プログレスレポート

森健一郎
配布対象：安藤先生

2019 年 12 月 10 日

1　研究目的

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，プロセッサの故障を引き起こす可能性が高い．[1–5] 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 (IQ:issue queue) がある．IQ 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，IQ の電力削減に対する要求は非常に大きい．

IQ の中で最も電力を消費するのは，タグ比較の回路である．タグ比較は，発行幅分のディスティネーション・タグとすべてのソース・タグで行われるため，電力効率が非常に悪い．そこで本研究では，ディスティネーション・タグとソース・タグの下位ビットが等しい命令についてのみ比較器を動作させることにより，動作する比較器の数を減少させ電力を削減する方法を提案する．

提案手法は，次のように実現する．IQ を複数のセグメントに分割し，第 n セグメントには，第 1 ソース・タグの下位ビットが n である命令のみをディスパッチする．そして，ウェイクアップのタグ比較の際には，ディスティネーション・タグの下位ビットが，自身に割り当てられた命令の第 1 ソース・タグの下位ビットと等しいセグメントのみ，比較器を動作させてタグ比較を行う．この方法により，第 1 ソース・タグについての比較器の動作回数を「1/セグメント数」に減少させることができる．

提案手法における欠点として，セグメントが詰まることによる性能低下が挙げられる．あるセグメントに空きがない状態で，そのセグメントにディスパッチされる命令が現れた場合を考える．この場合，他のセグメントにディスパッチすることはできないため，該当のセグメントに空きが出るまでディスパッチを停止する必要があり，これは性能低下に繋がる．本研究では，この欠点に対する対応策を考え，性能低下が許容できる範囲内に収まるようにする必要がある．

また，その他の欠点として，第 2 ソース・タグの比較器の動作回数は削減できないことなどが挙げられる．これらの欠点に対しても十分に検討し，提案手法における電力削減及び性能の変化について評価を行う．

2　経過

2.1　前回の経過

- 提案手法の修正及び評価
- レジスタ・フリーリストのセグメント化の実装及び評価

2.2　今回の経過

- 提案手法の修正及びその評価
- Swap 方式の改良及びその評価
- レジスタ・フリーリストのセグメント化の修正及びその評価
- 占有率が低下しても性能に影響を与えない原因の調査

3 活動報告

3.1 提案手法の修正及びその評価

3.1.1 提案手法の修正点

前回レポート時までの提案手法では，第 1 ソース・オペランドがレディの場合，それがゼロ・レジスタである場合を除いて，第 1 ソース・タグ値の下位ビットを用いて割り当てるセグメントを決定していた．しかし第 1 ソース・オペランドがレディの場合，タグ比較は行われないため，実際はどのセグメントにディスパッチしても問題ない．

そこで，第 1 ソース・オペランドがレディの場合には，ソース・タグ値に基づきセグメントを決定するのではなく，特定のアルゴリズムによってディスパッチを行うセグメントを決定するように修正した．これにより，第 1 ソース・オペランドがレディで，かつ第 1 ソース・タグ値より決定されたセグメントに空きがない場合にディスパッチがストールすることがなくなり，容量効率の低下が改善されると期待できる．(以降では，ソース・オペランドがレディでどのセグメントにディスパッチしても問題のない状態を，セグメント・フリーと呼ぶこととする)

なお，前回のレポートで第 1 ソース・オペランドを使用しない命令としていた命令は，すべて第 1 ソース・オペランドがレディである命令に含むこととする．また，以下のアルゴリズムをセグメント・フリーの場合の割当アルゴリズムとして使用した．

- ROUNDROBIN : 割当を行うセグメントをラウンドロビンで決定する
- MAX:最も空きの多いセグメントに割当を行う

提案手法におけるセグメントの割当方についてまとめる．

1. 第 1 ソース・オペランドがレディ : セグメント・フリーであるため，割当アルゴリズムに基づいてセグメントを決定
2. 第 1 ソース・オペランドがレディでない : 第 1 ソース・タグ値の下位ビットよりセグメントを決定．該当のセグメントに空きがない場合はストール

表 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	256 entries
IQ	128 entries
Load/Store queue	128 entries
Physical registers	256(int) + 256(fp)
Branch prediction	12-bit history 4K-entry PHT gshare 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 16B/cycle bandwidth
Prefetch	stream-based,32-stream tracked, 16-line distance, 2-line degree, prefetch to L2 cache

3.1.2 評価環境

評価環境について説明する．シミュレータには SimpleScalar をベースに修正を加えたものを使用した．表 1 にプロセッサ構成を示す．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．ベンチマークの測定区間は，プログラムの先頭から 16G 命令をスキップした後の 100M 命令である．

3.1.3 評価モデル

評価モデルとしては，以下の 4 つを用いた．

- base : 通常の発行キューを使用したモデル
- old : 前回レポート時点での提案手法を使用したモデル
- roundrobin : 修正した提案手法において，セグメント・フリーの場合に ROUNDROBIN でセグメントを決定するモデル
- max : 修正した提案手法において，セグメント・フリーの場合に MAX でセグメントを決定するモデル

3.1.4 修正した提案手法の評価

修正を加えた提案手法に関して，改めてタグ比較回数の削減，性能変化，容量効率に関する測定を行った．測定結果を，図 1， 2， 3， 4 に示す．

■占有率 まず, 図 4 の占有率に関して考える．old と比較すると，roundrobin と max では，占有率が僅かに上昇している．これは，レディである reg1 に該当するセグメントに空きがない場合にストールを防ぐという今回の修正の目的が達成できているためであると考えられる．base と roundrobin，max を比較してみると，平均で 10 %pt 程度占有率が低下していることが分かる．従って，容量効率を低下させないという観点から見ると，提案手法にはまだ改善の余地があると言える．(だが，スワップを使用しないという前提で考えると，これ以上改善することは難しいように思える．)

■タグ比較の削減率 続いて図 1， 2 のダグ比較の削減率に関して考える．図の大きさの都合上，int 系と fp 系に分割してある．図の横軸は，各ベンチマークにおいて左から old，roundrobin，segment モデルでのタグ比較の削減率を表したグラフとなっている．また, 削減率のうち, segment reduction(緑色の部分) がセグメント化による削減を表し，stall reduction(赤色の部分) が, ストールすることによって生じる削減を表す．両者をたした値がトータルの削減率となる．

図より，どのベンチマークでも修正した提案手法では old に比べて僅かに削減率が低下していることが分かる．これは，修正した提案手法では占有率が上昇したという結果と一致する．

提案手法の純粋な効果をあらわす segment reduction に関して考える．平均で見ると，理論値である 47% に近い値を表していることが分かる．従って，提案手法による削減は期待したとおりに動作していると言える．ベンチマークによっては理論値と少しズレが生じているが，これは理論値を求める際に第 1 ソース・オペランドと第 2 ソース・オペランドのタグ比較の回数が同じであるという仮定をおいているためである．実際には，ベンチマークごとに偏りがあるため，それが原因でベンチマークごとに理論値とずれが生じる．

続いてストールが原因で起こる削減を表す stall reduction について考える．stall reduction の値はベンチマークによって大きく異なり，小さいものだと 0% に近いが，大きいものだと 20% 程度となる．この値は，図 4 における base からの低下率と関係していて，base に対して大きく占有率が低下するベンチマークでは stall reduction の値も大きくなっている．これは，ストール回数が多く占有率が低下すると，それだけ比較器の動作回数も小さくなるという直感と一致した結果となっている．

■性能変化 次に図 3 の性能変化について考える．性能に関してはこれまでの評価と同様に，提案手法によって大きく低下していないことが分かる．

■raundrobin と max の比較 raundrobin と max に関して比較する．raundrobin の占有率は，max と比較して平均で 2% pt 程度と僅かに小さい値を示している．その結果，削減率においても 3% 程度 max よりも小さい値となっている．なお，これはほとんど stall reduction の差となっており，segment reduction は同じであると考えて良い．

max は，全セグメントの中から最も空き数の多いセグメントを選択する回路が必要である．2%pt 程度の占有率の違いであれば，簡単な回路で実現できると思われる roundrobin のほうが良いのではないかと考えている．従って，以降の測定では，セグメント・フリーの場合は raundrobin を使用することとする．

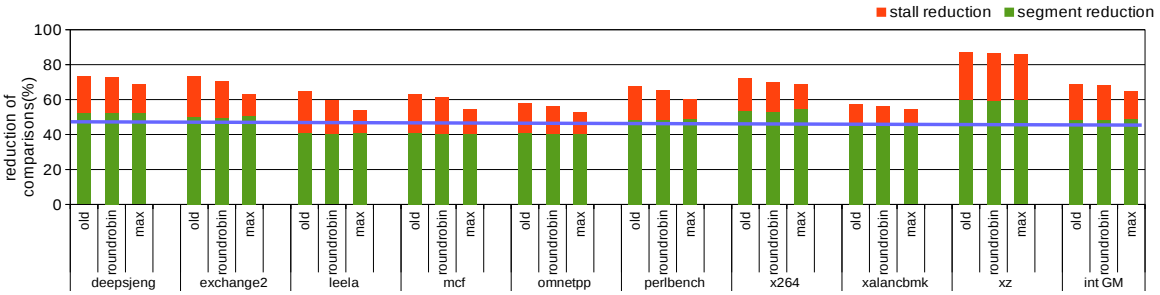

図 1　提案手法による比較器の動作回数削減率 (int 系)

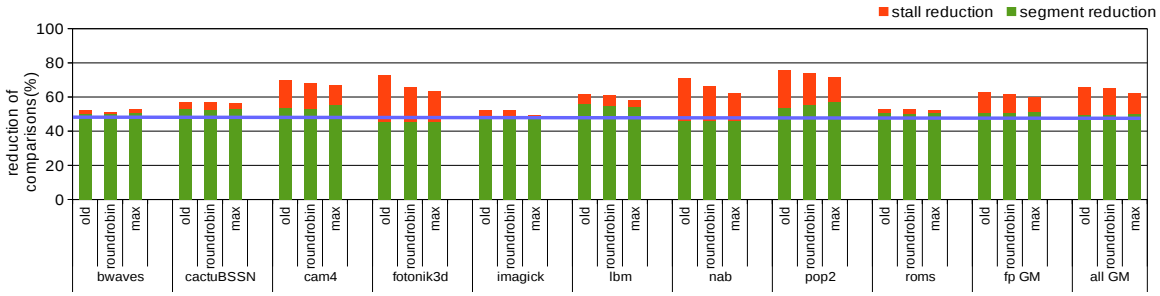

図 2　提案手法による比較器の動作回数削減率 (fp 系)

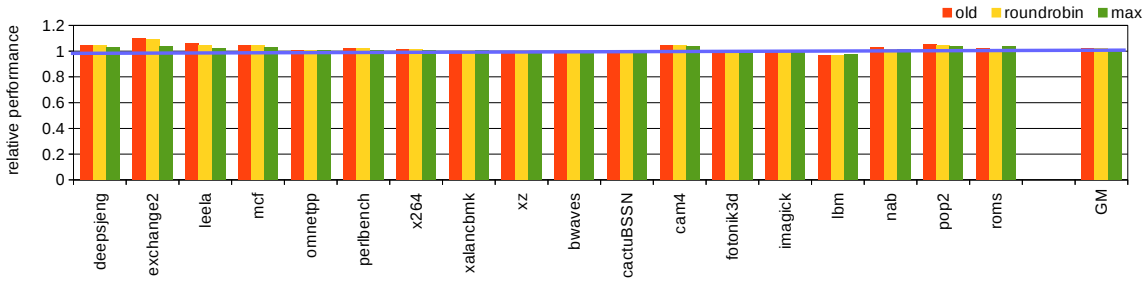

図 3　提案手法による IPC の変化

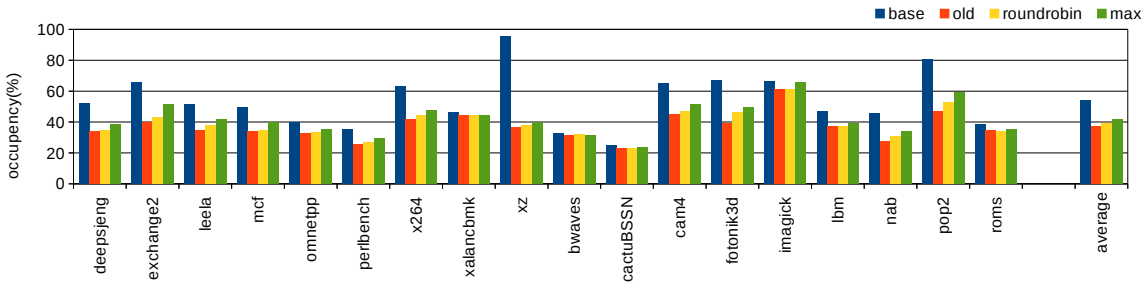

図 4　提案手法における IQ の占有率

3.2 Swap 方式の改善及び評価

3.2.1 Swap 方式の細分化とバリエーション

Swap 方式は第 1 ソース・オペランドがレディであり，第 2 ソース・オペランドがレディでない場合に第 1 ソース・タグと第 2 ソース・タグを交換するというものであった．今回はこの Swap 方式を，よりタグ比較の削減を重視した Swap_Aggresive と，より容量効率を重視した Swap_Conservative の 2 つの方式に細分化した．

また，現在の Swap 方式では，ソース・オペランドが両方共レディでない場合には，第 1 ソース・タグ値を使用してディスパッチするセグメントを決定している．もし，第 1 ソース・タグ値によって決定するセグメントに空きがない場合に，スワップを行い第 2 ソース・タグ値によってセグメントを決定すれば，ストールを回避できるかもしれない．そこで，ソース・オペランドが両方レディでない場合に，セグメントを決定するために使用するタグを選択するような Swap 方式のバリエーションを実装した．これを STDS (Selecting Tag to Dicide Segment) と呼ぶことにする．

以 降 ，Swap_Aggresive，Swap_Conservative，STDS のアルゴリズムについて詳細に説明する．

3.2.2 Swap_Aggresive

Swap_Aggresive は，これまでのレポートでの Swap 方式と同様のものと考えて良い．以下のアルゴリズムに基づきディスパッチするセグメントを決定する．説明の都合上，第 1 ソース・オペランドを reg1，第 2 ソース・オペランドを reg2 と表す．

1. reg1 がレディかつ reg2 もレディ : セグメント・フリーとして割り当てを行う．
2. reg1 がレディでなく reg2 がレディ : reg1 のタグ値を用いてセグメントを決定．該当のセグメントに空きがなければストールする．
3. reg1 がレディで reg2 がレディでない : reg2 のタグ値を用いてセグメントを決定 (スワップ)．該当のセグメントに空きがなければストールする．
4. reg1 がレディでなく reg2 もレディでない : reg1 のタグ値を用いてセグメントを決定．該当のセグメントに空きがなければストールする．

3.2.3 Swap_Conservative

Swap_Conservative は，Swap_Aggresive よりも容量効率を重視したものとなっている．Swap_Aggresive は，片方のオペランドがレディでもう片方がレディでない場合に，レディでない方のタグ値を用いてセグメントを決定し，もし，決定されたセグメントに空きがない場合にはストールしていた．このような場合に，Swap_Conservative では，ストールするのではなくセグメント・フリーとして割当を行い，発行キューの第 1 ソース・タグを格納するフィールドにレディであるタグを入れる．

以下に Swap_Conservative の割当アルゴリズムを示す． 2 番と 3 番での動作が Swap_Aggresive と異なる

1. reg1 がレディかつ reg2 もレディ : セグメント・フリーとして割り当てを行う．
2. reg1 がレディでなく reg2 がレディ : reg1 のタグ値を用いてセグメントを決定．該当のセグメントに空きがなければ，スワップを行いセグメント・フリーとして割当を行う．
3. reg1 がレディで reg2 がレディでない : reg2 のタグ値を用いてセグメントを決定 (スワップ)．該当のセグメントに空きがなければ，スワップをせずにセグメント・フリーとして割当を行う．
4. reg1 がレディでなく reg2 もレディでない : reg1 のタグ値を用いてセグメントを決定．該当のセグメントに空きがなければストールする

Swap_Aggresive と比較した Swap_Conservative の欠点として，タグ比較回数の削減率の低下が挙げられる．これは，アルゴリズムの 2 番と 3 番においてセグメント・フリーで割当を行った場合に，タグ比較の削減が行える第 1 ソース・タグのフィールドにレディなタグが入り，削減の行えない第 2 ソース・タグのフィールドにレディでないタグが入るため，提案手法によるタグ比較の削減が全く行えない

ためである．

3.2.4 STDS (Selecting Tag to Dicide Segment)

Swap_Aggresive，Swap_Conservative のいずれにおいても，アルゴリズムの 4 番，つまりどちらのソース・オペランドもレディでない場合には，reg1 のタグ値を用いてセグメントを決定していた．もし，reg1 によって決定されるセグメントに空きがない場合，スワップして reg2 を使用することによってストールを防ぐことができるかもしれない．そこで，セグメントの空き状況に応じて，reg1 と reg2 のどちらを使用するかを選択できるようにした STDS (Selecting Tag to Dicide Segment) を実装した．STDS により，Swap 方式において容量効率の低下を抑制することが期待できる．

STDS を適応した Swap_Aggresive，Swap_Conservative では，割当アルゴリズムのうち 4 番目が，以下のように変更される．

4. reg1 がレディでなく reg2 もレディでない：reg1 のタグ値を用いてセグメントを決定．該当のセグメントに空きがなければ reg2 のタグ値を用いてセグメントを決定 (スワップ)．該当のセグメントに空きがなければストールする．

3.2.5 評価モデル

Swap 方式の評価を行うため，以下のモデルを使用した．

- base：通常の発行キューを使用したモデル
- normal：通常の提案手法を使用したモデル (Swap を行わない)
- agg：提案手法の Swap_Aggresive 方式を使用したモデル
- agg+STDS：提案手法の Swpa_Aggresive 方式に STDS を適応したモデル
- cons：提案手法の Swap_Conservative 方式を使用したモデル
- cons+STDS：提案手法の Swap_Conservative 方式に STDS を適応したモデル

なお，評価環境は 3.1.2 節と同様である．

3.2.6 改良した Swap 方式の評価

改良した Swap 方式の評価結果を，図 5，6，7，8 に示す．

■占有率 まず，図 8 の占有率に関して考える．はじめに，Swap_Aggressive(agg) 及び Swap_Conservative(cons) について評価する．図より，normal に比べて agg では占有率が僅かに低下し，conf では比較的大きく上昇していることが分かる．これは，Swap_Aggresive がよりタグ比較の削減を行うためのモデルであり，Swap_Conservative がより容量効率を上昇させるモデルであることと一致する結果である．

次に STDS の効果について考える．図より，conf+STDS の場合に，normal や agg と比較して大きく占有率が向上していることが分かる．特に xz ベンチマークや pop2 などでは，normal の際に大きく低下していた占有率が，比較的 base と近い値となっていることが分かる．平均で考えても，normla の場合は base から 10%pt ほど低下していたのに対して，conf+STDS では 4%pt 程度の低下に抑えられている．このことから STDS によって，容量効率が低下するという提案手法の欠点が緩和されていると言える．

■タグ比較の削減率 続いて，図 5，6 のタグ比較の削減率に関して考える．図の横軸は，各ベンチマークにおいて，左から normal，agg，agg+STDS，cons，cons+STDS を表す．

図より Swap_Aggresive(agg) では，normal の場合に比べてより削減率が高くなっていることが分かる．これは，提案手法の純粋な効果を表す segment reduction が増加していることが主な要因であり，normal では削減できていなかった 第 2 ソース・オペランドのタグ比較が Swap 方式によって効果的に削減できていることを示している．なお，agg と agg+STDS を比較すると，僅かに agg+STDS のほうが削減率が低下しているが，ほとんど変わらないと言って良い程度である．

続いて，Swap_Conservative(cons) について考える．平均で見ると cons は agg と比較して，stall reduction が低下していることが分かる．これは，

cons において容量効率が増加しているためであっる．また，cons+STDS は，cons よりも更に stall reduction が低下している．これはもともと容量効率を重視する Swap_Conservative にプラスして STDS を使用しているためである．平均を見てみると，normal の場合よりも削減率が低下してしまっているが，これは stall reduction が大きく低下したためであり，容量効率を低下させないという目的は達成できている．

■性能変化　図 7 の性能変化に関して考える．図より Swap 方式でもほとんどのベンチマークで性能が低下していないことが分かる．唯一 Swap_Aggresive(agg) において imagick が 8% ほど性能が低下している．図 7 を見ると，imagick では agg の時に normal よりも占有率が 20%pt と大きく低下しており，これが性能に影響を与えていると考えられる．

■Swap 方式の評価のまとめ　Swap 方式に関するこれまでの評価をまとめる．

- より積極的にタグ比較を削減したい場合には，Swap_Aggresive 方式を利用すればよく，平均で 85% 程度のタグ比較の削減が可能である
- 発行キューの容量効率を低下させたくない場合には，Swap_Conservative 方式に STDS を適用した方式を用いればよく，占有率の低下を平均 3%pt 程度に抑えつつ，60% 程度のタグ比較の削減が可能ある．
- いずれの場合も性能はほとんど低下しないが，Swap_Aggresive では一部のベンチマークで 10% 弱性能が低下する．

3.3　レジスタ・フリーリストのセグメント化の修正及びその評価

前回のレポートで，レジスタ・フリーリストのセグメント化による発行キューの容量効率向上について検討した．ここで，レジスタ・フリーリストのセグメント化とは次のようなものであった．レジスタ・フリーリストを IQ のセグメント数と同数に分割し，第 n レジスタ・フリーリストには，下位ビットが n であるようなレジスタ・タグのみを保持する．そして，リネームの際には，各レジスタ・フリーリストにラウンドロビンにアクセスしてタグを得る．

レジスタ・フリーリストのセグメント化によって，発行キューの容量効率が向上すると考えられていたが，結果はセグメント化しない場合とほとんど同じとなり，期待した効果は得られなかった．この原因は，そもそもリネームされるタグには特に制限がないため，セグメント化せずとも発行キューの各セグメントに対して均等に割り当てられていたためである．

そこで今回は，ラウンドロビンでレジスタ・フリーリストのセグメントにアクセスするのではなく，発行キューのセグメントのうち最も空き数の多いセグメントに対応するレジスタ・フリーリストのセグメントからタグを供給するように変更した．例えば，4 つに分割された発行キューのセグメントにおいて，それぞれの空き数が segment0 = 10，segment1= 8，segment2 = 12，segment3 = 5 であった場合には，もっとも空き数の多い segment2 に対応する，レジスタ･フリーリストの第 2 segment からタグを供給する．

これによって，そのタグが表すレジスタをソース・オペランドとするような命令は，空きの多いセグメントに割り当てられるため，セグメント内の命令数が均等になると期待できる．

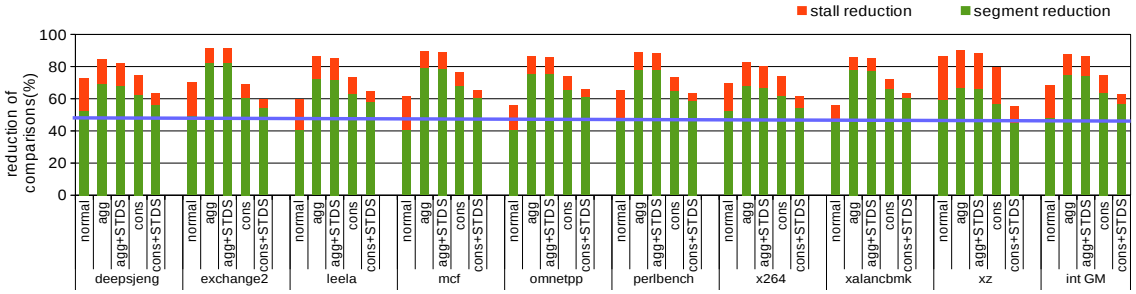

図 5　Swap 方式による比較器の動作回数削減率 (int 系)

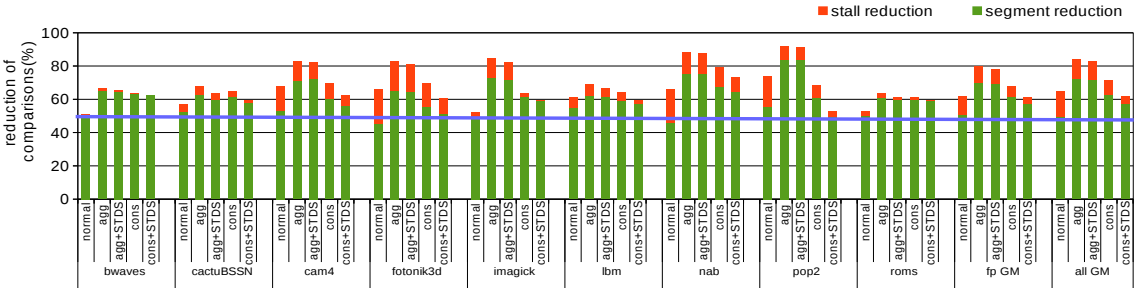

図 6　Swap 方式による比較器の動作回数削減率 (fp 系)

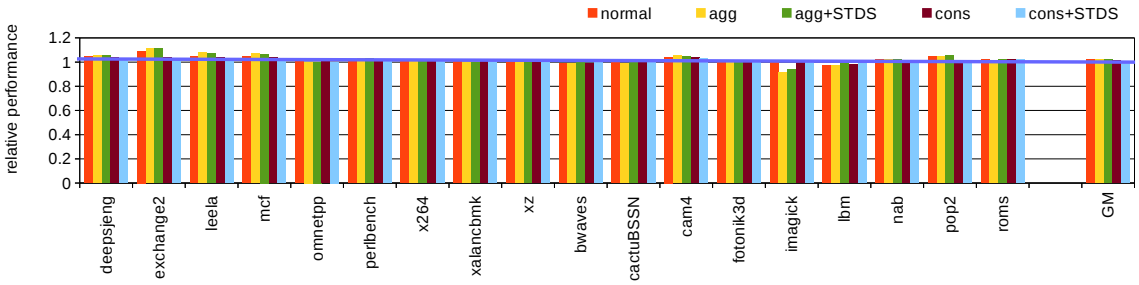

図 7　Swap 方式による IPC の変化

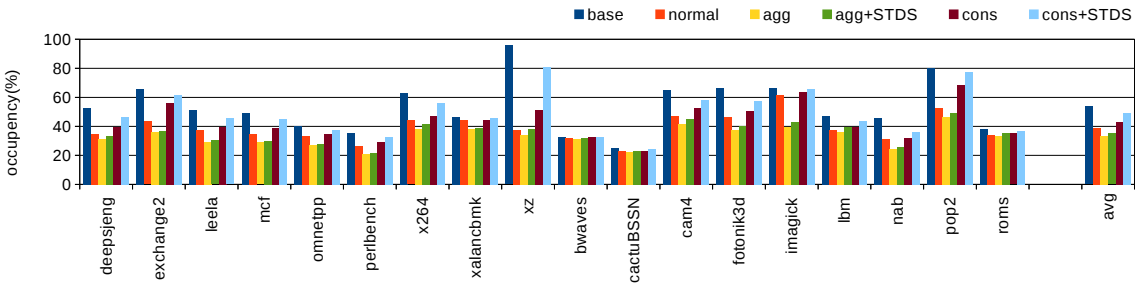

図 8　Swap 方式での IQ の占有率

3.3.1 評価モデル

修正したレジスタ・フリーリストのセグメント化に関して評価を行うため，以下の評価モデルを使用した．

- base：通常の発行キューを使用したモデル
- normal：通常の提案手法を使用したモデル (Swap を行わない)
- normal+reg：提案手法にレジスタ・フリーリストのセグメント化を適応したモデル
- cons+STDS：提案手法の Swpa_Conservative 方式に STDS を適応したモデル
- cons+STDS+reg：cons+STDS にレジスタ・フリーリストのセグメント化を適応したモデル

なお，今回の評価では，Swap_Aggresive 方式に関しては評価を行っていない．これは，Swap_Aggresive ではタグ比較の削減を優先することを目的としていて，容量効率を上げることは重要ではないためである．

評価環境は 3.1.2 と同様である．

3.3.2 修正されたレジスタ・フリーリストのセグメント化の評価

評価結果を，図 9， 10， 11， 12 に示す．

■占有率 図 12 の占有率に関して考える．normal と normal+reg を比較すると，すべてのベンチマークで normla+reg のほうが占有率が高く，平均では 5%pt 高くなっていることが分かる．従って，レジスタ・フリーリストのセグメント化によって，発行キューの容量効率が上昇していると言える．

cons+STDS と cons+STDS+reg を比較してみても，殆どのベンチマークで占有率が上昇しているが，その上昇幅は normal の時と比較すると小さい．この理由としては，cons+STDS の時点で占有率は Base に近い値を示しており，占有率が上昇する余地がそもそもないためであると考えられる．

■タグ比較の削減 図 9， 10 のタグ比較の削減に関して考える．normal と normal+reg を比較すると， stall reduction が小さくなっていることが分かる．これは，占有率が上昇したという結果と一致する．また，cons+STDS と cons+STDS+reg を比較してみると，segment reduction が増加し，結果として削減率が 5%pt 程度増加していることが分かる．

■性能変化 図 11 の性能変化に関して考える．図より，レジスタ・フリーリスををセグメント化しても性能はほとんど変化しないことが分かる．

■レジスタ・フリーリストのセグメント化のまとめ これまでの評価を，以下のようにまとめる．

- 通常の提案手法にレジスタ・フリーリストのセグメント化を適応した場合，占有率が 5%pt 程度向上し，容量効率が改善する．
- Swap_Conservative+STDS にレジスタ・フリーリストのセグメント化を適応した場合，占有率はほとんど変化しないが，セグメント化によるタグ比較の削減率が 5%pt ほど上昇する．

特に normal の場合の占有率を下げることができるため，レジスタ・フリーリストのセグメント化は有向であるといえる．

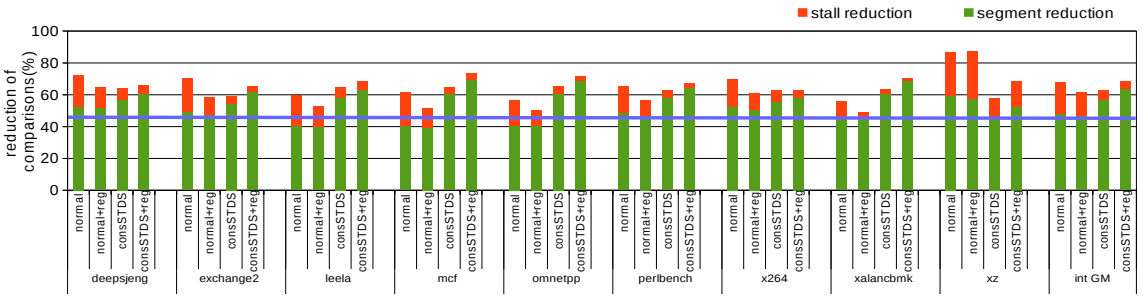

図 9　レジスタ・フリーリストのセグメント化による比較器の動作回数削減率 (int 系)

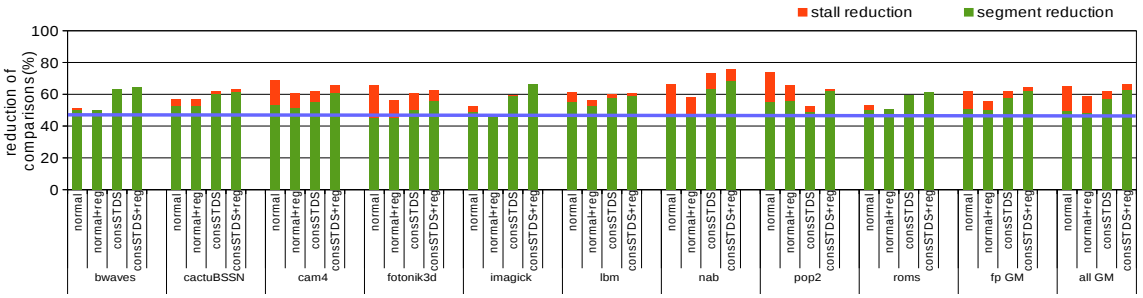

図 10　レジスタ・フリーリストのセグメント化による比較器の動作回数削減率 (fp 系)

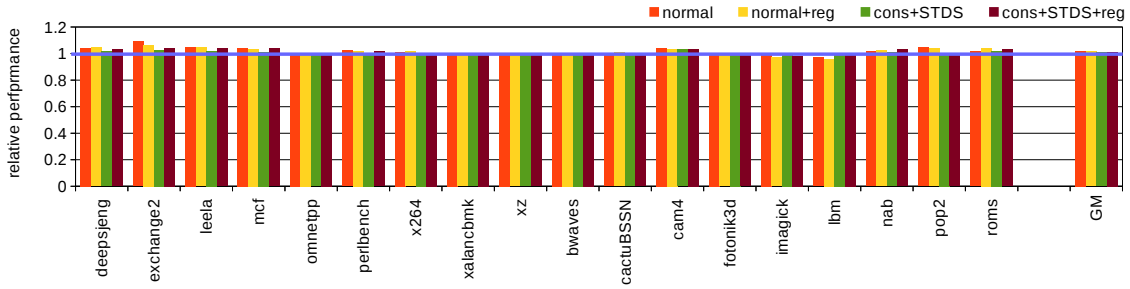

図 11　レジスタ・フリーリストのセグメント化による IPC の変化

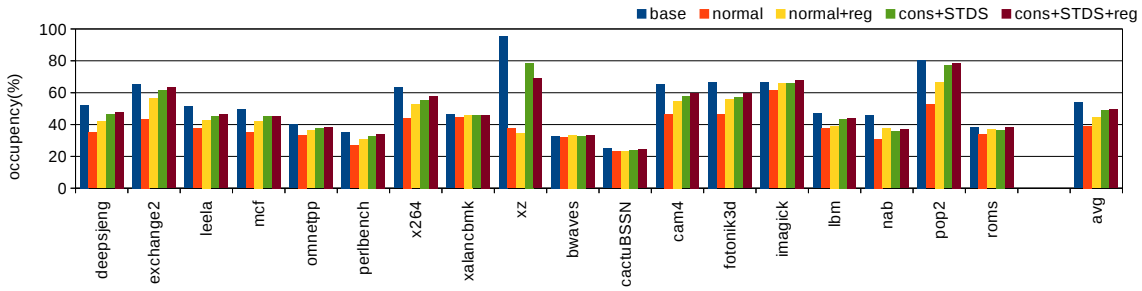

図 12　レジスタ・フリーリストのセグメント化による IQ の占有率

3.4 IPC が低下しない原因の考察

これまでの評価で示したように，提案手法では発行キューの容量効率は低下するが，性能は低下しない．しかしベンチマークの中には容量効率が性能に対して重要なものがいくつかある．それらのベンチマークでも性能が低下しない原因に関していくつか考察する．

表 2　容量効率の重要性によるベンチマークの分類

大	omnetpp, xalancbmk, bwabes, cactuBSSN, roms
中	mcf, x264, lbm
小	deepsjeng, exchange2, leela, perlbench, xz, cam4, fotonik3d, imagik, nab, pop2

3.4.1 ベンチマークの分類

安藤先生に頂いた資料より，SPEC CPU 2017 ベンチマークを，容量効率が性能に与える影響の大きさに応じて分類した．その分類を 2 に示す．容量効率の低下によって性能が低下すると考えられるのは，表中の大に該当するベンチマークである．

ここで，大に該当するベンチマークについて図 4 に示された通常の提案手法での占有率の低下を見てみる．大に該当するベンチマークは，Base に対してほとんど占有率が低下していないことが分かる．（平均で 2.5%pt の低下）従って，容量効率の影響が大きいベンチマークでは，そもそも占有率がほとんど低下していないため性能低下が生じていないと言える．

4 研究計画

- 提案手法によって性能が低下しない原因を更に詳しく調べる
  - ストールしている時の発行キューの占有率
  - 1 つの命令が連続してストールする平均サイクル数
- HSPAICE を用いた電力測定を行う
  - 測定方法などを松田さんから教わる

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, p. 116, August 2000.