プログレスレポート

森健一郎
配布対象：安藤先生

2020 年 3 月 31 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

- ROB のサイズによる提案手法への影響の調査
- Last Tag Prediction の実装及び評価

2.2　今回の経過

- Last Tag Prediction(LTP) に GHB 方式を実装
- セグメントを第 2 ソースタグに基づき更に分割するサブ・セグメントを実装

3 活動報告: g-share 方式の LTP の実装及び評価

前回のレポートで，両方のタグがレディでない場合に，より遅くにレディとなるタグを予測して，そのタグを発行キューのセグメント化された方に入れるという Last Tag Prediction(LTP) を実装した．しかし，前回レポート時点では LTP を構成するテーブルのインデクスに単純な PC のみを用いていたため，予測精度が低くなっていた．

LTP が提案された論文 [6] によると，テーブルのインデクスには，PC とグローバル分岐履歴の XOR を用いる g-share 方式が採用されている．そこで今回は， g-share 方式を実装し，予測精度の測定を行った．g-share 方式では，図 1 のように，テーブルのインデクスを PC の下位ビットとグローバル分岐履歴の XOR によって計算する．なお，グローバル分岐履歴は分岐予測ミスのリカバー時に正しい履歴に書き換えるようにしている．また，グローバル分岐履歴は論文 [6] に基づき 8 bit としている．

g-share 方式の LTP をシミュレータに実装し，予測精度の評価を行った．評価環境とその結果を以下に示す．

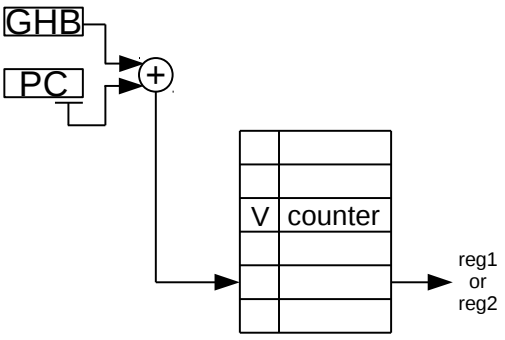

図 1 Last Tag Predictor(g-share)

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	(Small) 256 entries (Medium) 320 entries (Huge) 1024 entries
IQ	128 entries
Load/Store queue	128 entries
Physical registers	(Small) 256(int) + 256(fp) (Medium) 320(int) + 320(fp) (Huge) 1024(int) + 1024(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 8B/cycle bandwidth
Prefetch	stream-based,32-stream tracked, 16-line distance, 2-line degree, prefetch to L2 cache

3.1 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． Reorder buffer 及び Physical registers の構成として，Small，Medium，Huge の 3 つのモデルを想定している．これまでの測定では Small モデルを用いていたが，以降の測定では，基本として Medium モデルを使用して行う．これは， ROB が詰まることを防ぎ，発行キューの評価を適切なものにするためである．

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．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．

3.2 評価結果

g-share 方式の LTP による予測精度について，図 2 に測定結果を示す．横軸がベンチマーク，縦軸

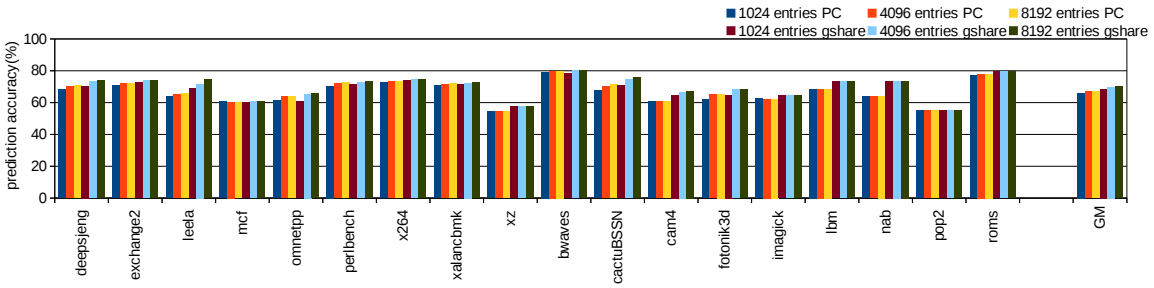

図 2　LTP の予測精度

が予測精度を表す．また，各ラベルは LTP のエントリ数及びインデクスの方法が PC か g-share 方式かを表している．

同図より，g-share 方式での測定精度は，エントリ数が同じ場合で PC の 2% 程度しか向上しておらず，大きな向上は見られないことが分かる．これはどのベンチマークでも同様である．従って， g-share 方式での LTP の有効性は PC 方式の場合とほぼ同程度であるといえる．

ただ，論文 [6] によると，g-share 方式による LTP の予測精度は 8 から 9 割程度とされているのに対して，今回の測定では 7 割程度の精度にとどまっている．評価環境が異なるので一概には言えないが，もしかすると論文で想定されている予測方法と今回の実装で異なる部分がある可能性がある．もう少し改良の余地がないか探ろうと思う．

なお，g-share 方式の場合の性能及び比較回数の削減率に関しては，前回レポートで示した PC を用いた場合の LTP と優位な差が見られなかったため，本レポートでは省略している．

4　活動報告：サブ・セグメントの実装と評価

現在の提案手法の欠点として，第 1，2 ソース・レジスタがともにレディでない場合に，第 2 ソース・レジスタでのタグ比較を削減できないことが挙げられる．この欠点を解消するため，サブ・セグメント方式を提案する．

4.1　サブ・セグメント方式の概要

サブ・セグメントは図 3 のように，第 1 ソース・タグ値に基づいて分割された各セグメント (図中の黒枠) を，第 2 ソース・タグ値に基づいて更に分割する手法である．図の例では，4 つに分割したセグメントを，それぞれさらに 2 つのサブ・セグメントに分割している．

なお以降の説明では，第 1 ソース・タグ値に基づく通常のセグメントを，サブ・セグメントと対比してメイン・セグメントと呼ぶ．また，図 3 中の (a,b) で表される番号は，a がメイン・セグメントの番号を，b がサブ・セグメントの番号を表している．

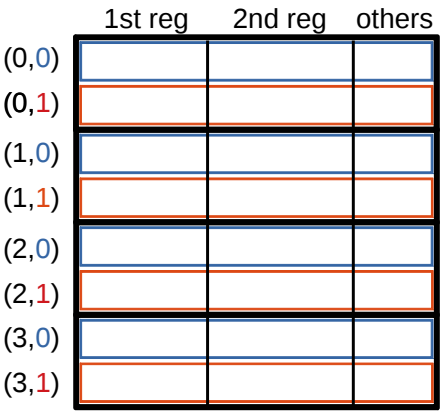

図 3　サブ・セグメントを実装した IQ

4.1.1 サブ・セグメント方式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ついて，ソース・レジスタのレディ状況ごとに例を用いて説明する．以下の例で用いる発行キューは，メイン・セグメント数が 4，サブ・セグメント数が 2 (図 3 と同様) であるとする．

■1: reg1，reg2 ともにレディでない場合　r2 = r6 + r3 という命令について考える．r6，r3 はともにレディでないとする．第 1 ソース・タグより，メイン・セグメントは 2(=6 % 4) となる．また，第 2 ソース・タグより，サブ・セグメントは 1(= 3 % 2) となる．したがってディスパッチするセグメントは (2, 1) のセグメントとなる．

■2: reg1 がレディで reg2 がレディでない場合　r2 = r6 + r3 という命令について考える．r6 はレディで r3 はレディでないとする．第 1 ソース・レジスタはレディであるため，メイン・セグメントはどこでも良い．しかし第 2 ソース・レジスタはレディでないため． サブ・セグメントは 1(= 3 % 2) である必要がある．従って，(0, 1)(1, 1)(2, 1)(3,1) のいずれかのセグメントに割当を行う．

■3: reg1 がレディなく reg2 がレディである場合　r2 = r6 + r3 という命令について考える．r6 はレディでなく r3 はレディであるとする．第 1 ソース・レジスタはレディでないため，メイン・セグメントは 2(=6 % 4) となる．一方で第 2 ソース・レジスタはレディであるため，サブ・セグメントはどこでも良い．従って，(2, 0)(2, 1) のいずれかのセグメントに割当を行う．

■4: reg1，reg2 ともにレディである場合　どのセグメントにディスパッチしても良い (セグメント・フレキシブル)．

なお，今回の測定では，ディスパッチ可能なセグメントが複数ある場合 (2,3,4 の場合)，もっとも空き数の多いセグメントにディスパッチするようにしている．

4.1.2 サブ・セグメント方式におけるタグ比較

サブ・セグメント方式における第 2 ソース・タグの比較は，ディスティネーション・タグの下位ビットがサブ・セグメント番号と同じ場合のみ行われる．これによって，これまでの提案手法では削減ができなかった第 2 ソース・タグのタグ比較も削減することができる．

4.1.3 SWAP 方式へのサブ・セグメントの適応

サブセグメントは，SWAP 方式にも適応が可能である．基本的なディスパッチの動作は上述の説明と同じで，スワップを行う際には reg1 のタグを第 2 ソース・タグとして扱い，reg2 のタグを第 1 ソース・タグとして扱う．どのタイミングでスワップを行うかは，以前のレポートで説明したものと同様である．

4.2 評価

サブ・セグメント方式を適応した場合の提案手法による性能変化と比較回数の削減率を測定した．今回の測定では，提案手法のうちスワップを行わない通常の方式 (NORMAL) と，スワップを行い出来るだけ容量効率を低下させない方式 (SWAP_CONSERVATIVE : SWAP_CONS) について測定を行った．測定結果に関して評価を行う．評価環境は 1 と同様である．

4.2.1 NORMAL の評価

サブ・セグメント方式を実装した NORMLA における 性能変化，発行キューの占有率，比較回数の削減を図 4， 5， 6 に示す．各図におけるラベル Normal(a,b) は，メイン・セグメントの分割数が a，サブ・セグメントの分割数が b である NORMAL 方式であることを表す．

まず，図 4 の性能変化と図 5 の占有率について考える．図 4 の横軸はベンチマークとその平均を，縦軸は通常の発行キュー (age 論理) に対する相対性能を表す．図 5 の横軸はベンチマークとその平均を，縦軸は発行キューの占有率を表している．ラベルの Base は通常の発行キュー (age 論理) の際の占有率である．

図より，(32,1) 及び (16,2) ではいくつかのベンチマーク (cactuBSSN や lbm など) で性能が低下していることが分かる．これは図 5 の占有率を見て分かるとおり，分割するセグメント数が多すぎて大幅に占有率が低下していることが原因であると考えられる．

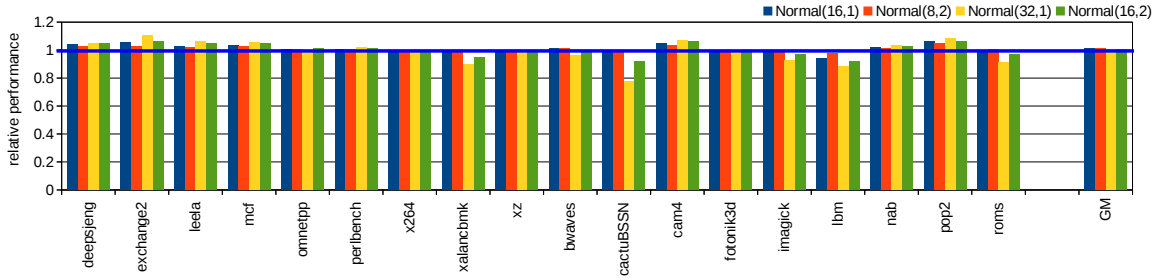

図 4　サブ・セグメントによる性能変化 (NORMAL)

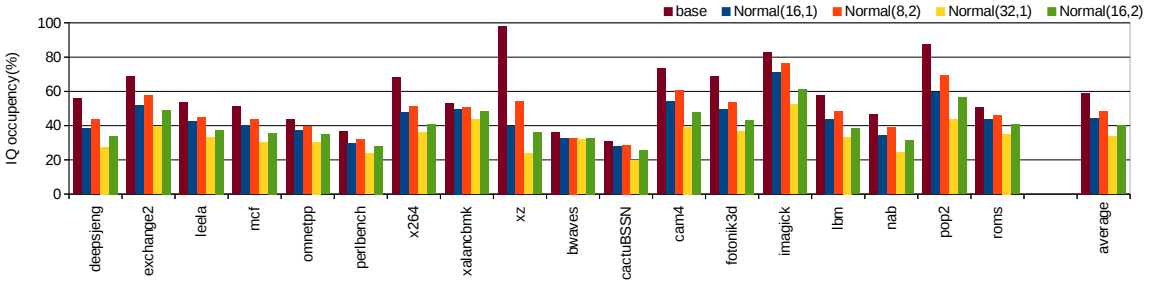

図 5　サブ・セグメントによる IQ 占有率の変化 (NORMAL)

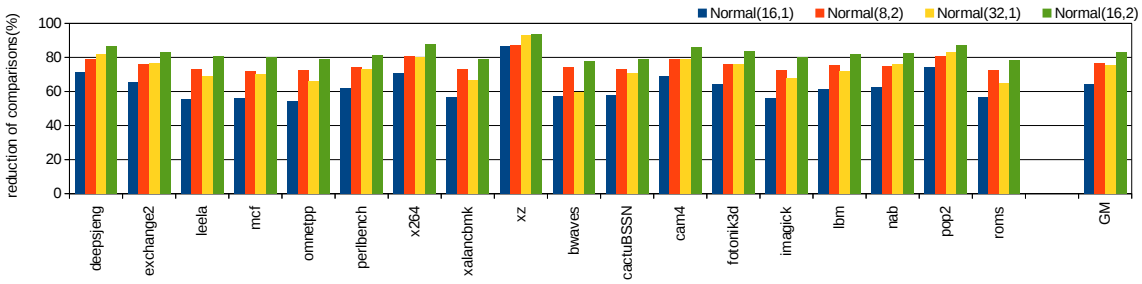

図 6　サブ・セグメントによる比較回数削減率 (NORMAL)

(32,1) と (16,2) に関して詳しく見てみると，性能が低下しているすべてのベンチマークで (32,1) よりも (16,2) の性能低下が小さくなっていることが分かる．これは，(32,1) よりも (16,2) のほうが占有率の低下が小さいためであると考えられる．従って，トータルのセグメント数が同じ場合 ((32,1) と (16,2) はどちらも 32 個のセグメントに分割されている) は，サブ・セグメントを用いたほうが占有率の低下が抑えられるため，結果として性能低下を小さくできるということがわかった．

次に，図 6 の比較回数削減に関して考える．図の横軸はベンチマーク及びその平均，縦軸は比較回数の削減率を表す．図より，サブ・セグメントを利用すると，利用しない場合に対して 10 % 程度削減率が増加することが分かる．従って，第 2 ソース・レジスタがレディでない場合も有効的にタグ比較の削減が行えていることが分かる．

4.2.2　SWAP_CONSERVATIVE の評価

サブ・セグメント方式を実装した SWAP_CONSERVATIVE における性能変化，発行キューの占有率，比較回数の削減を図 7, 8, 9 に示す．各図の表現は NORMAL の場合と

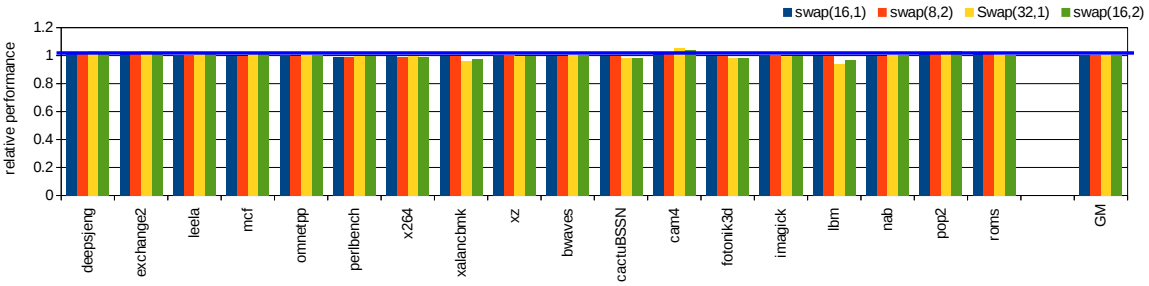

図 7　サブ・セグメントによる性能変化 (SWAP_CONSERVATIVE)

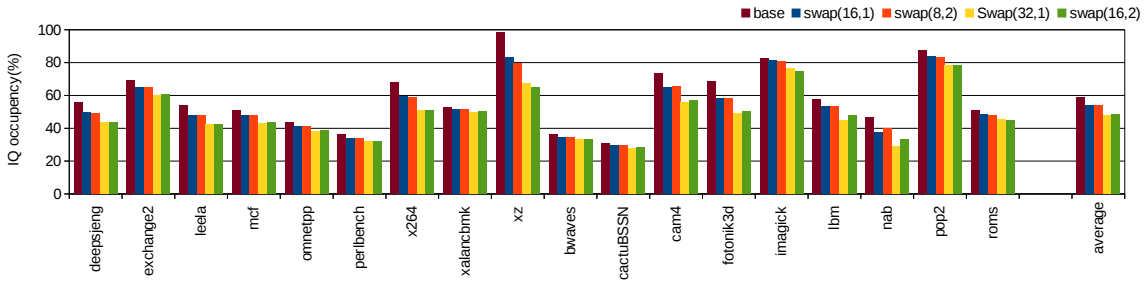

図 8　サブ・セグメントによる IQ 占有率の変化 (SWAP_CONSERVATIVE)

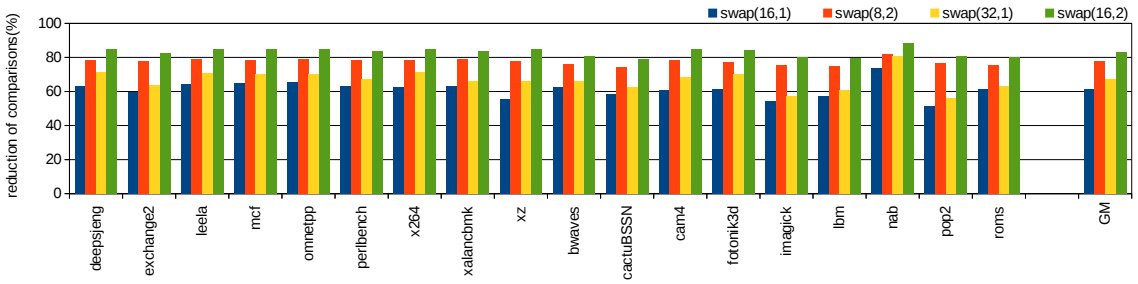

図 9　サブ・セグメントによる比較回数削減率 (SWAP_CONSERVATIVE)

同様である．

まず，図 7 の性能変化に関して考える．SWAP_CONSERVATIVE は容量効率を重視した方式であるため，基本的に大きな性能低下は見られない．

次に図 8 の占有率に関して見てみると，NORMAL の場合と比較して占有率の低下は小さいことが分かる．また，(32,1) と (16,2) を比較してみても，多くのベンチマークで大きな違いはみられない．従って SWAP_CONSERVATIVE の場合には，サブ・セグメントを利用することによる占有率の変化は小さいと言える．

次に，図 9 の比較回数の削減に関して考える．こちらも NORMAL の場合と同様に，サブ・セグメントを利用したほうがより多く削減できていることがわかり，サブ・セグメントが有効に働いていると言える．

4.2.3　サブ・セグメント方式のまとめ

以上の結果により，サブ・セグメントを利用すると，

- NORMAL の場合は占有率の低下を抑えることができ

- 比較回数を 10% 程度多く削減できる

ことがわかった．従って，サブ・セグメントは非常に有効な手段であると言える．

5 研究計画

- HSPICE を用いた電力測定を開始する
  - 現在は測定方法などを松田さんから教わり，各コードで何をやっているかの理解をしている段階
  - ウェイクアップ論理を修正して提案手法での消費電力及び遅延を測定できるようにする
  - フリー・リストの論理に関してはおそらく実装されていないため，実装して測定する
  - 提案手法におけるディスパッチ時の遅延の増加が懸念されるため実装し測定したい
- LTP の予測精度向上
- サブ・セグメント方式のチューニング

6 関連文献: MLP-Aware Dynamic Instruction Window Resizing for Adaptively Exploiting Both ILP and MLP [7]

本論文では，MLP を利用できる場合発行キューのサイズを大きくし，そうでない場合には発行キューのサイズを小さく制限する手法を提案している．

一般に，IQ のサイズが大きいほど利用できるメモリ・レベル並列性 (MLP) は大きくなり，キャッシュ・ミスによるメモリアクセスのレイテンシを隠蔽することができ，性能の向上につながる．しかし，すべてのプログラムにおいて MLP が利用できるわけでない． MLP が利用できない場合， IQ のサイズが大きいことが原因となり， ILP が損なわれたり，分岐命令の実行が遅れ分岐予測ミス・ペナルティが増加してしまい，性能の低下が発生する．

従って，MLP が利用できる場合には IQ のサイズを大きくし，そうでない場合にはサイズを小さく制限することが有効であると考えられる．本研究では，LLC キャッシュ・ミスが生じた際に IQ のサイズを大きくし，メモリ・アクセスに必要なサイクルが経過した後に元のサイズに戻すという方法を提案している．

通常 LLC キャッシュ・ミスは固まって発生するため，一度 LLC ミスを起こすと連続して メモリ・アクセスが発生するため，この間に発行キューのサイズを大きくしておけばより MLP を利用できる．

本論文によると，この提案手法により平均で 21% の性能向上を達成している．

6.1 自身の提案手法への応用

自身の提案手法における SWAP_AGGRESIVE と SWAP_CONSERVATIVE は，容量効率と比較回数の削減の間にトレードオフの関係がある．そこで，この関連論文のように容量効率が重要なときのみ SWAP_CONSERVATIVE を用いて，そうでない場合には SWAP_AGGRESIVE を用いれば，より積極的な削減を達成しつつ性能低下を抑制できるのではないかと考えられる．これは，提案手法の応用方法として検討すべき今後の課題と言える．

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*, pp. 20–25, October 2001.

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

[4] J.R. 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.

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

[7] Y. Kora, K. Yamaguchi, and H. Ando, Mlp-aware dynamic instruction window resizing for adaptively exploiting both ilp and mlp, In *In Proceedings of the 46th Annual International Symposium on Microarchitecture, pp.37-48*, December 2013.