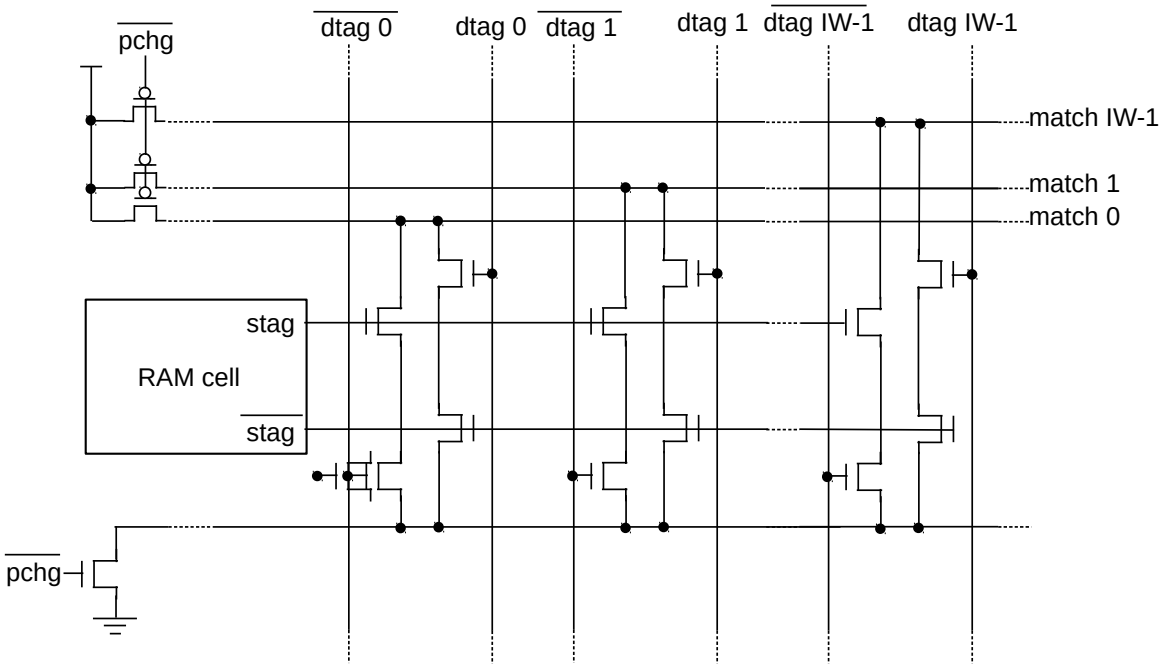

図 3.3: タグ比較器の CAM 回路

IQ 内の全命令に放送される．各命令は 2 つのソース・タグ（stag）を保持しており，放送されたデスティネーション・タグと比較される．いずれかのデスティネーション・タグとソース・タグが一致した場合，そのソース・オペランドのレディ・ビットがセットされる．2 つのレディ・ビットがセットされた命令は発行が可能となるため発行要求が出力される．

図 3.3 に，IQ に使用されるタグ比較器の CAM 回路を示す．同図は，ソース・タグ 1 ビット分の比較回路を表す．同図に示すように，高速化のため通常ダイナミック論理によって構成される [6]．比較の動作は，次のように行われる．まず，マッチ線がプリチャージされる．次にデスティネーション・タグが放送され，比較が行われる．タグが不一致であれば，直列に接続された 2 つのプルダウン・トランジスタが両方とも ON となり，マッチ線がディスチャージされる．タグが一致する場合，マッチ線は H の状態が維持される．比較

器はマッチ線のディスチャージ時に電力を消費する．

3.3 IQ の方式

これまで，IQ の方式としてシフト・キュー，サーキュラ・キュー，ランダム・キューの 3 つの方式が商用プロセッサで使用された．各方式に関して説明した後，現在主流の方式であるエイジ論理付きのランダム・キューに関して説明する．

3.3.1 シフト・キュー

シフト・キューは，非常に古い（20 年以上前）商用プロセッサに使用された IQ の方式である [23]．シフト・キューでは，IQ は基本的に FIFO バッファであり，末尾のエントリに命令をディスパッチする．これにより，古い命令に高い発行優先度を与えることができる．[1]

また，シフト・キューで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により，高い容量効率も達成することができる．正しい発行優先度と，高い容量効率を同時に達成するため，シフト・キューは IQ の方式の中で最も高い性能を得ることができる．

一方でシフト・キューには，コンパクションの回路が非常に複雑で，また消費電力が非常に大きいという欠点がある．そのため，シフト・キューはスケーリングが困難であり，現在のプロセッサでは使用されていない．

3.3.2 サーキュラー・キュー

サーキュラー・キューは，シフト・キューと同様，命令をプログラム順に並べるが，コンパクションを行わない方式である [25]．サーキュラー・バッファで実装される．

[1] 一般に，古い命令から優先的に発行すると，性能がより高くなることが知られている [24].

サーキュラー・キューでは，先頭と末尾の間の命令が発行されても，新たに命令をディスパッチできないため，IQ の容量効率がシフト・キューと比較して低下する．また，ヘッド・ポインタとテール・ポインタの位置が逆転するラップ・アラウンドが生じた際には，新しい命令に高い優先度が与えられる優先度逆転が起き，選択論理が正しい優先度で命令を選択できない．これらの理由から，サーキュラー・キューはシフト・キューと比較して性能が低下する．特に，容量効率が低下する影響は大きく，現在のプロセッサでは使用されていない．

3.3.3 ランダム・キュー

近年は，回路の単純化や電力削減のため空いているエントリに単純にディスパッチするランダム・キューが使用されている [26, 27, 28]．ランダム・キューでは IQ の容量を無駄にすることがなく，高い容量効率を達成する．その一方で，命令が年齢とは無関係にランダムに並ぶため，正しい優先度で命令を発行することができない．

ランダム・キューでは，IQ の空きエントリのインデクスを保持するフリー・リストを用意する．ディスパッチ時には，フリー・リストから読み出したインデクスが指す IQ のエントリに命令を書き込む．IQ から命令が発行されエントリが無効化されると，そのインデクスをフリー・リストへ返す．フリー・リストは FIFO バッファで管理される．

3.3.4 エイジ論理付きランダム・キュー

ランダム・キューにおける発行優先度に関する欠点を緩和するため，ランダム・キューは一般にエイジ論理が併用される [26]．エイジ論理は選択論理と並列に動作する回路で，発行要求が出された命令の中で最も古い 1 命令を選ぶ．最も古い命令はクリティカル・パス上の命令である可能性が高いため，これを優先して発行することができ，結果としてエイジ論理付きランダム・キューは通常のランダム・キューと比較して性能が大きく向上する．

本研究における IQ は，エイジ論理付きのランダム・キューを仮定する．

第4章　提案手法：セグメント化した IQ

本論文では，IQ のタグ比較器の動作回数を削減するための手法として，IQ をセグメント化する手法を提案する．本章では，4.1 節で提案手法の基本アイデアに関して説明したあと，4.2 節で提案手法における第 2 ソース・タグ比較の削減方法である，**スワップ**と**サブ・セグメント**に関して説明する．

4.1　IQ のセグメント化

本節では，提案手法の概要を説明した後，提案手法におけるウェイクアップとディスパッチに関して詳しく説明する．

4.1.1　提案手法の概要

提案手法の基本アイデアは，大容量 CAM の電力削減に関する研究 [29, 30] から着想を得ている．この研究において提案されている手法では，CAM を複数の**セグメント** [1]に分割する．各セグメントには下位ビットが同一のデータのみを記録する．そして，タグ比較においては，比較対象のデータの下位ビットと，記録されているデータの下位ビットが一致するセグメントのみで比較を行う．これによって，比較器が動作する回数を「1/セグメント数」まで削減することができ，消費電力が削減できる．

本手法においても，図 4.1 に示すように IQ を複数のセグメントに分割する．各セグメントには，第 1 ソース・タグの下位ビットがセグメントの番号と一致する命令をディスパッチする．ウェイクアップ時の第 1 ソース・タグのタグ比較では，デスティネーション・タグの下位ビットとセグメントの番号が一致するセグメントのみでタグ比較を行う．これによって，第 1 ソース・タグのタグ比較回数を「1/セグメント数」に削減できる．

[1]文献 [29, 30] ではバンクと呼ばれている．

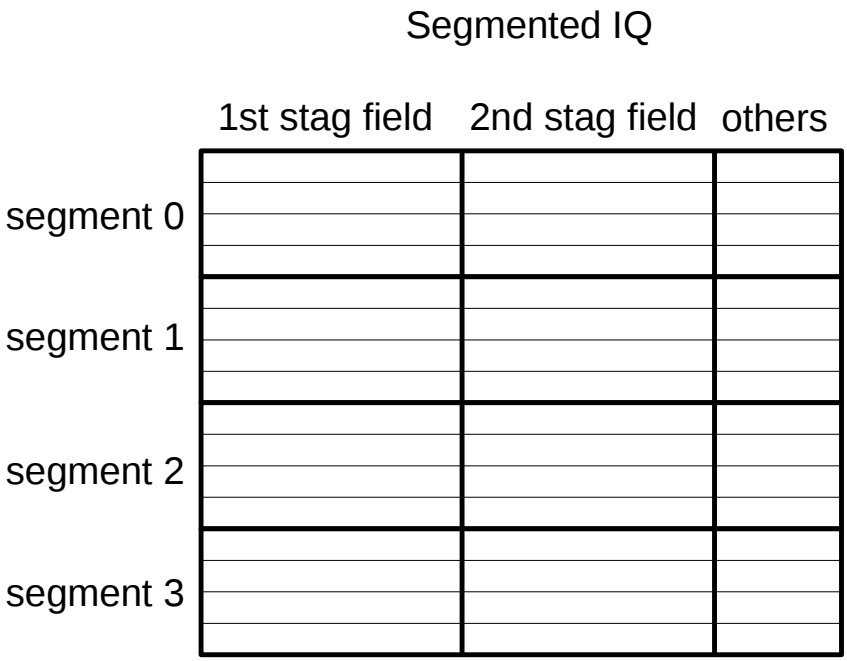

図 4.1: セグメント化した IQ

4.1.2　提案手法におけるディスパッチ

ディスパッチする IQ のエントリを決定する回路を図 4.2 に示す．本手法では，フリー・リストをセグメントと同じ数だけ用意する．各フリー・リストは，対応するセグメントの空きエントリのインデクスを FIFO バッファで管理する．各フリー・リストからは IQ のインデクスが出力され，その中の 1 つを選択してディスパッチするエントリを決定する．どのフリー・リストからの出力を選択するかは，セグメント選択回路（図中の segment select logic）によって決定される．セグメント選択回路は，第 1 ソース・オペランドのタグとレディ・ビットと，各セグメントの空きエントリ数を入力とし，ディスパッチするセグメント番号を出力する．

セグメント選択回路の選択アルゴリズムについて説明する．セグメントの選択方法は，

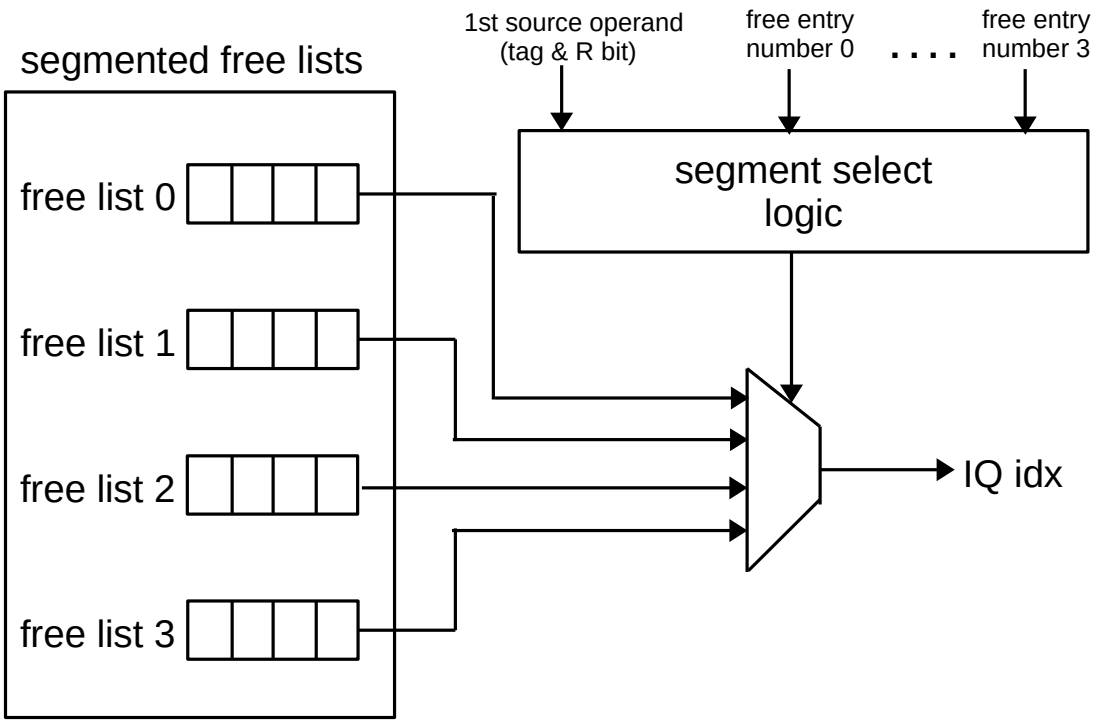

図 4.2: 提案手法におけるディスパッチエントリの決定回路

ディスパッチ時に第 1 ソース・オペランドがレディであるかによって異なるため，それぞれの場合に関して説明する．

- **第 1 ソース・オペランドがレディでない場合**： 第 1 ソース・タグの下位ビットと番号が同じセグメントを選択する．選択されたセグメントに空きエントリがある場合，ディスパッチ可能であるため，対応するフリー・リストから読み出したエントリにディスパッチする．対応するセグメントに空きがない場合は，セグメントに空きが出るまでディスパッチをストールさせる．

- **第 1 ソース・オペランドがレディである場合**：この場合，第 1 ソース・タグの比較は行われないため，どのセグメントにディスパッチしても問題ない．このような場合を**セグメント・インディペンデント**と呼ぶ．この場合，空きエントリのあるセグメン

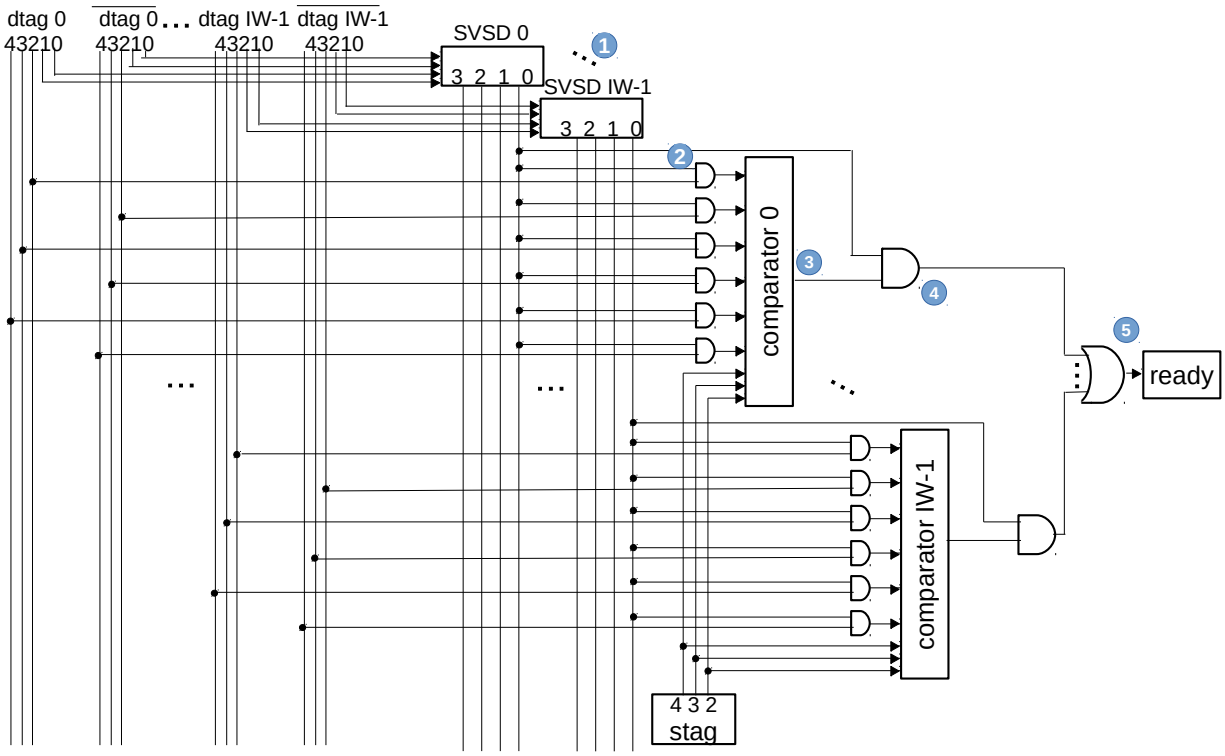

図 4.3: 提案手法におけるタグ比較回路（第 0 セグメント）

トから，ラウンドロビンでディスパッチするセグメントを選択しディスパッチする．

例として，第 1 ソース・オペランドがレディでなく，タグが 15（1111_2）である命令を，図 4.1 に示す 4 つに分割された IQ にディスパッチする場合を考える．第 1 ソース・タグの下位 2 ビットが 3（11_2）であるので，この命令は第 3 セグメントにディスパッチされる．

なお，ソース・オペランドを使用しない命令も存在するが，そのような命令はディスパッチ時にソース・オペランドがレディであるものとして扱う．

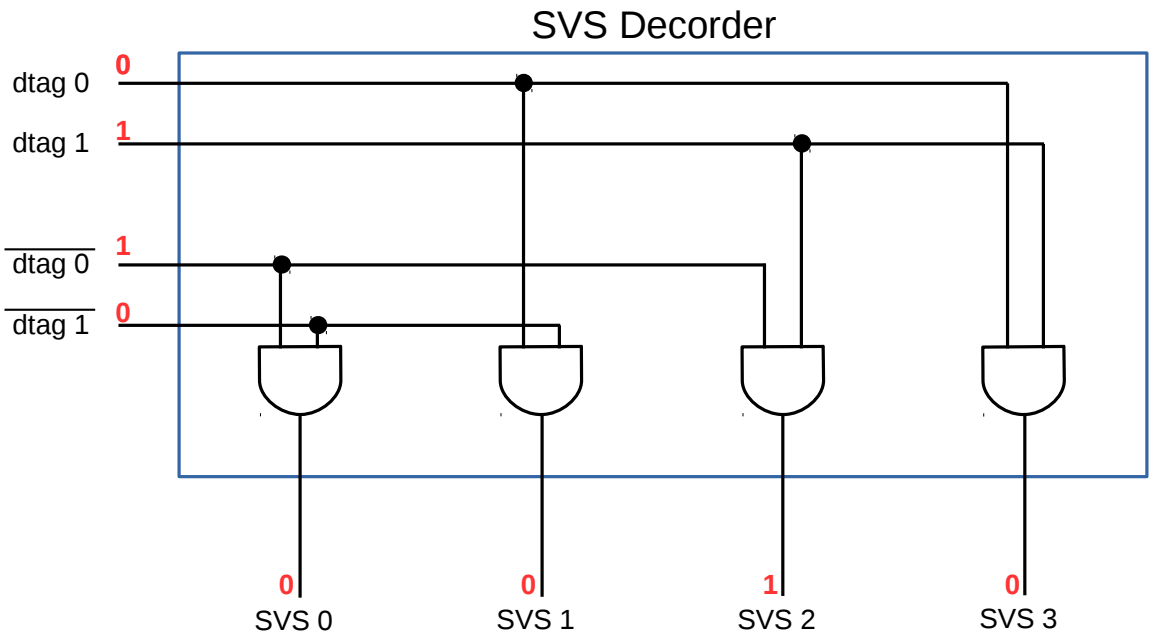

図 4.4: SVSD（Segment-Validation Signal Decoder）

4.1.3 提案手法におけるウェイクアップ

提案手法におけるウェイクアップでは，デスティネーション・タグの下位ビットがセグメント番号と一致するセグメントでのみ，第 1 ソース・タグのタグ比較器を動作させ比較を行う．一致しないセグメントはタグ比較器を動作させない．これは，デスティネーション・タグの下位ビットと番号が一致しないセグメントには，第 1 ソース・タグの下位ビットがデスティネーション・タグの下位ビットと異なる命令しか入っておらず，タグは必ず不一致となるためである．

例として，放送されたデスティネーション・タグが 6（110_2）で，IQ が図 4.1 のように 4 つのセグメントに分割されている場合を考える．この場合，下位ビットは 2（10_2）であるため，第 2 セグメントでのみ，第 1 ソース・タグのタグ比較を行う．

なお，第 2 ソース・タグのタグ比較に関しては，セグメントの番号とタグの下位ビットに関係性はないため，すべてのセグメントでタグ比較を行う必要がある．

提案手法におけるタグ比較の回路を図 4.3 に示す．同図は 4 つのセグメントに分割された IQ のうち，第 0 セグメントのエントリにおける，第 1 ソース・タグの比較回路を示している．タグ・ビット数は 5 とし，発行幅を IW とする．

タグ比較の動作を図中の番号を用いて説明する．①放送されるデスティネーション・タグの下位 2 ビットはデコーダ（SVSD：Segment-Validation Signal Decoder）へ送られる．SVSD はセグメント数だけ信号線を出力する．第 n 番目の信号線は，第 n セグメントでのタグ比較を有効化することを示す．つまり，デスティネーション・タグの下位ビットが n の場合，n 番目の出力線のみ H を出力し，残りはすべて L を出力する．SVSD の出力信号線のことを，以下， SVS（Segment-Validation Signal）と呼ぶ．

②AND ゲートによって，SVS が H の場合にのみ，デスティネーション・タグの高位ビット及びその反転信号がタグ比較器へ入力される．図 4.3 に示す回路は第 0 セグメントのタグ比較回路であるため，0 番目の SVS が AND ゲートに入力されている．

SVS が H の場合，つまり，デスティネーション・タグの下位ビットとセグメント番号が一致していた場合のみ，比較器にデスティネーション・タグの高位ビットとその反転信号が送られ，ソース・タグの高位ビットと比較が行われる．SVS が L の場合，デスティネーション・タグとその反転信号がどちらも L としてタグ比較器へ入力される．この場合，デスティネーション・タグとその反転信号に接続されたプルダウン・トランジスタがすべて OFF となるため，マッチ線はディスチャージされず，電力を消費しない．

③タグ比較の結果，タグの高位ビットが一致した場合は，比較器から H が出力される．

④タグ比較器が H を出力し，かつ SVS が H である場合に，タグ比較は一致となる．

⑤IW 個の比較のいずれかが一致となった場合に，ソース・オペランドのレディ・ビットがセットされる．

SVSD の回路図を図 4.4 に示す．SVSD はデスティネーション・タグの下位ビットのう

ち log_2(セグメント数）ビットの正転及び反転信号を入力とし，セグメント数分の SVS を出力するバイナリ・デコード回路である．SVSD は セグメント数分の AND ゲートで構成される．

図ではセグメント数が 4 の場合の SVSD を表記しており，デスティネーション・タグの下位 2 ビット（$log_2 4$）を入力とし，SVS が 4 本出力される．赤字でタグの下位ビットが 2（01_2）の場合を例示している．この場合，SVS2 のみ H となり，その他の SVS は L を出力する．

4.2 第 2 ソース・タグ比較の削減

4.1 節節で述べた手法では，命令の第 2 ソース・タグのタグ比較回数は削減できない．そこで本節では，第 2 ソース・タグの比較回数の削減を可能とするスワップとサブ・セグメントという 2 つの手法を提案する．

4.2.1 スワップ

スワップは，第 1 ソース・タグと第 2 ソース・タグを格納するフィールドを交換し，第 2 ソース・タグの下位ビットをもとにディスパッチするセグメントを決定する手法である．以下で詳しく説明する．

第 1 ソース・オペランドがレディで，第 2 ソース・オペランドがレディでない場合について説明する．この場合，4.1 節で説明した方法では，命令はセグメント・インディペンデントとしてディスパッチされる．第 1 ソース・オペランドは既にレディであるため，比較は第 2 ソース・タグについてのみ行われるが，第 2 ソース・タグのタグ比較は全てのセグメントで行われるため，タグ比較の回数は削減されない．

そこでこのような場合に，第 1 ソース・タグと第 2 ソース・タグを交換し（スワップ），第 2 ソース・タグの下位ビットを使用してディスパッチするセグメントを選択する．これにより，4.1 節で述べたセグメント化の効果でタグ比較回数が削減される．なお，スワップ

ではタグを交換するが，ペイロード RAM に格納するソース・タグを交換するわけではないので，命令の意味は保持される．

スワップを行う場合のセグメント選択アルゴリズム

セグメント選択回路は，表 4.1 に示すアルゴリズムによってディスパッチするセグメントを決定する．なお，表中のソース・タグの状態とは，ディスパッチ時にソース・オペランドがレディであるかどうかを示しており,（第 1 ソース・オペランド，第 2 ソース・オペランド）の形式で，R がレディであることを，NR がレディでないことを表す．

表 4.1: スワップを行う場合のセグメント選択アルゴリズム

ソース・タグの状態	アルゴリズム
（NR，NR）	第 1 ソース・タグでセグメントを選択する．
（R，NR）	スワップを行い，第 2 ソース・タグでセグメントを選択する．
（NR，R）	第 1 ソース・タグでセグメントを選択する．
（R，R）	セグメント・インディペンデントとしてラウンドロビンでセグメントを選択する．

なお，両ソース・オペランドがレディのとき以外で，選択されたセグメントに空きがない場合は，ディスパッチをストールして当該のセグメントに空きが出るまで待ち合わせる．

4.2.2　サブ・セグメント

サブ・セグメント方式は，第 1 ソース・タグの下位ビットに応じて分割されるセグメントを，第 2 ソース・タグの下位ビットに応じてさらに細かく分割する．第 2 ソース・タグの下位ビットによる分割をサブ・セグメント（S-seg）と呼び，従来の第 1 ソース・タグによる分割をサブ・セグメントに対応してメイン・セグメント（M-seg）と呼ぶこととする．

サブ・セグメントを導入した IQ の分割を図 4.5 に示す．黒色の枠で示す各メイン・セグメントを，赤色と青色で示すようにさらにサブ・セグメントに分割する．同図は，メイン・セグメント数が 4 ，サブ・セグメント数が 2 の場合の例を表しており， IQ は合計 $4 \times 2 = 8$ 個のセグメントに分割される．各セグメントの左には,（M-seg，S-seg）という形式でメイン及びサブ・セグメントの番号を表している．

図 4.5: サブ・セグメントを実装した IQ

サブ・セグメント方式について，ディスパッチとウェイクアップの動作をそれぞれ説明する．

サブ・セグメントにおけるディスパッチ

サブ・セグメント方式におけるディスパッチにおいては，フリー・リストを M-seg × S-seg だけ用意する．図 4.5 に示した例の場合 8 個のフリー・リストが必要となる．

サブ・セグメント方式におけるセグメント選択のアルゴリズムに関して説明する．アルゴリズムはソース・オペランドのレディ状況によって異なるため，以下ですべての場合に関して説明する．説明を簡単にするため，命令 $p5 = p13 + p6$ を，図 4.5 に示す IQ にディスパッチする場合について例示する．第 1 ソース・タグが 13 で，第 2 ソース・タグが 6 である．

- **両ソース・オペランドともレディでない場合**：第 1 ソース・タグでメイン・セグメントを，第 2 ソース・タグでサブ・セグメントを選択する．例の場合，第 1 ソース・タグ 13（1101_2） の下位ビット 1（01_2）より，メイン・セグメントは 1 となる．また，第 2 ソース・タグ 6（110_2） の下位ビット 0（0_2）より，サブ・セグメントは 0 となる．従って（1，0）のセグメントを選択する．

- **第 1 ソース・オペランドのみレディである場合**：第 2 ソース・タグでサブ・セグメントを選択する．例の場合，第 2 ソース・タグ 6（110_2） の下位ビット 0（0_2）より，サブ・セグメントは 0 となる．第 1 ソース・オペランドは既にレディであるため，メイン・セグメントの制限はない．従って，(0，0)，(1，0)，(2，0)，(3，0) のいずれかのセグメントをラウンドロビンで選択する．このように，メイン・セグメントの制限がない場合をメイン・セグメント・インディペンデント（M-seg インディペンデント）と呼ぶこととする．

- **第 2 ソース・オペランドのみレディである場合**：第 1 ソース・タグでメイン・セグメントを選択する．例の場合，第 1 ソース・タグ 13（1101_2） の下位ビット 1（01_2）より，メイン・セグメントは 1 となる．第 2 ソース・オペランドは既にレディであるため，サブ・セグメントの制限はない．従って，(1，0) または（1，1）のいずれかのセグメントをラウンドロビンで選択する．このように，サブ・セグメントの制限がない場合をサブ・セグメント・インディペンデント（S-seg インディペンデント）と呼ぶこととする．

- **両ソース・オペランドがレディである場合**：M-seg インディペンデントかつ S-seg インディペンデントであるため，空きのあるすべてのセグメントからラウンドロビンで選択する．

サブ・セグメントにおけるウェイクアップ

第 1 ソース・タグの比較は，デスティネーション・タグの下位ビットがメイン・セグメント番号と一致するセグメントのみで行う．また，第 2 ソース・タグの比較は，デスティネーション・タグの下位ビットがサブ・セグメント番号と一致するセグメントのみで行う．このような比較により，第 1 ソース・タグだけでなく，第 2 ソース・タグの比較に関しても,「1/サブ・セグメント数」まで削減が可能となる．

サブ・セグメントとスワップの併用

サブ・セグメント方式はスワップと併用することが可能である．併用する場合は，ディスパッチ時に第 1 ソース・オペランドのみレディである場合の選択アルゴリズムを，以下のように変更する．

- **第 1 ソース・オペランドのみレディである場合**：スワップを行い，第 2 ソース・タグでメイン・セグメントを選択する．例の場合，第 2 ソース・タグ 6（110_2） の下位ビット 2（10_2）より，メイン・セグメントは 2 となる．第 1 ソース・オペランドは既にレディであるため，S-seg インディペンデントである．従って,（2，0）または（2，1）のいずれかのセグメントを選択する．

サブ・セグメント方式とスワップを併用することによって，ディスパッチ時に第 1 ソース・オペランドのみレディである命令におけるタグ比較回数の削減が「1/サブ・セグメント数」から「1/メイン・セグメント数」となる．従って，図 4.5 に示した分割のようにメイン・セグメント数がサブ・セグメント数よりも多い場合に，タグ比較回数をより多く削減できる．

サブ・セグメントとスワップを併用する場合のセグメントの選択アルゴリズムを表 4.2 にまとめる．

表 4.2: スワップを行う場合のセグメント選択アルゴリズム（サブ・セグメント併用）

ソース・タグの状態	アルゴリズム
（NR，NR）	第 1 ソース・タグでメイン・セグメントを，第 2 ソース・タグでサブ・セグメントを選択.
（R，NR）	スワップを行い，第 2 ソース・タグでメイン・セグメントを選択する．サブ・セグメントは S-seg インディペンデントとしてセグメントを選択.
（NR，R）	第 1 ソース・タグでメイン・セグメントを選択する．サブ・セグメントは S-seg インディペンデントとしてセグメントを選択.
（R，R）	M-seg インディペンデントかつ S-seg インディペンデントとしてセグメントを選択.

第5章　SWITCH 方式

従来の IQ では，空きエントリがあればディスパッチすることができたが，これまで提案した手法では，ディスパッチできるエントリは選択されたセグメントに限定されている．このため，IQ の容量効率が低下するという問題がある．本章では，この問題に対応する手法である SWITCH 方式に関して説明する．5.1 節で IQ の容量効率の低下に関して説明した後，5.2 節で SWITCH 方式に関して説明する．

5.1　容量効率の低下

提案手法には IQ の容量効率が低下するという問題がある．この問題は，IQ の容量効率が重要なプログラムにおいて，性能低下を引き起こす．本節では，容量効率が低下する原因について説明した後，容量効率の低下により性能低下を引き起こすプログラムの特徴に関して説明する．

5.1.1　提案手法による容量効率低下の原因

IQ の容量効率の低下に関して，図 5.1 を用いて説明する．図において，灰色のエントリは命令を保持していることを示している．

図の状態の IQ に，新たに命令 $p2 = p5 + p6$ をディスパッチする場合を考える．この命令のソース・オペランドは両方レディでないとする．この場合，第 1 ソース・タグの下位ビットから第 1 セグメントにディスパッチされることが決定する．しかし，第 1 セグメントに空きエントリはないため，空きが出るまでディスパッチをストールさせ，待ち合わせを行う必要がある．

このように，命令がディスパッチされるセグメントに空きがない場合，他のセグメント

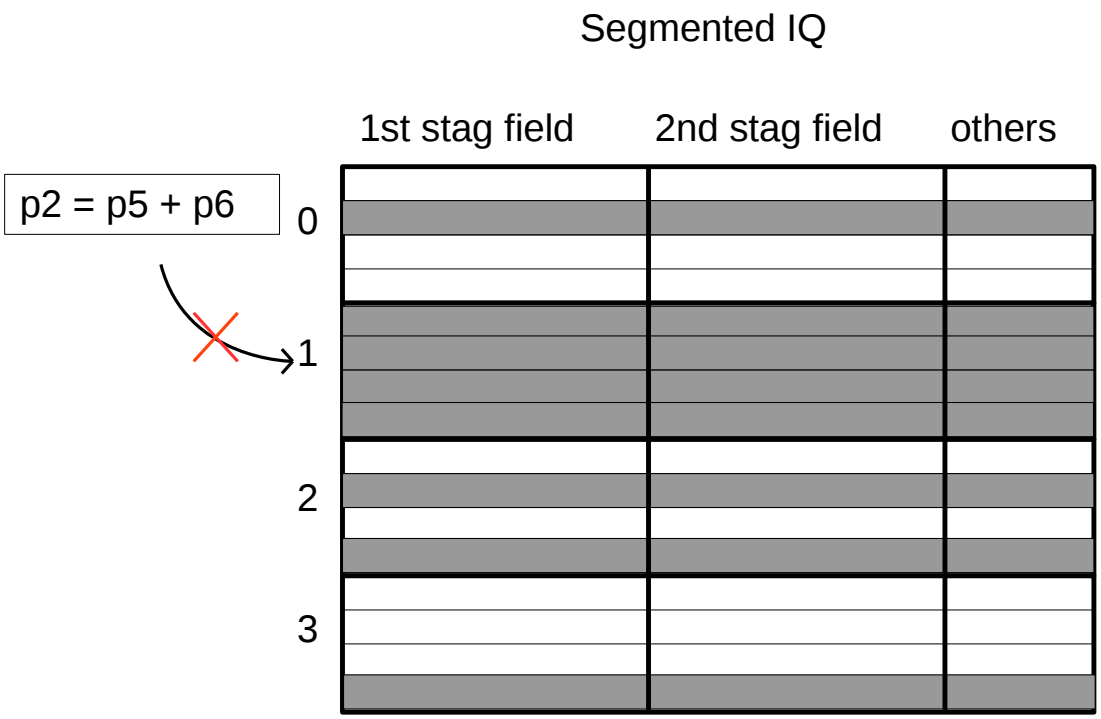

図 5.1: 容量効率が低下する例

に空きがあってもディスパッチできないため，提案手法ではセグメント化されていない IQ と比較して容量効率が低下する．

5.1.2　容量効率の低下による性能低下

プログラムには，性能が IQ の容量に敏感なものとそうでないものとがある [14, 15, 19]. 次の 2 つの特徴のうちいずれかに当てはまるプログラムでは，性能が IQ の容量に敏感なため，与えられた IQ の容量においては，その利用効率が重要である．このため，提案手法による容量効率の低下によって性能が低下する．

- 命令レベル並列性（**ILP：Instruction Level Parallelism**）が高いプログラム
- メモリ・レベル並列性（**MLP：Memory Level Parallelism**）が高いプログラム

ILP が高いプログラムでは，IQ に命令をできるだけ多く供給し，より多くの命令を並列に発行できるようにすることで高い性能が得られる．IQ の容量効率が低下すると，並列に発行できる命令数が減少するため，性能が低下する．

MLP が高いプログラムでは，できるだけ多くのキャッシュ・ミスを並列に実行することにより，メモリ・アクセスのレイテンシが実行時間に与える影響を縮小できる．IQ の容量効率が低下すると，並列に処理できるメモリ・アクセスが減少するため，性能が低下する．

これらのことから，ILP もしくは MLP が高い場合には，提案手法による容量効率の低下を最小限に抑える工夫が必要となる．

5.2 容量効率低下への対策：SWITCH 方式

IQ の容量効率低下による性能低下を抑制する方式として，**SWITCH** と呼ぶ方式を提案する．SWITCH 方式では，次のようにして性能低下を抑制する．

- セグメント選択回路の選択アルゴリズムとして，容量効率は低下するが，タグ比較回数を多く削減できるような選択を行う **AGGRESSIVE モード** と，タグ比較回数の削減率は低下するが，容量効率が大きく低下しないような選択を行う **CONSERVATIVE モード** の 2 つを用意する．
- 実行プログラムの ILP 及び MLP を一定のインターバルで監視し，ILP もしくは MLP が高いと判断されたなら次のインターバルでは CONSERVATIVE モードでディスパッチし，そうでないなら AGGRESSIVE モードでディスパッチを行う．

本節では，まず 2 つのセグメント選択のアルゴリズムに関して説明を行う．その後，ILP 及び MLP の評価方法と，切り替えアルゴリズムに関して説明する．

5.2.1 2 つのセグメント選択アルゴリズム

SWITCH 方式では，タグ比較回数の削減重視の AGGRESSIVE モードと，容量効率重視の CONSERVATIVE モードの 2 つを適切に切り替えて使用する．各モードには，タグ

表 5.1: 2 つのセグメント選択モードのトレード・オフ

モード	タグ比較回数の削減	容量効率
AGGRESSIVE	○	×
CONSERVATIVE	×	○

比較回数の削減と容量効率に関して，表 5.1 に示すトレード・オフの関係がある．それぞれのセグメントの選択方法に関して説明する．

なお， SWITCH 方式はサブ・セグメントと併用が可能である．まず，サブ・セグメントを使用しない場合の AGGRESSIVE 及び CONSERVATIVE のアルゴリズムに関した説明した後，サブ・セグメントと併用する場合のアルゴリズムへと拡張して説明する．

5.2.2 AGGRESSIVE モード（サブ・セグメント不使用）

AGGRESSIVE モードは，表 4.1 で示した選択アルゴリズムを使用してディスパッチするエントリを決定する．このモードでは，選択されたセグメントに空きがない場合，他のセグメントに空きがあってもディスパッチは行わないため，容量効率が低下する．しかし，セグメント化の利益を最大限利用し，タグ比較回数を大幅に削減できる．

5.2.3 CONSERVATIVE モード（サブ・セグメント不使用）

AGGRESSIVE モードでは，命令のソース・オペランドが両方レディであり，セグメント・インディペンデントとしてディスパッチできる場合以外では，ソース・タグによって選択されるセグメントに空きがない場合にディスパッチをストールさせる．これに対し，CONSERVATIVE モードでは，以下で説明する工夫を行うことによって，このディスパッチのストールを回避し，容量効率の低下を抑制する．

- 両ソース・オペランドともレディでない場合：

  AGGRESSIVE モードでは第 1 ソース・タグの下位ビットによってセグメントを選択する．選択されたセグメントに空きがない場合，ディスパッチを行わない．これに

対して CONSERVATIVE モードでは，第 1 ソース・タグによって選択されたセグメントに空きがない場合には，**スワップ**[1]**して**ディスパッチを試みる．スワップするため，第 2 ソース・タグにより選択されるセグメントに空きがあればディスパッチが可能となる．

- **第 1 ソース・オペランドのみレディである場合**：

  AGGRESSIVE モードでは，スワップを行い，第 2 ソース・タグでセグメントを選択する．選択されたセグメントに空きがない場合，ディスパッチを行わない．これに対して CONSERVATIVE モードでは，第 2 ソース・タグによって選択されたセグメントに空きがない場合には，**スワップをやめて**ディスパッチする．スワップをやめるため，第 1 ソース・タグによってセグメントが選択されようとするが，第 1 ソース・オペランドは既にレディであるため，どのセグメントにディスパッチしても良い．従って，セグメント・インディペンデントとしてディスパッチが可能となる．

- **第 2 ソース・オペランドのみレディである場合**：

  AGGRESSIVE モードでは第 1 ソース･タグでセグメントを選択する．選択されたセグメントに空きがない場合，ディスパッチを行わない．これに対して CONSERVATIVE モードでは，第 1 ソース・タグによって選択されたセグメントに空きがない場合には，**スワップして**ディスパッチする．スワップするため，第 2 ソース・タグによってセグメントが選択されようとするが，第 2 ソース・オペランドは既にレディであるため，どのセグメントにディスパッチしても良い．従って，セグメント・インディペンデントとしてディスパッチが可能となる．

上述の工夫によって，CONSERVATIVE モードでは，どちらかのソース・オペランドがレディである場合は，必ずディスパッチが可能となる．また，両ソース・オペランドともレディでない場合でも，第 1 ソース・タグにより選択されるセグメントと第 2 ソース・タグ

[1] 4.2.1 節では，スワップの定義を「第 1 ソース・オペランドのみレディの場合に，第 1 ソース・タグと第 2 ソース・タグを書き込むフィールドを交換する」としていたが，本節以降ではこの定義を拡大し，単に「第 1 ソース・タグと第 2 ソース・タグを書き込むフィールドを交換する」という意味で用いる．

により選択されるセグメントのうち，いずれかのセグメントに空きがあればディスパッチが可能となる．従って，ストールする確率は大きく減少する．

表 5.2 に，CONSERVATIVE モードにおけるセグメントの選択アルゴリズムをまとめる．

表 5.2: CONSERVATIVE モードのアルゴリズム

ソース・タグの状態	アルゴリズム
（NR，NR）	第 1 ソース・タグでセグメントを選択．選択したセグメントに空きがない場合， スワップして第 2 ソース・タグをもとにセグメントを決定．なおも空きがない場合はストール．
（R，NR）	スワップを行い，第 2 ソース・タグでセグメントを選択．選択したセグメントに空きがない場合，スワップをやめてセグメント・インディペンデントとしてセグメントを選択する．
（NR，R）	第 1 ソース・タグでセグメントを選択．選択したセグメントに空きがない場合，スワップを行いセグメント・インディペンデントとしてセグメントを選択．
（R，R）	セグメント・インディペンデントとしてセグメントを選択．

CONSERVATIVE モードにおけるタグ比較回数の削減

CONSERVATIVE モードでは，AGGRESSIVE モードと比較してタグ比較回数の削減率が 低下する可能性がある．この理由について説明する．例として，第 2 ソース・オペランドのみレディである命令をディスパッチする場合について説明する．

CONSERVATIVE モードでは，まず第 1 ソース・タグでセグメントを選択する．選択されたセグメントに空きがあれば，そのセグメントにディスパッチする．この場合，レディでない第 1 ソース・タグが，セグメント化によってタグ比較回数が削減される第 1 ソース・タグのフィールドに書き込まれるため，AGGRESSIVE モードと同様にタグ比較回数が削減される．

第 1 ソース・タグによって選択されたセグメントに空きがなければ，CONSERVATIVE モードではスワップしてセグメント・インディペンデントとしてディスパッチする．この場合，タグ比較回数の削減は行うことができない．これは，既にレディである第 2 ソース・オペランドのタグが，セグメント化によってタグ比較回数を削減できる第 1 ソース・タグのフィールドに書き込まれ，一方で，まだレディでなくタグ比較が行われる第 2 ソース・オペランドのタグが，セグメント化によってタグ比較回数が削減されない第 2 ソース・タ

グのフィールドに書き込まれるためである．

AGGRESSIVE モードでは，第 1 ソース・タグによって選択されたセグメントに空きがなければ，ストールして空きが出るまで待ち合わせる．このストールにより，容量効率は低下するが，空きが出たあとディスパッチするため，タグ比較回数は削減される．これに対して CONSERVATIVE モードでは，タグ比較回数の削減は行えなくなるが，スワップしてディスパッチすることによってストールを回避し，容量効率の低下を防ぐ．

従って，CONSERVATIVE モードは，タグ比較回数の削減をある程度犠牲にして，IQ の容量効率の低下を抑制するアルゴリズムであるといえる．

5.2.4 サブ・セグメントとの併用

SWITCH 方式とサブ･セグメントを併用する際の，AGGRESSIVE と CONSERVATIVE のディスパッチ・アルゴリズムに関して説明する．

AGGRESSIVE モードに関しては，表 4.2 で示したアルゴリズムがそのまま サブ・セグメントを併用する場合の AGGRESSIVE モードでのアルゴリズムとなる．

CONSERVATIVE モードのアルゴリズムを，表 5.3 に示す．

表 5.3: CONSERVATIVE モードのアルゴリズム（サブ・セグメントと併用）

ソース・タグの状態	アルゴリズム
（NR，NR）	第 1 ソース・タグでメイン・セグメントを，第 2 ソース・タグでサブ・セグメントを選択．選択したセグメントに空きがない場合， スワップして第 2 ソース・タグでメイン・セグメントを，第 1 ソース・タグでサブ・セグメントを決定．なおも空きがない場合はストール．
（R，NR）	スワップを行い，第 2 ソース・タグでメイン・セグメントを選択し，サブ・セグメントは S-seg インディペンデントとして選択．選択したセグメントに空きがない場合，スワップをやめ，第 2 ソース・タグでサブ・セグメントを選択し，メイン・セグメントは M-seg インディペンデントとして選択．なおも空きがない場合はストール．
（NR，R）	第 1 ソース・タグでメイン・セグメントを選択し，サブ・セグメントは S-seg インディペンデントとして選択．選択したセグメントに空きがない場合，スワップして，第 1 ソース・タグでサブ・セグメントを選択し，メイン・セグメントは M-seg インディペンデントとして選択．なおも空きがない場合はストール．
（R，R）	セグメント・インディペンデントとしてセグメントを選択．

（NR，NR），（R，NR），（NR，R）の場合に関して，例を用いて以下で説明する．メイン・セグメント数を 4，サブ・セグメント数を 2 とし，命令 $p5 = p13 + p6$ をディスパッ

チする場合について例示する．第 1 ソース・タグが 13 で，第 2 ソース・タグが 6 である．

- （NR，NR）: 第 1 ソース・タグでメイン・セグメントを，第 2 ソース・タグでサブ・セグメントを選択する．この場合，第 1 ソース・タグ 13（1101_2），第 2 ソース・タグ 6（110_2）より,（1，0）のセグメントを選択する． もし（1，0）に空きがない場合はスワップを行い，第 2 ソース・タグでメイン・セグメントを，第 1 ソース・タグでサブ・セグメントを選択する．この場合,（2，1）が選択される．なおも空きがない場合はストールする．

- （R，NR）: スワップを行い，第 2 ソース・タグでメイン・セグメントを選択する．例の場合，第 2 ソース・タグ 6（110_2）より，メイン・セグメントは 2 となる．第 1 ソース・オペランドは既にレディであるため，S-seg インディペンデントである．従って,（2，0）または（2，1）のいずれかのセグメントを選択する.（2，0）と（2，1）のいずれも空きがない場合は，スワップをやめ，第 2 ソース・タグでサブ・セグメントを決定する．この場合，サブ・セグメントは 0 となる．第 1 ソース・オペランドは既にレディであるため，M-seg インディペンデントである．したがって（0，0）,（1，0）,（2，0）,（3，0）のいずれかのセグメントが選択される．なおも空きがない場合はストールする．

- （NR，R）: 第 1 ソース・タグでメイン・セグメントを選択する．例の場合，第 1 ソース・タグ 13（1101_2）より，メイン・セグメントは 1 となる．第 2 ソース・オペランドは既にレディであるため，S-seg インディペンデントである．従って,（1，0）または（1，1）のいずれかのセグメントを選択する.（1，0）と（1，1）のいずれも空きがない場合は，スワップを行い，第 1 ソース・タグでサブ・セグメントを決定する．この場合，サブ・セグメントは 1 となる．第 2 ソース・オペランドは既にレディであるため，M-seg インディペンデントである．したがって（0，1）,（1，1）,（2，1）,（3，1）のいずれかのセグメントが選択される．なおも空きがない場合はストールする．

5.2.5 モードの切り替え

SWITCH 方式では， AGGRESSIVE と CONSERVATIVE の 2 つのモードを，実行プログラムの ILP や MLP の量に応じて切り替えて使用する．ここで重要となるのは ILP や MLP の量の評価方法である．

本研究では， ILP の評価方法として Instructions Per Cycle（IPC）と Issue Stall Rate（ISR） という評価値が有効ではないかと考えた．また， MLP の評価方法としては，最終レベル・キャッシュ（LLC: last-level cache）の MPKI（misses per kilo instructions）が有効ではないかと考えた．それぞれに関して詳しく説明した後，切り替えアルゴリズムを説明する．

なお，評価の結果，ILP を評価する評価値としては，IPC と ISR のどちらも有効であるが，IPC のほうがより精度が高いことが分かったため，IPC を ILP の評価値として使用する．これらの評価は 6.4 節で説明する．

Instructions Per Cycle（IPC）

IPC は,「サイクルあたりの平均コミット命令数」を表す指標であり，プロセッサの性能指標として一般的に使用される評価値である．IPC が高い場合，ILP は高いと判断される．

あらかじめ IPC にしきい値を設け，定期的に IPC を観測し，インターバルでの IPC がしきい値を上回った場合に ILP が高いと判断し，そうでなければ低いと判断する．

Issue Stall Rate（ISR）

ISR は,「インターバルの全サイクルのうち 1 命令も発行されないサイクルの割合」を表す指標である．ILP が高い場合，多くのサイクルで命令が発行されるため，ISR は低い値を示す．一方， ILP が低い場合には，命令が発行されないサイクルが一定の割合で発生するため，ISR は高くなる．

あらかじめ ISR にしきい値を設け，定期的に IPC を観測し，インターバルでの ISR がしきい値を下回った場合に ILP が高いと判断し，そうでなければ低いと判断する．

LLC MPKI

LLC MPKI は LLC のキャッシュ・ミスの発生頻度を表す指標である．あらかじめ LLC MPKI にしきい値を設け，定期的に LLC MPKI を観測し，インターバルでの LLC MPKI がしきい値を上回った場合に MLP が高いと判断し，そうでなければ低いと判断する．

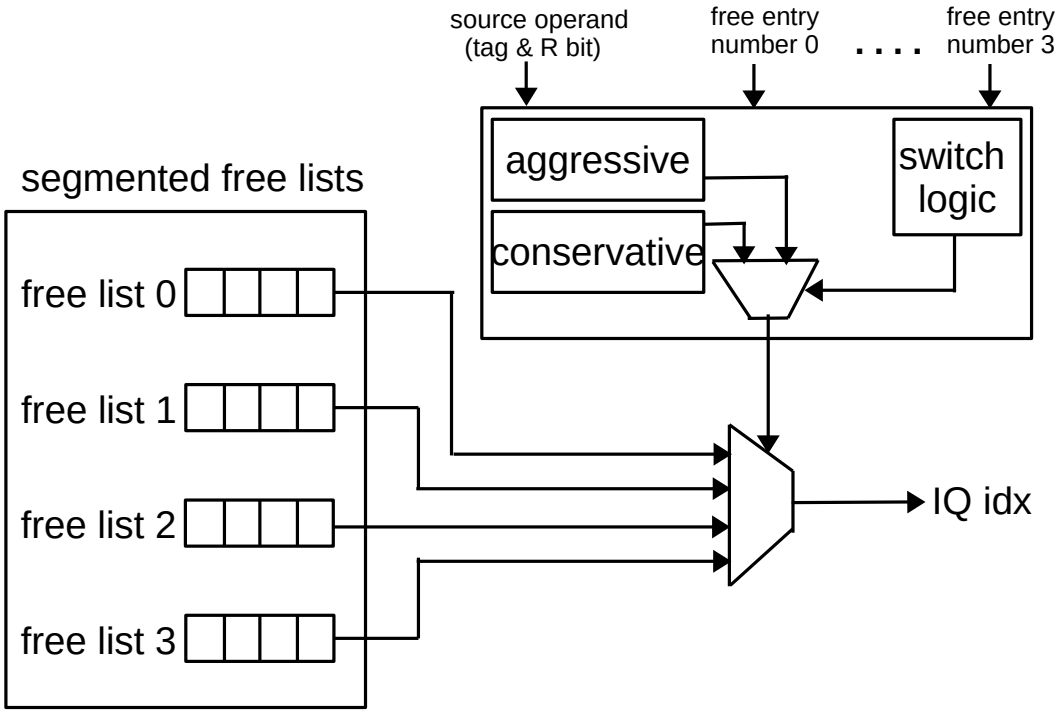

図 5.2: SWITCH 方式におけるディスパッチエントリの決定回路

切り替えアルゴリズム

切り替えアルゴリズムは以下に示すとおりである．定期的に評価指標である IPC（もしくは ISR）と LLC MPKI を測定し，ILP および MLP の高低を判断する．ILP または MLP

のいずれかが高いと判定された場合，次のインターバルを CONSERVATIVE モードで実行する．ILP と MLP がどちらも低いと判定された場合，次のインターバルを AGGRESSIVE モードで実行する．

SWITCH 方式におけるディスパッチするエントリの決定回路を図 5.2 に示す．AGGRESSIVE と CONSERVATIVE の 2 つの選択アルゴリズムのうち，どちらを利用するかを SWITCH 回路が選択し，その結果に応じてセグメントが選択される．

第6章　評価

本章では，提案手法の評価を行う．6.1 節で評価環境について説明し，6.2 節で提案手法によるタグ比較回数と性能低下の評価を行う．6.3 節でサブ・セグメントに関する評価を説明した後，6.4 節で SWITCH 方式のパラメータに関する評価を行い，6.5 節でセグメント数に関する評価について説明する．

6.1　評価環境

評価環境について説明する．性能やタグ比較回数を評価するために，SimpleScalar v.3.0a [31] をベースに作成したシュミレータを使用した．評価で仮定したプロセッサ構成を表 6.1 に示す．

提案手法の SWITCH 方式に関するパラメータを，表 6.2 に示す．これらのパラメータは，6.4 節 で説明する評価に基づいて決定した最適なパラメータである．

測定ベンチマークには，SPEC CPU 2017 ベンチマークのうち，int 系 9 本と fp 系 9 本の計 18 本を使用した（gcc と wrf は，現在のところ，シミュレータでは正しく動作しなかったため，除いている）．プログラムの入力には refspeed データ・セットを用いた．ベンチマークの測定区間は，プログラムの先頭から 16B 命令をスキップした後の 100M 命令である．

ベンチマークの分類

提案手法は，プログラムの ILP や MLP に着目した制御を行う．そこで，SPEC CPU 2017 ベンチマークを ILP が高いベンチマーク，MLP が高いベンチマーク，いずれも低いベンチマークの 3 種類に分類する．ここて，ILP 及び MLP が高いベンチマークとは，次の条件を満たすベンチマークであるとした．

表 6.1: プロセッサの基本構成

Pipeline width	8 instructions wide for each of fetch，decode，issue，and commit
Reorder buffer	300 entries
IQ	128 entries，w/ age matrix
Load/Store queue	128 entries
Physical registers	300（int） + 300（fp）
Branch prediction	16KB Perceptron predictor [32] 2K-set 4-way BTB 10-cycle misprediction penalty
Function unit	4 iALU，2 iMULT， 3 FPU，2 LSU
L1 D-cache	32KB，8-way，64B line 2-cycle hit latency
L1 I-cache	32KB，8-way，64B line 2-cycle hit latency
L2 cache	2MB，16-way，64B line 12-cycle hit latency
Main memory	300-cycle latency 8B/cycle bandwidth
Prefetch	stream-based，32-stream tracked， 16-line distance，2-line degree， prefetch to L2 cache

表 6.2: 提案手法の SWITCH 方式に関するパラメータ構成

切り替えインターバル	10K instructions
IPC しきい値	3.5
LLC MPKI しきい値	2.0

- high ILP：IPC が 3.5 以上のベンチマーク
- high MLP：LLC MPKI が 2.0 以上のベンチマーク

分類結果を表 6.3 に示す．また，以降に示す図において，ILP（青色） 及び MLP（赤色）の表記は，ILP もしくは MLP が高いベンチマークであることを表す．

評価モデル

評価モデルは以下の 4 種類である．

- BASE：セグメント化しない通常の IQ を使用するモデル

表 6.3: ベンチマークの分類

分類	ベンチマーク
high ILP	xz，bwaves，cactuBSSN，cam4，imagick，pop2，roms
high MLP	omnetpp，xalancbmk，lbm
low ILP and low MLP	exchange2，leela，deepsjeng，mcf perlbench，x264，fotonik3d，nab

- AGGRESSIVE：提案手法において常に AGGRESSIVE モードで実行するモデル
- CONSERVATIVE：提案手法において常に CONSERVATIVE モードで実行するモデル
- SWITCH：提案手法の SWITCH 方式を使用するモデル

タグ比較回数の測定

提案手法によるタグ比較回数の削減を評価するために，タグ比較の回数を測定した．ここで，タグ比較の回数とは，**ウェイクアップ時にレディでないオペランドの比較において，タグが一致しなかった数**とする．これは，電力消費に関する以下の理由による．

- すでにレディなオペランドの比較器は動作させない．比較器のプリチャージを抑制することにより，容易に停止させることができ，このとき電力を消費しない．
- デスティネーション・タグは，最大で発行幅分送られてくるが，送られてこなかったデスティネーション・タグのタグ線につながっている比較器は動作しないとする．ここで「送られてこなかったタグ」とは，物理的には，プロセッサ内で使用されないタグ（偽のタグ）を定めそれを送信することとする．偽のタグとしてすべてのビットが 0 のタグを選択する．これにより，偽のタグの放送においては，タグ線はすべて L となり，比較器のマッチ線はディスチャージされず，電力を消費しない．
- タグが一致した比較器は，プリチャージされたマッチ線の電荷がディスチャージされないので，電力を消費しない．

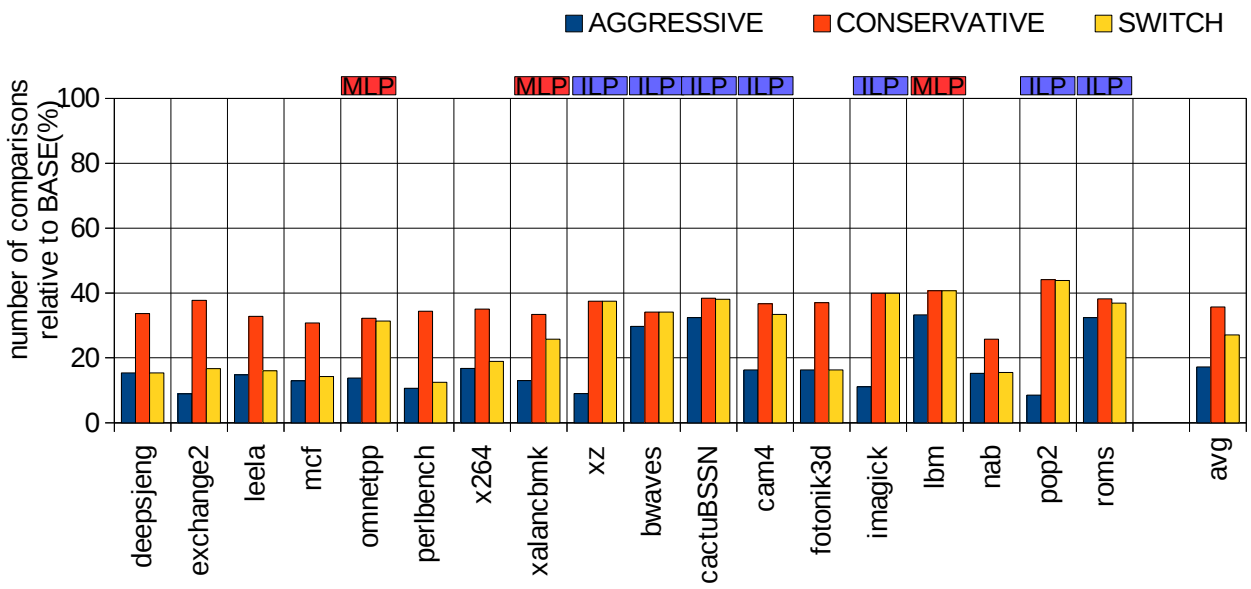

図 6.1: 提案手法によるタグ比較回数（16，1）

6.2 提案手法によるタグ比較回数削減と性能低下の評価

提案手法によるタグ比較回数の削減と性能低下に関して評価を行う．サブ・セグメントを使用せず，セグメント数は 16 とした．このセグメント数は，提案手法による性能低下とタグ比較回数のバランスを考慮し，提案手法の特徴をよく評価できるパラメータであると考え選んだものである．

なお，以降の評価において,（メイン・セグメント数，サブ・セグメント数） の形式でセグメント数を表記する．また，サブ・セグメントを使用しない場合は，サブ・セグメント数は 1 と表記する．今回の場合は（16，1）となる．

6.2.1 タグ比較回数の削減

図 6.1 に，提案手法の BASE モデルに対するタグ比較回数の割合をベンチマークごとに示す．GM は全ベンチマークでの幾何平均を表す．AGGRESSIVE と CONSERVATIVE のタグ比較回数削減に関しては，同図より，いずれのベンチマークにおいても，AGGRESSIVE の方がタグ比較回数が少ないことがわかる．その差は 平均で 20% ポイント程度となってお

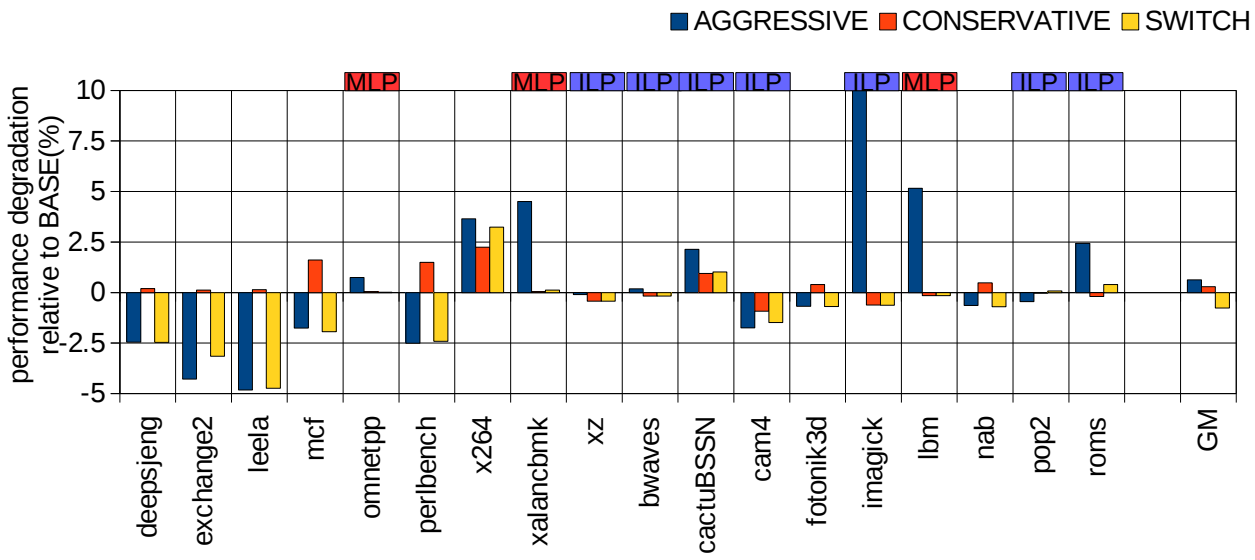

図 6.2: 提案手法による性能低下（16，1）

り，AGGRESSIVE モードのタグ比較回数を積極的に削減できるという性質が確認できる．

同図より SWITCH 方式では，平均で BASE モデルの 25% 程度のタグ比較回数となっており，75% の削減を達成している．

SWITCH 方式では，ILP や MLP の高いベンチマークにおいては CONSERVATIVE と同程度のタグ比較回数であるのに対して，そうでないベンチマークにおいては AGGRESSIVE に近いタグ比較回数となっていることがわかる．したがって，IQ の容量効率が重要でないベンチマークにおいては，AGGRESSIVE モードを選択して積極的にタグ比較回数の削減が行えていることがわかる．

6.2.2　性能低下

図 6.2 に，BASE に対する提案手法による性能低下をベンチマークごとに示す．同図より，SWITCH 方式による性能低下は最大で 3.5% 程度であり，多くのベンチマークでは 0% に近く性能はほとんど低下しないということが確認できる．

SWITCH 方式の有効性に関して述べる．同図より，ILP や MLP が高い xalancbmk や imagick，lbm などのベンチマークにおいて，AGGRESSIVE では大きく性能低下してい

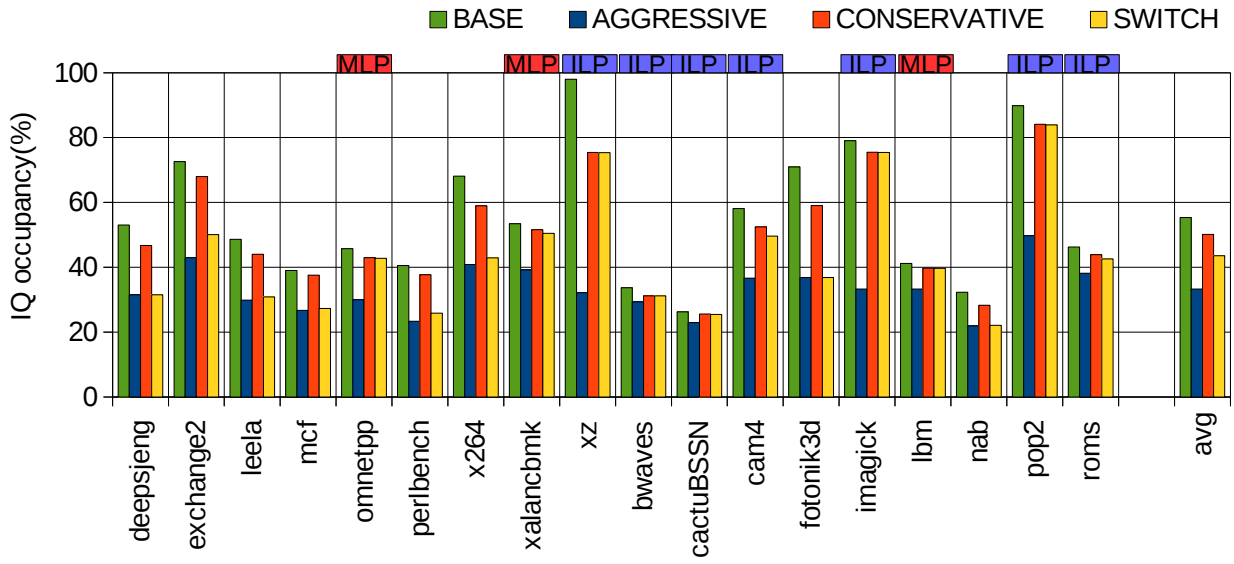

図 6.3: IQ の占有率（16，1）

るのに対して，CONSERVATIVE では性能低下が抑制されていることが分かる．そして SWITCH では，CONSERVATIVE と同程度の性能低下にとどまっている．従って，容量効率が性能にとって重要なプログラムにおいて，SWITCH 方式によって性能低下が抑制できていることが分かる．

図 6.3 に各モデルでの IQ の占有率を示す．占有率とは，IQ の全エントリのうち 1 サイクルあたり使用されたエントリの割合であり，この値が BASE のそれに近いほど容量効率が低下していないことを示す．

同図より，AGGRESSIVE では BASE に対して占有率が大きく低下しているのに対して，CONSERVATIVE では占有率の低下がある程度抑制できていることが分かる．そして，ILP や MLP が高いベンチマークでは SWITCH 方式での占有率が CONSERVATIVE と同程度となっていることが分かる．このことからも，SWITCH 方式では，容量効率の性能に対する重要性に応じて適切にモードを選択し，容量効率の低下による性能低下を抑制できており，SWITCH 方式が有効であると言える．

図 6.2 より，いくつかのベンチマークでは性能が向上していることが分かる．この理由に関して説明する．一般にランダム・キュー方式の IQ には，命令がプログラム順に並ん

でいないため，最も優先して発行すべき命令の発行が遅れる可能性があるという欠点が存在する．ランダム・キューでは，命令の並びが年齢についてランダムになる一方，選択論理は，下のエントリほど優先して発行命令を選択するため，レディ命令が発行幅以上に存在する発行コンフリクトが生じた場合，性能に好影響を与えない命令の選択が生じる．

提案手法では IQ の容量効率が低下するため，IQ 内の命令数が少なくなり，結果的に発行コンフリクトが生じる確率が低下する．この結果，問題が生じにくくなり僅かに性能が向上する．

図 6.2 において，性能が向上するベンチマークの他に，ILP や MLP が高いにもかかわらず，AGGRESSIVE においても性能低下の小さいベンチマークがあることも分かる．こういったベンチマークにおいても，発行コンフリクトの緩和による性能向上が発生しているため，容量効率の低下による性能低下と中和されて，結果として性能低下が小さいと考えられる．

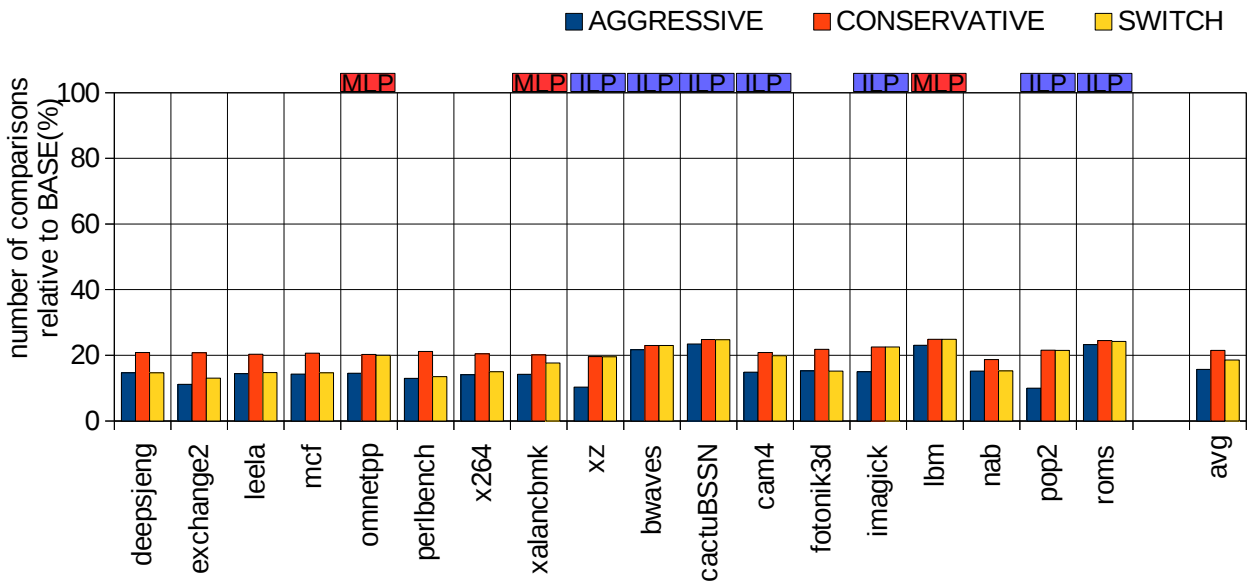

図 6.4: 提案手法によるタグ比較回数（8，2）

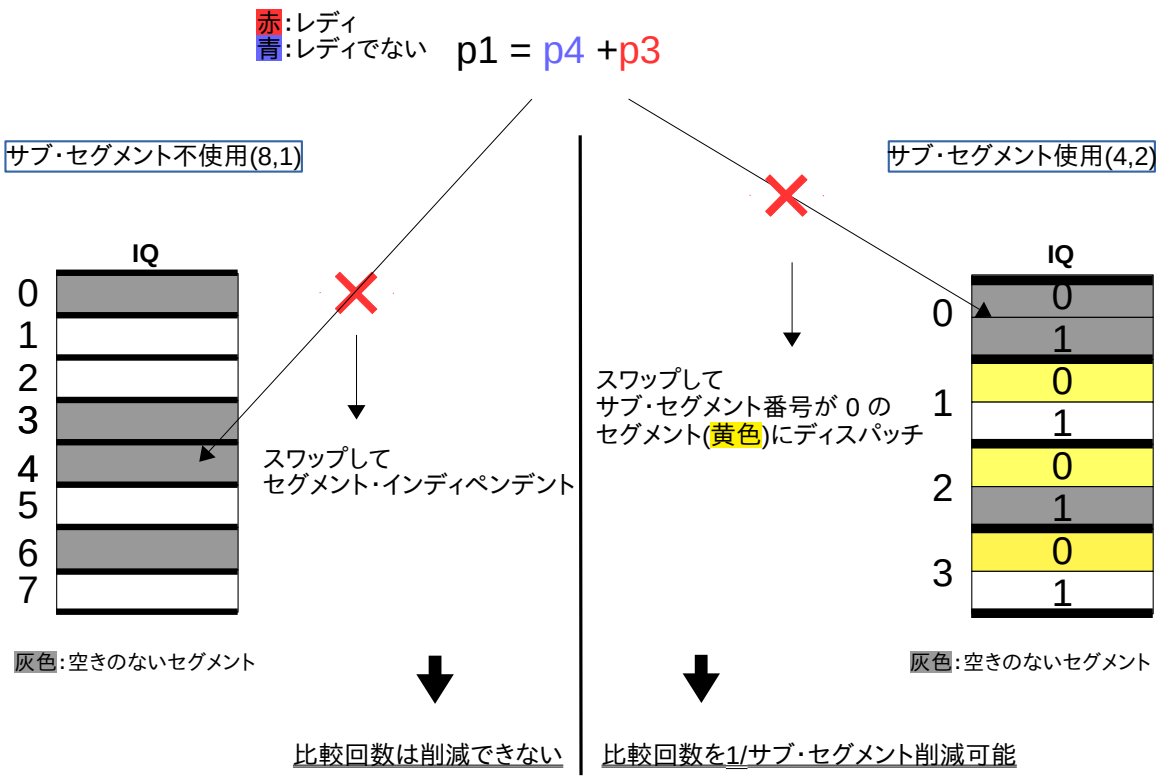

図 6.5: サブ・セグメントを使用する CONSERVATIVE モードにおけるタグ比較回数

6.3 サブ・セグメントに関する評価

サブ・セグメントを使用する場合の提案手法に関して評価する．メイン・セグメント数が 8，サブ・セグメント数が 2 の場合（(8，2）と表記する）に関して評価を行った．このセグメントの組み合わせは，(16，1） の場合とセグメントの総数が同じであり，比較の対象として適していると考え選んだものである．また，(8，2）の組み合わせは，6.5 節で説明するセグメントの分割数に関する評価において，性能と電力削減のバランスにおいて最適と判断された組み合わせである．

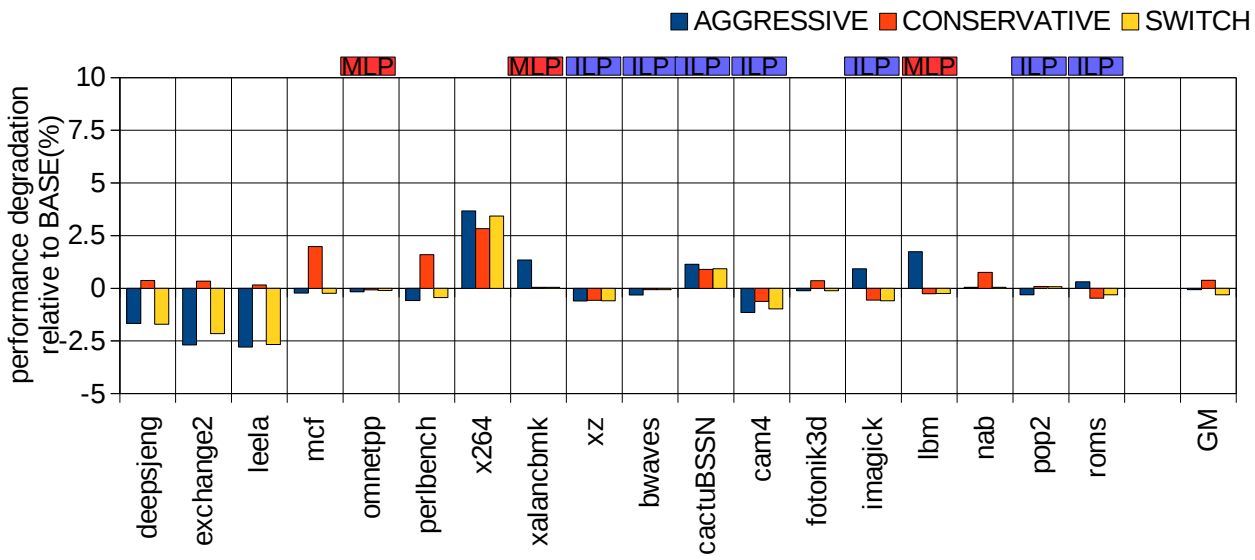

図 6.6: 提案手法による性能低下（8，2）

表 6.4: （16，1）と（8，2）の比較

		タグ比較回数	性能低下（最大）	性能低下（平均）
（16，1）	AGGRESSIVE	16%	10.8%	0.6%
	CONSERVATIVE	35%	2.2%	0.3%
	SWITCH	25%	3.8%	-0.8%
（8，2）	AGGRESSIVE	15%	3.7%	-0.1%
	CONSERVATIVE	21%	2.8%	0.3%
	SWITCH	18%	3.4%	-0.4%

6.3.1　タグ比較回数の削減

図 6.4 に，提案手法の BASE モデルに対するタグ比較回数の割合をベンチマークごとに示す．また，表 6.4 に，(16，1） と（8，2）の各評価モデルにおけるタグ比較回数の平均値，性能低下の最大値，性能低下の平均値を示す．

表 6.4 より，(16，1）と （8，2）の CONSERVATIVE のタグ比較回数を比較すると，(16，1）の場合が平均で 35% であるのに対して，(8，2） では 21 % 程度と，(8，2）のほうがより削減できていることが分かる．これは，サブ・セグメントを使用する場合，5.2.1 節で説明した CONSERVATIVE モードでのストールの回避を行った際にも，タグ比較の削減が可能となるためである．図 6.5 を用いて詳しく説明する．

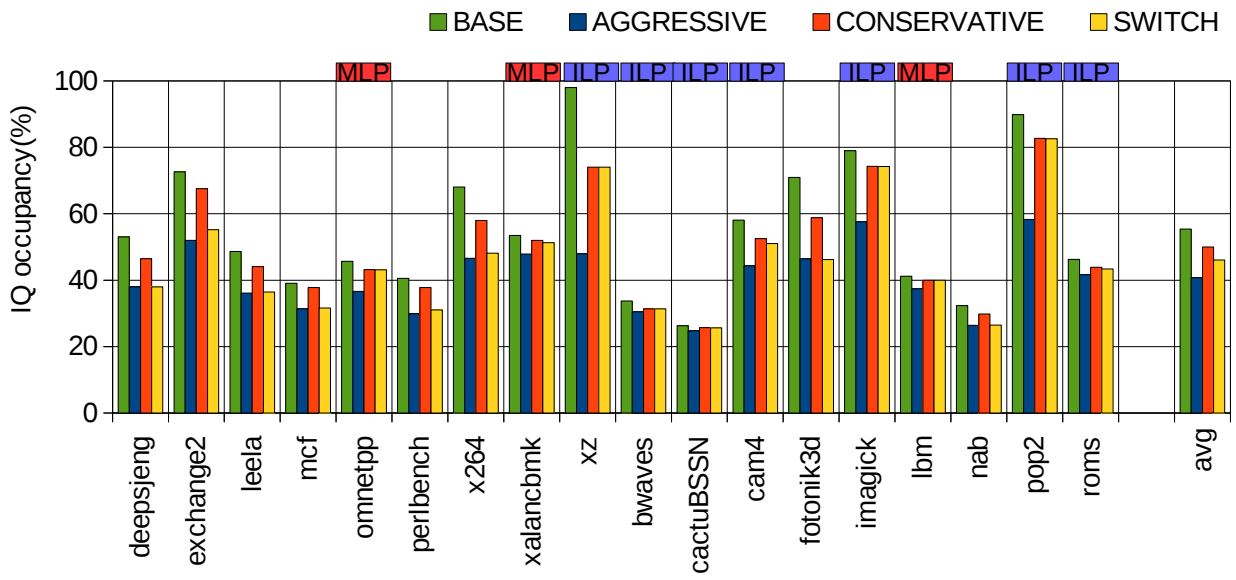

図 6.7: IQ の占有率（8，2）

図中に示す命令をディスパッチする場合を考える．サブ・セグメントを使用しない場合（図左側），第 1 ソース・タグ $p4$ によって決定されるセグメント（第 4 セグメント）に空きがなければ，CONSERVATIVE ではスワップを行い，セグメント・インディペンデントとしてディスパッチを行う．この場合，第 1 ソース・タグ $p4$ のタグ比較回数は削減されない．

一方で，サブ・セグメントを使用する場合（図右側），第 1 ソース・タグ $p4$ によってメイン・セグメントが決定され，もし空きがなければ，スワップを行う．そして，第 1 ソース・タグ $p4$ によってサブ・セグメント番号が決定され，該当する番号のサブ・セグメントのいずれかに空きがあれば（図中の黄色で示したセグメント），そのセグメントにディスパッチする．この場合，第 1 ソース・タグ $p4$ の比較は，タグの下位ビットがサブ・セグメント番号と一致する場合のみ行われるため，その比較回数は「1 / サブ・セグメント数」まで削減が可能となる．

以上で説明したように，サブ・セグメントを用いることにより，CONSERVATIVE モードでストールの回避を行った際にも，タグ比較削減の機会が増加する．その結果，CONSERVATIVE モードで高いタグ比較削減率を達成することができる．

最後に SWITCH 方式に関して評価する．(16，1） の場合と同様に容量効率の重要性に応じてモードの切り替えができており，IQ の容量効率が重要でないベンチマークにおいては AGGRESSIVE モードと同程度の削減率を達成できている．

6.3.2　性能低下

図 6.6 に，BASE に対する提案手法による性能低下をベンチマークごとに示す．同図より，SWITCH 方式による性能低下は最大で 4% 程度であり，多くのベンチマークでは 0% に近く性能はほとんど低下しないということが確認できる．

AGGRESSIVE モードでの性能低下率に関して考える．表 6.4 より，(8，2） では，(16，1）と比較して AGGRESSIVE モードでの性能低下率が低いことが分かる．これは，サブ・セグメントによって AGGRESSIVE モードでの容量効率の低下が抑制されているためであると考えられる．

図 6.7 に，(8，2）の場合の IQ の占有率を示す．図 6.3 と図 6.7 の占有率を比較すると，(16，1）の場合は平均で 30% 程度であった占有率が，(8，2）では平均で 40% となっている．また，(16，1）の AGGRESSIVE モードにおいて特に性能低下の大きかった imagick に関して見てみると，(16，1）の AGGRESSIVE モードでは占有率が 35% 程度で，性能低下が 10% であるのに対して，(8，2） の AGGRESSIVE モードでは占有率が 65% 近くまで上昇しており，その結果性能低下が 3% 程度となっている．

以上の考察から，セグメントの総数が同じである場合，サブ・セグメントを使用することによって，AGGRESSIVE モードでの容量効率の低下による性能低下を抑制できることがわかった．

最後に，SWITCH 方式の有効性に関して説明する．(8，2）の場合，サブ・セグメントが有効であり，AGGRESSIVE での大幅な性能低下が見られないため，(16，1）の場合と比較して SWITCH 方式の有効性は平均では高くない．しかし，imagick や lbm などのベンチマークにおいては AGGRESSIVE モードで発生する性能低下を抑制できている．また，

CONSERVATIVE モードでのタグ比較削減率が高く，その結果 SWITCH 方式自体のタグ比較削減率も高くなっている．

サブ・セグメントの評価のまとめ

サブ・セグメントを使用すると，以下のメリットがあることがわかった．

- CONSERVATIVE モードでのタグ比較回数がより削減できる
- AGGRESSIVE モードにおける性能低下が改善される

したがって，サブ・セグメントは有効な手法であると言える．

6.4 SWITCH 方式のしきい値に関する評価

SWITCH 方式において，ILP と MLP の高低を判定するために使用するしきい値に関する評価を行う．しきい値に関する評価は，すべてのセグメント数の組み合わせに対して行い，それぞれ最適なしきい値を決定した．本節では（8，2）の場合に関して説明し，その他のセグメント数の場合に関しては ?? 節にて評価結果を示す．

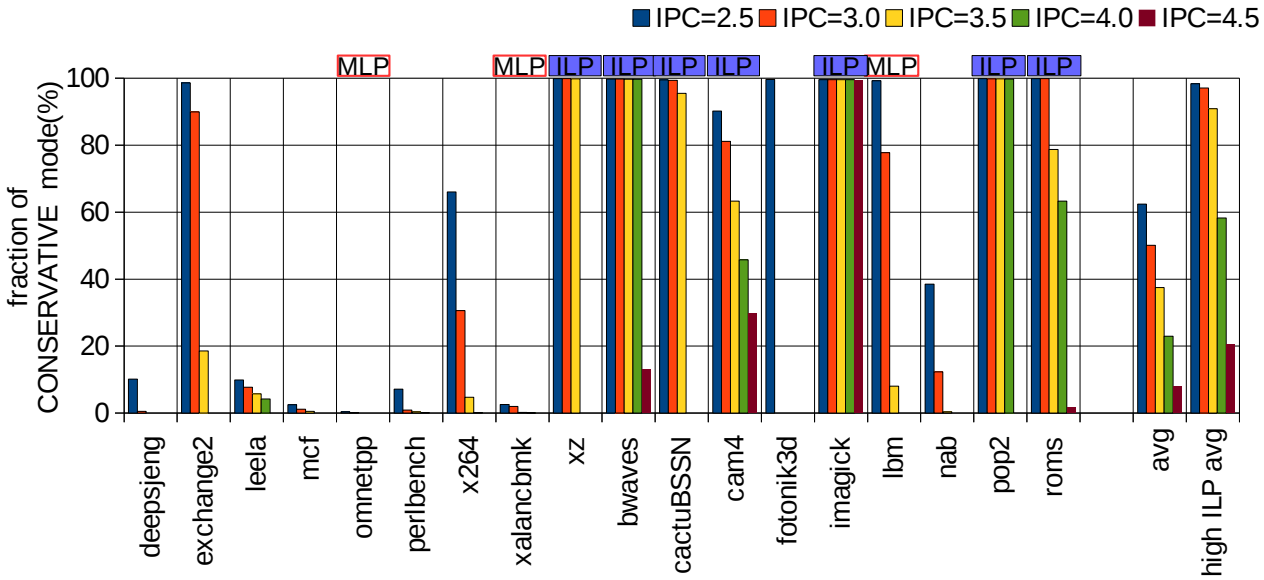

図 6.8: IPC を用いた SWITCH 方式の制御

6.4.1　ILP の評価値

ILP を評価する値として IPC と ISR の評価を行う．評価の方針としては，まず IPC と ISR のしきい値に関して適当な値を求める．適当ななしきい値は，しきい値を変化させた場合に，ILP の高いベンチマークにおいて CONSERVATIVE モードで実行される割合を評価することによって決定する．

その後，それぞれの評価値を利用する場合の提案手法によるタグ比較削減と性能低下を比較し，IPC と ISR のどちらがより適した評価指標か評価する．

IPC による制御

ILP の評価指標として，IPC を使用する場合の評価を行った．また，ILP による SWITCH 方式の制御のみ行い，MLP による制御は行っていない．

図 6.8 に，IPC のしきい値を変化させた場合の，CONSERVATIVE モードで実行される割合を示す．この割合が大きいほど，ILP が高いと判断され多くのサイクルが CONSERVATIVE モードで実行されていることを表す．また，各凡例の IPC=X は，ILP が高いと判定する IPC のしきい値を X とした場合を示している．また，avg は全ベンチマークの平均を，high ILP avg は ILP の高いベンチマークでの平均を表している．

同図より，IPC のしきい値が高くなるほど，CONSERVATIVE モードで実行される割合が小さくなっていることが分かる．これは，ILP が高いと判定される基準が厳しくなるためである．ILP の高いベンチマークに関して見ると，多くのベンチマークにおいて，しきい値が 3.5 の場合は CONSERVATIVE モードの割合が大きく，しきい値が 4.0 になると CONSERVATIVE モードの割合が小さくなっていることが分かる．high ILP avg を見ると，しきい値が 3.5 の場合は CONSERVATIVE モードの割合は 90% 程度であるが，しきい値が 4.0 の場合は CONSERVATIVE モードの割合は 60% 程度と小さくなる．

ILP が高い場合は CONSERVATIVE モードで実行することが望ましい．従って，IPC のしきい値は，3.5 程度が適当であるといえる．

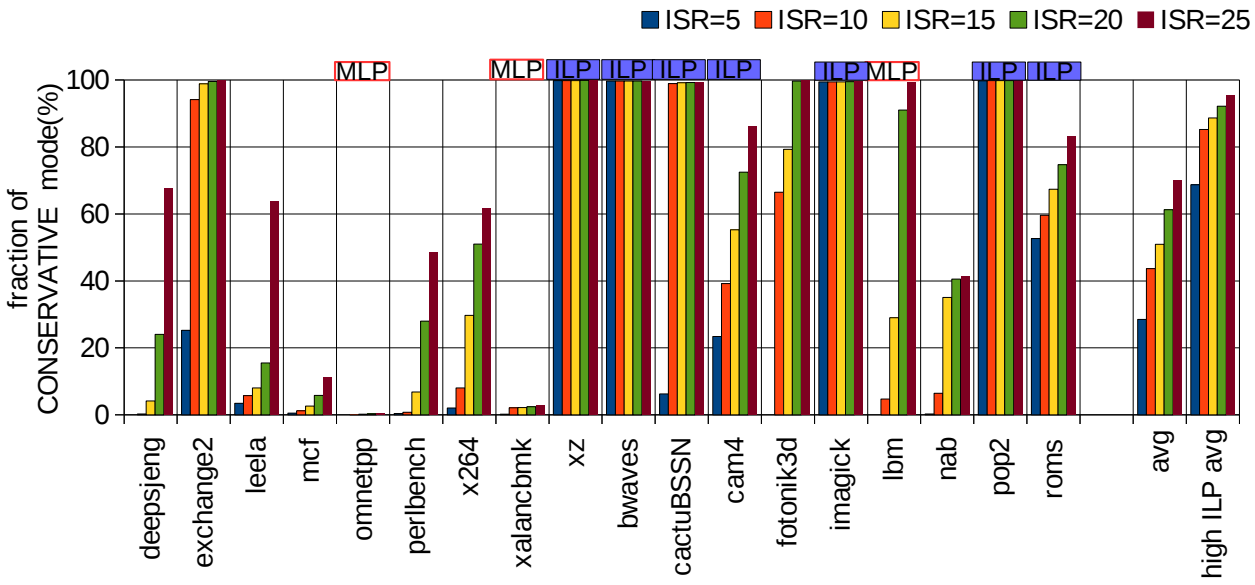

図 6.9: ISR を用いた SWITCH 方式の制御

ISR による制御

ILP の評価指標として，ISR を使用する場合の評価を行った．IPC の評価と同様に，ILP による SWITCH 方式の制御のみ行い，MLP による制御は行っていない．

図 6.9 に，ISR のしきい値を変化させた場合の，CONSERVATIVE モードで実行される割合を示す．各凡例の ISR=X は，ILP が高いと判定する ISR のしきい値を X とした場合を表す．

同図より，ISR のしきい値が低くなるほど，CONSERVATIVE モードで実行される割合が小さくなっていることが分かる．これは，ILP が高いと判定される基準が厳しくなるためである．high ILP avg を見ると，しきい値が 5 の場合は CONSERVATIVE モードの割合は 60% 程度であるが，しきい値が 10 の場合は CONSERVATIVE モードの割合は 90% 程度と大きくなる．

ILP が高い場合は CONSERVATIVE モードで実行することが望ましい．従って，ISR のしきい値は，10 程度が適当であるといえる．

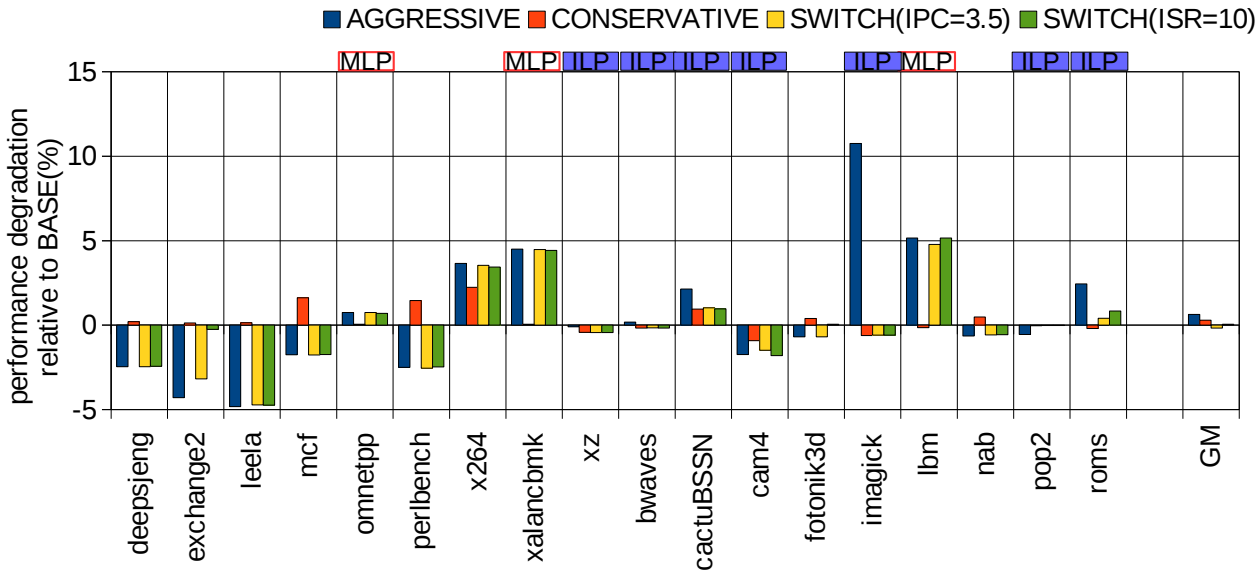

図 6.10: ILP による制御を行った SWITCH 方式による性能低下

IPC と ISR の比較

図 6.10 に，IPC と ISR を用いて制御を行う SWITCH 方式での性能低下を示す．MLP による制御は行っていない．同図より，ILP が高いベンチマークにおいては，IPC と ISR いずれの指標においても正しく評価できており，結果として SWITCH による性能低下が CONSERVATIVE と同程度に抑制できていることがわかる．また，ILP が高くないベンチマークにおいては，IPC と ISR による制御において大きな差はみられない．平均を見ても，IPC と ISR は同程度であると言える．

図 6.11 に，IPC と ISR を用いて制御を行う SWITCH 方式でのタグ比較回数を示す．タグ比較回数も，性能低下と同様に IPC と ISR で大きな差は見られないが，exchange2 と fotonik3d に関しては，IPC を用いた制御のほうが ISR を用いた制御よりもタグ比較回数が少ないことが分かる．この理由は，この 2 つベンチマークについては，IPC を用いた制御で ILP が低いと判定されているのに対して，ISR を用いた制御では ILP が高いと判定されているためである．

exchange2 及び fotonik3d は提案手法によって性能低下を起こさないベンチマークであ

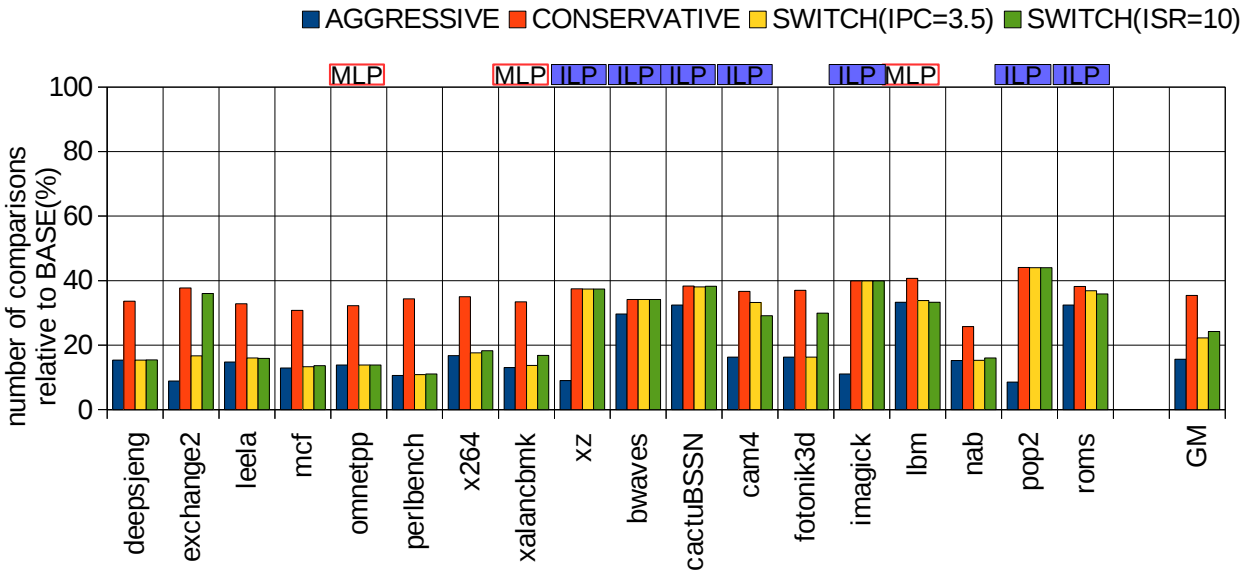

図 6.11: ILP による制御を行った SWITCH 方式によるタグ比較回数

るため，ILP が高いと判定されて CONSERVATIVE モードで実行されることは望ましくない．従って，これらのベンチマークを ILP が低いと判定できる IPC による制御のほうが適していると考えられる．

以上の評価から，ISR による制御は一部の ILP が低いベンチマークを ILP が高いと判定してしまうため，IPC を 用いた制御のほうがより適していると言える．また，この性質はセグメント数が異なる場合でも同様であった．したがって，SWITCH 方式における ILP の判定は，すべてのセグメント数の組み合わせで IPC を用いることとする．

6.4.2 MLP の評価値

MLP を評価する値として LLC MPKI の評価を行う．評価の方針としては，LLC MPKI のしきい値に関して適当な値を求める．適当なしきい値は，しきい値を変化させた場合に，MLP の高いベンチマークにおいて，CONSERVATIVE モードで実行される割合を評価することによって決定する．

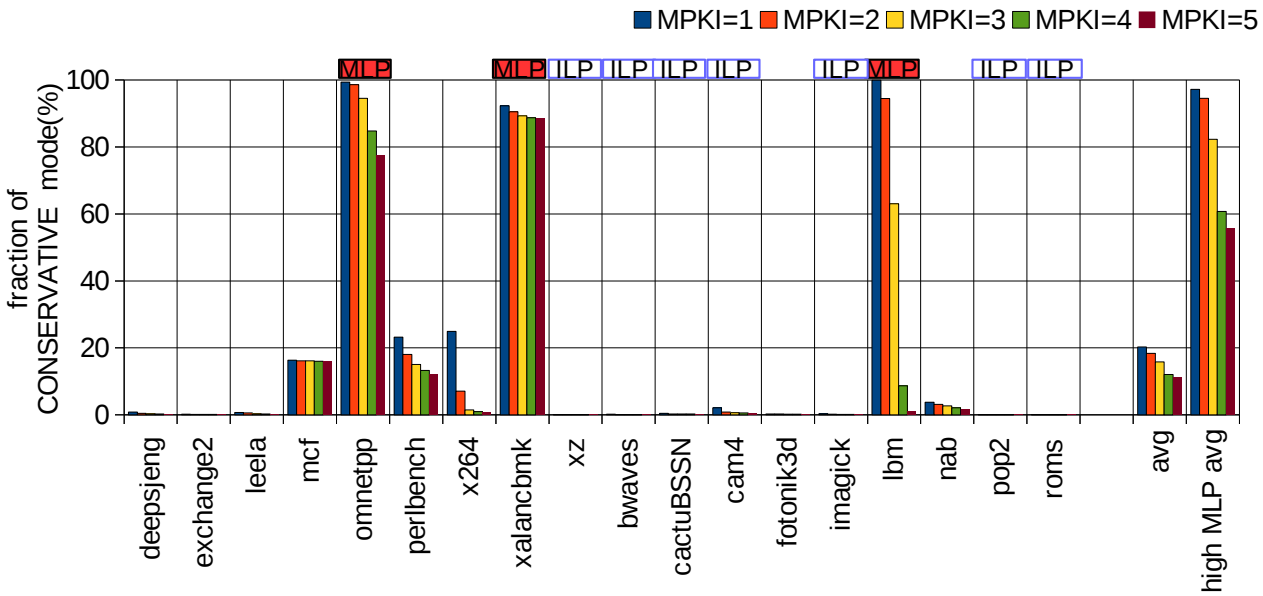

図 6.12: LLC MPKI による SWITCH 方式の制御

LLC MPKI による制御

図 6.12 に，LLC MPKI のしきい値を変化させた場合の，CONSERVATIVE モードで実行される割合を示す．各凡例の MPKI=X は，MLP が高いと判定する LLC MPKI のしきい値を X とした場合を表す．また，avg は全ベンチマークの平均を，high MLP avg は MLP の高いベンチマークでの平均を示している．なお，MLP による SWITCH 方式の制御のみ行い，ILP による制御は行っていない．

同図より，LLC MPKI しきい値が高くなるほど，CONSERVATIVE で実行される割合が小さくなっていることが分かる．これは，LLC MPKI が高いと判定される基準が厳しくなるためである．

MLP の高いベンチマークに関して考える．omnetpp と lbm においては，しきい値が 2 以下場合は CONSERVATIVE モードの割合が大きいが，しきい値が 3 以上 になると CONSERVATIVE モードの割合が急激に小さくなっていることが分かる．xalancbmk に関しては，MLP の高いベンチマークの中でも特にメモリ・インテンシブなベンチマーク（LLC MPKI が 10 程度）であるため，しきい値を 5 まで増加させても CONSERVATIVE

の割合は大きく変化しない．

MLP が高い場合は CONSERVATIVE モードで実行することが望ましい．従って，LLC MPKI のしきい値は，2 程度が適当であるといえる．

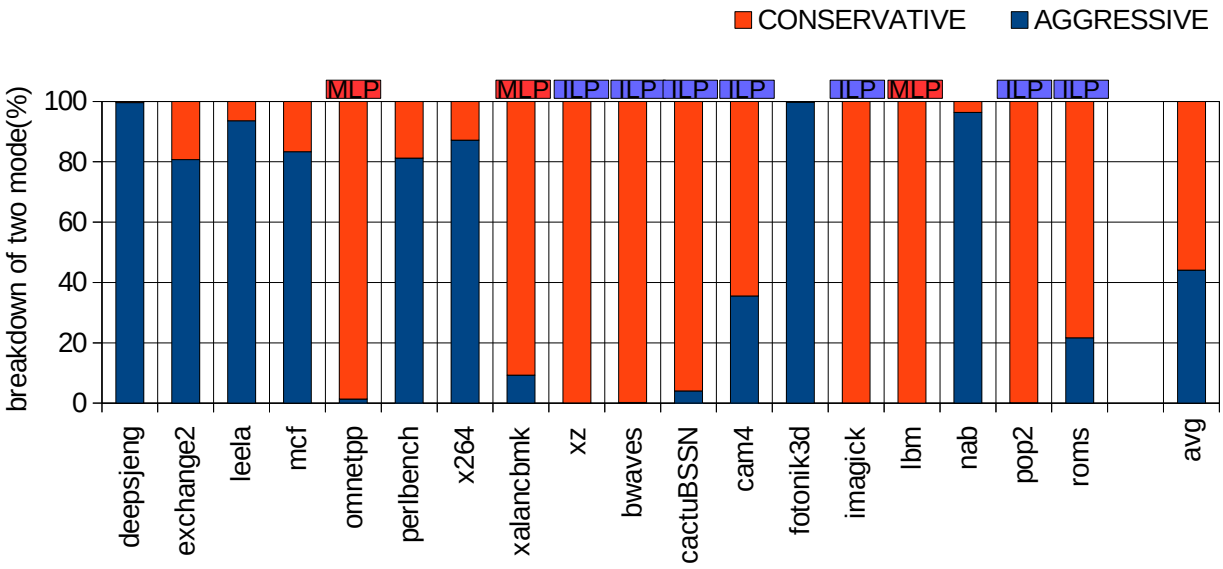

図 6.13: SWITCH 方式におけるモードの割合（8，2）

6.4.3 IPC と LLC MPKI を用いた制御に関する評価

ILP と MLP による制御を同時に行った場合に関して評価を行う．SWITCH 方式において，上記で決定したしきい値を用いて制御を行った際の，AGGRESSIVE モードと CONSERVATIVE モードで実行される割合を図 **??**に示す．

同図より，ILP 及び MLP の高いベンチマークは CONSERVATIVE モードの割合が多く，その他のベンチマークにおいては AGGRESSIVE モードの割合が高いことがわかる．したがって，容量効率の重要性に応じて適切なモードを選択できていると言える．

6.4.4　IPC 及び LLC MPKI のしきい値に対する提案手法の敏感性の評価

しきい値を多少変化させても，性能低下やタグ比較回数が大きく変化しないことが望ましい．そこで，IPC と LLC MPKI のしきい値を変化させた場合の，性能低下とタグ比較回数に関して評価を行う．

IPC のしきい値を変化させた場合の評価

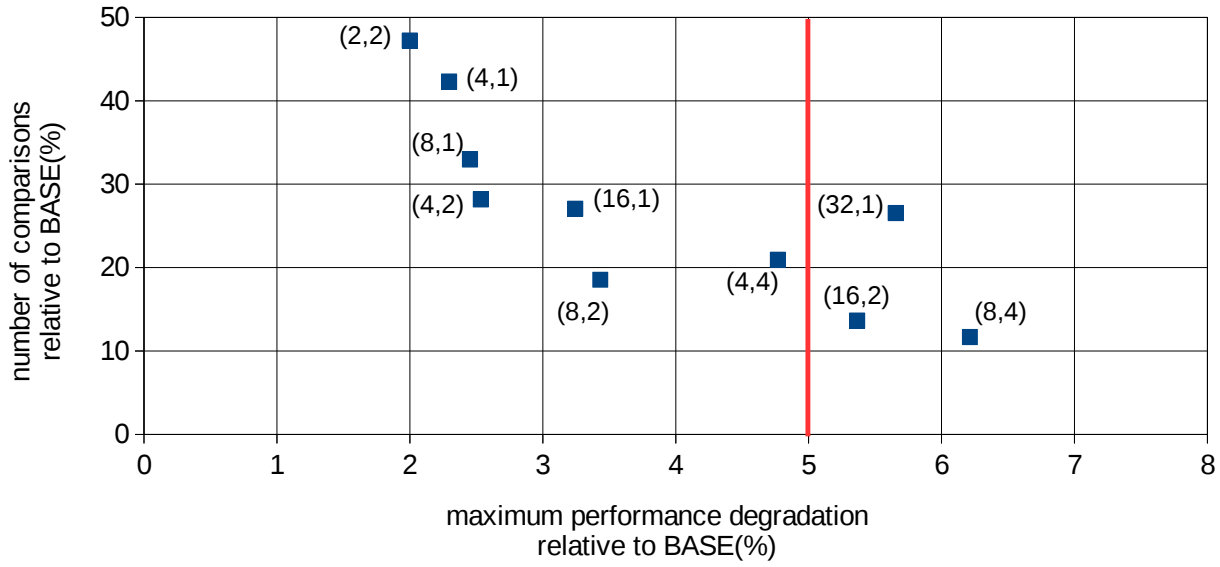

図 6.14: セグメント数の違いによるタグ比較回数と性能低下の散布図

6.5　セグメントの分割数に関する評価

セグメントの分割数を変化させて提案手法を評価し，最適であるセグメントの分割数を決定する．評価の基準としては，提案手法において次の条件を満たすセグメント数の組み合わせを最適と判断する．

- 性能低下が全ベンチマークで 5% 以下
- タグ比較回数の平均が最も少ない

図 6.14 に，メイン・セグメント数とサブ・セグメント数を変化させた場合の，SWITCH 方式での性能低下とタグ比較回数の散布図を示す．図中の各点にはメイン・セグメント数とサブ・セグメント数を（メイン・セグメント数，サブ・セグメント数）という形式で付与している．横軸は最も性能低下が大きかったベンチマークにおける性能低下を示し，縦軸は全ベンチマークでの平均のタグ比較回数を示す．

同図より，セグメントの総数が増えると，タグ比較回数はより削減されるが，同時に性能低下も大きくなることがわかる．特に，(8，4）や（16，2）など，セグメントの総数が 32 を超えると，性能低下率が急激に増加している．

図 6.14 において性能低下が 5% より小さいセグメント数の組み合わせのうち，最もタグ比較回数の少ない（8，2）を最適が最適な組み合わせである．このときのタグ比較回数は 18%（82% 削減）となる．

また，サブ・セグメントを使用しない場合に最適な組み合わせは（16，1）であり，タグ比較回数は 25%（75% 削減） となっている．

第7章　まとめ

LSIの微細化の進展に伴って，経年劣化が加速し摩耗故障が増加する問題が深刻になっている．この故障は，デバイスの温度に関して指数関数的に加速するため，チップ内のホット・スポットの解消が求められている．

発行キューはこのホット・スポットの 1 つとして知られている．この主な原因はウェイクアップ時の多数のタグ比較である．本論文では，ウェイクアップ時のタグ比較回数を削減するために，発行キューをセグメント化，および，それに関わるいくつかの手法を提案した．

提案手法には発行キューの容量効率が低下するという問題点が存在する．この問題点に対して，本論文ではさらに，異なる 2 つのディスパッチ・アルゴリズムを，性能についての容量効率の重要性に応じて切り替えて使用することにより，容量効率の低下による性能低下を抑制する手法を提案した．提案手法を SPEC CPU 2017 を使って評価したところ，性能低下を最大でも 5% 以下に抑えつつ，タグ比較回数を 82% 削減できることを確認した．

発表実績

- 森健一郎, 安藤秀樹, “容量効率を意識したソース・タグ値に基づくセグメント化による発行キューのエネルギー削減”, 情報処理学会研究報告, Vol.2020-ARC-241, No.3, pp.1-12, 2020 年 7 月

謝辞

本研究を進めるにあたり，多大なる御指導と御鞭撻を賜わりました名古屋大学大学院工学研究科 情報・通信工学専攻 安藤秀樹教授に心より感謝いたします．また，本研究の遂行を支えてくださいました，名古屋大学大学院工学研究科情報・通信工学専攻安藤研究室の諸氏に深く感謝します．

参考文献

[1] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective, 4th edition.* Addition Wesley, 2010.

[2] F. Monsieur, E. Vincent, D. Roy, S. Bruyre, G. Pananakakis, and G. Ghibaudo, "Time to breakdown and voltage to breakdown modeling for ultra-thin oxides (Tox<32Å)," in *Proceedings of the 2001 IEEE International Integrated Reliability Workshop*, October 2001, pp. 20–25.

[3] S. Khan and S. Hamdioui, "Temperature dependence of NBTI induced delay," in *Proceedings of the 2010 IEEE 16th International On-Line Testing Symposium*, July 2010, pp. 15–20.

[4] J. Black, "Electromigration–a brief survey and some recent results," *IEEE Transactions on Electron Devices*, vol. ED-16, no. 4, pp. 338–347., April 1969.

[5] R. Viswanath, V. Wakharkar, A. Watwe, and V. Lebonheur, "Thermal performance challenges from silicon to systems," *Intel Technology Journal*, vol. 4, no. 3, pp. 1–16, August 2000.

[6] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-effective superscalar processors," in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, June 1997, pp. 206–218.

[7] J. Stark, M. D. Brown, and Y. N. Patt, "On pipelining dynamic instruction scheduling logic," in *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, December 2000, pp. 57–66.

[8] M. Goshima, K. Nishino, T. Kitamura, Y. Nakashima, S. Tomita, and S. Mori, "A high-speed dynamic instruction scheduling scheme for superscalar processors," in *Proceedings of the 34th Annual International Symposium on Microarchitecture*, December 2001, pp. 225–236.

[9] P. G. Sassone, J. Rupley II, E. Brekelbaum, G. H. Loh, and B. Black, "Matrix scheduler reloaded," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, June 2007, pp. 335–346.

[10] A. R. Lebeck, J. Koppanalil, T. Li, J. Patwardhan, and E. Rotenberg, "A large, fast instruction window for tolerating cache misses," in *Proceedings of the 29th Annual International Symposium on Computer Architecture*, May 2002, pp. 59–70.

[11] S. E. Raasch, N. L. Binkert, and S. K. Reinhardt, "A scalable instruction queue design using dependence chains," in *Proceedings of the 29th Annual International Symposium on Computer Architecture*, May 2002, pp. 318–329.

[12] I. Kim and M. H. Lipasti, "Macro-op scheduling: Relaxing scheduling loop constraints," in *Proceedings of the 36th Annual International Symposium on Microarchitecture*, December 2003, pp. 277–289.

[13] D. Gibson and D. A. Wood, "Forwardflow: A scalable core for power-constrained CMPs," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, June 2010, pp. 14–25.

[14] H. Ando, "SWQUE: A mode switching issue queue with priority-correcting circular queue," in *Proceedings of the 52nd Annual International Symposium on Microarchitecture*, October 2019, pp. 506–518.

[15] Y. Kora, K. Yamaguchi, and H. Ando, "MLP-aware dynamic instruction window resizing for adaptively exploiting both ILP and MLP," in *Proceedings of the 46th Annual International Symposium on Microarchitecture*, December 2013, pp. 37–48.

[16] D. Folegnani and A. González, "Energy-effective issue logic," in *Proceedings of the 28th Annual International Symposium on Computer Architecture*, June 2001, pp. 230–239.

[17] D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources," in *Proceedings of the 34th Annual International Symposium on Microarchitecture*, December 2001, pp. 90–101.

[18] D. Ernst and T. Austin, "Efficient dynamic scheduling through tag ellimination," in *Proceedings of the 29th Annual International Symposium on Computer Architecture*, May 2002, pp. 37–46.

[19] A. Sembrant, T. Carlson, E. Hagersten, D. Black-Shaffer, A. Perais, A. Seznec, and P. Michaud, "Long Term Parking (LTP): Criticality-aware resource allocation in ooo processors," in *Proceedings of the 48th International Symposium on Microarchitecture*, December 2015, pp. 334–346.

[20] H. Homayoun, A. Sasan, J. Gaudiot, and A. Veidenbaum, "Reducing power in all major CAM and SRAM-based processor units via centralized, dynamic resource size management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 11, pp. 2081–2094, Nov 2011.

[21] 小林誠弥, "発行キューにおけるタグの 2 段階比較による消費エネルギー削減," 名古屋大学大学院工学研究科博士課程 (前期課程), 修士学位論文, 2015 年 3 月.

[22] 松田　康誉, “ランダム発行キューにおけるタグの 2 段階比較による電力削減,” 名古屋大学大学院工学研究科博士課程 (前期課程), 修士学位論文, 2020 年 3 月.

[23] J. A. Farrell and T. C. Fischer, “Issue logic for a 600-mhz out-of-order execution microprocessor,” *IEEE Journal of Solid-State Circuits*, vol. 33, no. 5, pp. 707–712, 1998.

[24] M. Butler and Y. Patt, “An investigation of the performance of various dynamic scheduling techniques,” in *Proceedings of the 25th Annual IEEE/ACM International Symposium on Microarchitecture*, December 1992, pp. 1–9.

[25] J. Abella, R. Canal, and A. Gonzalez, “Power- and complexity-aware issue queue designs,” *IEEE Micro*, vol. 23, Issue 5, no. 5, September-October 2003.

[26] R. P. Preston, R. W. Badeau, D. W. Bailey, S. L. Bell, L. L. Biro, W. J. Bowhill, D. E. Dever, S. Felix, R. Gammack, V. Germini, M. K. Gowan, P. Gronowski, D. B. Jackson, S. Mehta, S. V. Morton, J. D. Pickholtz, M. H. Reilly, and M. J. Smith, “Design of an 8-wide superscalar RISC microprocessor with simultaneous multithreading,” in *2002 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, February 2002, pp. 334–472.

[27] M. Golden, S. Arekapudi, and J. Vinh, “40-entry unified out-of-order scheduler and integer execution unit for the AMD Bulldozer x86-64 core,” in *2011 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, February 2011, pp. 80–82.

[28] B. Sinharoy, J. A. V. Norstrand, R. J. Eickemeyer, H. Q. Le, J. Leenstra, D. Q. Nguyen, B. Konigsburg, K. Ward, M. D. Brown, J. E. Moreira, D. Levitan, S. Tung, D. Hrusecky, J. W. Bishop, M. Gschwind, M. Boersma, M. Kroener, M. Kaltenbacha, T. Karkhanis, and K. M. Fernsler, “IBM POWER8 processor core microarchitecture,”

IBM Journal of Research and Development, vol. 59, issue 1, pp. 2:1 – 2:21, January - February 2015.

[29] M. Motomura, J. Toyoura, K. Hirata, H. Ooka, H. Yamada, and T. Enomoto, "A 1.2-million transistor, 33 mhz, 20-bit dictionary search processor with a 160 kb CAM," in *1990 37th IEEE International Conference on Solid-State Circuits*, 1990, pp. 90–91.

[30] ——, "A 1.2-million transistor, 33-mhz, 20-b dictionary search processor (DISP) ULSI with a 160-kb CAM," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 5, pp. 1158–1165, 1990.

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

[32] D. A. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings of Seventh International Symposium on High-Performance Computer Architecture*, January 2001, pp. 197–206.