Vol.2013-ARC-206 No.4
2013/7/31

マルチスレッド・プロセッサにおける レジスタ・キャッシュ・システムの評価

西川 卓[1]　倉田 成己[1]　塩谷 亮太[2]　五島 正裕[1]　坂井 修一[1]

概要：
プログラムの実行効率を高める方法に，マルチスレッド・プロセッサを用いたものがある．シングルプログラムの実行性能を上げるマルチスレッド実行によってプログラムの実行効率は上がるが，マルチスレッド・プロセッサであるため，そのコンテキストの数に応じた，多くのレジスタが必要となる．プロセッサ中のレジスタが増えることによる消費電力の増大，それに伴う熱量の増加は，深刻な問題である．レジスタ・ファイルの面積を削減する手法の１つに，**レジスタ・キャッシュ**がある．レジスタ・キャッシュはその名の通りレジスタのキャッシュで，レジスタ・ファイルの複雑さを軽減することで，レジスタの増加に伴う上記の問題点を解決する．そこで本稿ではシングル・プログラムの実行性能を向上するマルチスレッド・プロセッサ に，レジスタ・キャッシュを適用することを提案する．評価にあたっては，当研究室で研究している，**SoF-MT(Switch-on-Future Event Multithreading)** と **NORCS(Non-Latency-Oriented Register Cache System)** を用いた．その結果，あるベンチマークを除いては，マルチスレッド実行によりレジスタ・キャッシュ・ヒット率は低くなるが，それが性能低下に影響を与えないことが分かった．

1. はじめに

シングル・プログラムの実行性能向上を妨げる大きな要因として，**delinquent 命令**がある．delinquent 命令とは，キャッシュ・ミスや分岐予測ミスなどのペナルティを頻繁に起こす静的な命令のことである．ロード命令では，多くのキャッシュ・ミスがごく少数の静的なロード命令によって引き起こされていることが知られている [1]．また，分岐命令についても同様であり，大部分の分岐予測ミスがごく少数の静的な命令によって引き起こされている [2].

この delinquent 命令の重要な性質として, 大部分の delinquent 命令が比較的小さなループの中で繰り返し実行されていることがあげられる．後の 2 章で述べるが，SPEC CPU2006[3] における多くのプログラムでは，数十から 200 命令程度の比較的小さなループ内において delinquent 命令の大部分が実行されていることが我々の調査で分かった [4].

この delinquent 命令の遅延を隠蔽する手法として，マルチスレッド・プロセッサによるものがいくつかある．例えば，**Switch-on-Event** [5], [6] では，delinquent 命令 のキャッシュ・ミスをトリガとしてスレッドを切りかえることで，遅延の隠蔽を図っている．しかし，Switch-on-Event はミスが発生してからスレッドを切り替えるため，分岐予測ミスによる遅延は削減できない.

他にもシングル・プログラムの実行性能向上をはかる，**ヘルパースレッディング** [1], [7], [8], [9] がある．この手法では，シングル・プログラムから delinquent 命令にかかわる命令のみを抜き出した**ヘルパースレッド**と呼ばれるスレッドを生成する．ヘルパースレッドをメインのスレッドに対して先行実行することで, 分岐予測精度の向上やプリフェッチを行う．しかし delinquent 命令の大多数は小さなループ中に存在するため，ヘルパースレッディングでは性能向上を効果的に得ることができない [4].

対して我々の提案する **SoF-MT (Switch-on-Futrue Event Multithreading)** では，こうした小さなループ中に存在する delinquent 命令の遅延をも隠蔽することができる．これはループの各イタレーションをスレッドとし，マルチスレッド実行することで実現している．これにより SoF-MT はシングル・プログラムの実行性能を向上させた [4]

しかし上記で述べたマルチスレッド・プロセッサにおいては，プロセッサコンテキストが複数あるため，レジスタ・ファイルの面積も大きくなる．そのため十分な性能向上を得ようと，並列実行するスレッド数を増やせば，その分だけレジスタ・ファイルの面積が増える．レジスタ・ファイ

[1] 東京大学大学院情報理工学系研究科
[2] 名古屋大学大学院工学研究科

ルが大きくなることは，プロセッサ内に発生する熱量，消費電力の増加を伴うので，非常に深刻な問題である．このような面積の問題はレジスタ・キャッシュによって解決される．

そこで我々はマルチスレッド・プロセッサに レジスタ・キャッシュ・システムを適用することを提案する．レジスタ・キャッシュはレジスタ・ファイルの複雑さを軽減することで，その面積を削減するものである．4 章で述べるが，レジスタ・ファイルのポート数を削減することで，レジスタ・ファイルの面積を大きく削減する．

レジスタ・キャッシュの 1 つである **NORCS(Non-Latency-Oriented Register Cache System)**[10] はパイプラインが乱れにくい特徴がある．これはストールが発生するメイン・レジスタ・ファイルへのポート数より，多くのキャッシュ・ミスが発生した時にのみストールが発生するということだ．このことについて後の 4 章でくわしく述べるこのため NORCS はマルチスレッド・プロセッサのような，ワーキング・セット・サイズが多くレジスタ・キャッシュ・ヒット率を下げるようなアーキテクチャにおいても，性能を下げずにその面積効率を向上させる．

そこで本論文では，シングルプログラムの実行を性能を高めるマルチスレッド・プロセッサである SoF-MT に対して，NORCS を適用することを提案し，その評価を行う．それによって，実際にマルチスレッド・プロセッサのヒット率の低下が，SoF-MT による性能向上に影響を及ぼしているか確認する．

本稿の構成は以下の通りである．まず 2 章で，ループ中の delinquent 命令の性質について述べる．3 章では SoF-MT の遅延の隠蔽について述べ，レジスタ・ファイルの増加について言及する．4 章では，レジスタ・キャッシュの具体的な面積削減方法について述べ，SoF-MT への NORCS の適用を提案する．つづく 5 章では SoF-MT と NORCS を組み合わせたプロセッサ・モデルをシミュレータ上に実装した結果を述べる．最後に 6 章で本稿をまとめ，今後の方針を述べる．

2. delinquent 命令とループの関係

delinquent 命令とは，キャッシュミスや分岐予測ミスを頻繁に起こす静的な命令を指す．したがって，1 つの静的な命令に対して複数回の実行が行われていることになる．通常，プログラムカウンタは後方分岐を経なければ単調増加する．同じ命令アドレスの命令が複数回実行された場合，それは前回の実行からの間に後方分岐を 1 回以上実行したことを意味する．また，後方分岐を含む繰り返し構造はループであるといえる．これを考慮すると，delinquent 命令はループの中に存在すると考えられる．

我々はループのサイズと，ループ中で実行された delinquent 命令の割合の関係に着目して評価を行った [4]．

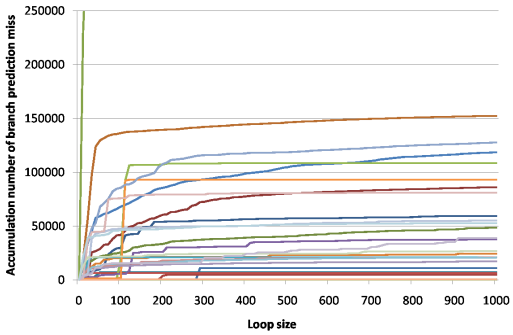

図 1 ループ・サイズと累積分岐予測ミス回数

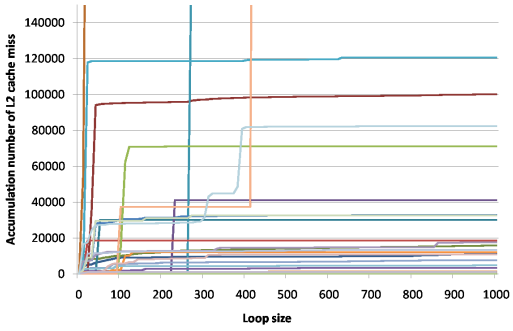

図 2 ループ・サイズと累積 L2 $ ミス回数

その評価結果が**図 1** および**図 2** である．図 1 は分岐予測ミスについて，図 2 は L2 キャッシュ・ミスについてのものである．評価には，SPEC CPU2006[3] に含まれる全アプリケーション（29 本）のシミュレーションを行っている．

グラフの横軸はミスを起こした命令が所属しているループのサイズを表す．また，縦軸はそのサイズ以下のループで発生してるミス数を表す．グラフの読み方の例としては，たとえばループ・サイズが 400 でミス数が 10000 の折れ線上の点は，ループ・サイズが 400 より小さいループ中で 10000 命令のミスが発生していることを意味する．

図 1 より，分岐予測ミス数の絶対値が大きなプログラムでは，およそ 50 命令から 200 命令程度以下のループ内でミス回数が飽和していることがわかる．図 2 についても同様であり，L2 キャッシュ・ミス数の絶対値が大きなプログラムでは，およそ 400 命令以内のループでミス回数が飽和していることがわかる．

上記の結果より，数十から 400 命令程度の比較的小さなループ内において delinquent 命令の大部分が実行されていることがわかる．

3. Switch-on-Future Event Multithreading

マルチスレッド・プロセッサでは，スレッドを切り替えることによって，キャッシュ・ミスや分岐予測ミスの遅延を隠蔽することが可能である．しかし 1 章で述べたように，delinquent 命令に関しては小さなループ中に存在する

ため，遅延の隠蔽を投機によって行うのは難しい．

我々は delinquent 命令が小さなループ中に大多数存在することに注目し，Switch-on-Future Event Multithreading(SoF-MT) を提案している．これは，単一プログラム内のループから生成されたスレッドの実行により delinquent 命令の遅延を隠蔽する手法である．

3.1 プロセッサのモデル

SoF-MT のプロセッサ・モデルは，基本的には SMT [*1] と同等の機能を持つものである．ループの各イタレーションはスレッドとして割り当てられて実行が行われる．また，各スレッドは隣接するスレッド同士でレジスタ間通信が可能になっている．

3.2 ループのマルチスレッド実行

スレッドは，ループにおける各イタレーション中の命令列によって構成される．新しいスレッドの生成は各スレッドの完了時に行われ，後続のイタレーションに相当するスレッドを 1 つ生成する．**図 3** はプログラム中の for 文における各イタレーションをスレッドに分ける様子を表している．

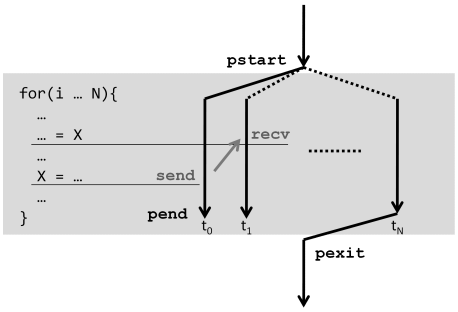

図 3 イタレーションのマルチスレッド実行

マルチスレッドの実行開始，スレッドの実行終了と新しいスレッドの生成，マルチスレッドの実行終了はそれぞれコンパイラによって挿入される専用の命令によって行われる．またスレッド間の依存，つまりループのイタレーション間の依存の解決についても専用の送受信命令にて行う．これらの命令の挿入を行うループは，プログラマがコード上から明示的に指定を行うか，実行時プロファイルの結果を元にコンパイラによって選択される．

3.3 delinquent 命令の検出と遅延の隠蔽

delinquent 命令の検出は，実行時にミスを頻繁に起こす命令を動的に特定し，次回の命令フェッチ時に行う．この手法では命令フェッチ時に，実行時に起きるであろう各種ミスを予測し，それらのイベントが起こるよりも前にスレッドの切り替えを行う．**図 4** は for ループ中に delinquent な分岐命令が存在した場合の遅延の隠蔽を示す．図中では $if(item[i])$ が delinquent 命令を含む．今，スレッド t_0 の分岐命令をフェッチし，delinquent 命令と判定されたら次のスレッド t_1 にフェッチ先を切り替えて実行を継続する．このスレッドでも，delinquent 命令をフェッチしたら次のスレッドにフェッチ先を切り替える…という動作を繰り返す．t_0 の分岐先が決定したところでフェッチ先のスレッドを t_0 に戻すと，分岐予測ミスによる遅延が隠蔽される．

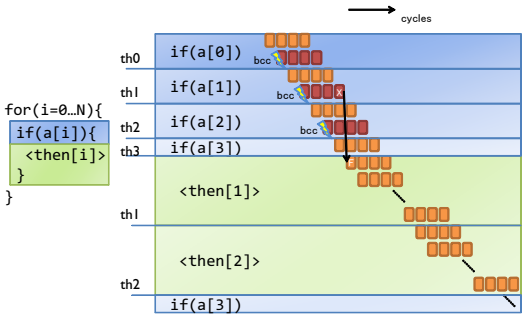

図 4 分岐予測ミスの隠蔽

レジスタ・ファイルは，近年の OoO スーパースカラ・プロセッサの構成要素の中で，特に面積の大きいものの一つとなっている．その理由の一つとして，ポート数の増加が挙げられる．例えば 4 命令同時発行のスーパースカラ・プロセッサでは，リード・ポートが 8 つ，ライト・ポートが 4 つ必要となる．レジスタ・ファイルを構成する RAM の回路面積はポート数の 2 乗に比例するため，その面積は容量の割に非常に大きなものとなり，それに伴う消費電力や発熱が大きな問題となる．

マルチスレッド・プロセッサにおいては，上記に加えてさらにエントリ数の増加が問題となる．通常，レジスタ・ファイルは in-flight な命令数に応じた容量が必要となるが，マルチスレッド・プロセッサにおいてはさらに同時実行するスレッド数に比例した容量が必要となるためである．delinquent 命令の遅延を隠蔽するために同時実行可能スレッド数を確保しようとすると，レジスタ・ファイルのエントリ数が増加し，回路面積の増大がより深刻なものとなる．

4. レジスタ・キャッシュ・システムの適用

マルチスレッド・プロセッサにおけるレジスタ・ファイルの回路面積増加に対して，我々はレジスタ・キャッシュ・システムの適用を提案する．以下ではまず，レジスタ・キャッシュ・システムによる回路面積削減の仕組みについて説明する．その後，レジスタ・キャッシュ・システムの 1 つである NORCS について説明し，これをマルチス

[*1] Simultaneous Multithreading(SMT) プロセッサ:単一のコア内に複数のプログラムカウンタなどのコンテキストを持ち，複数のスレッドを逐次切り替えることによりそれらを同時に実行できるプロセッサ

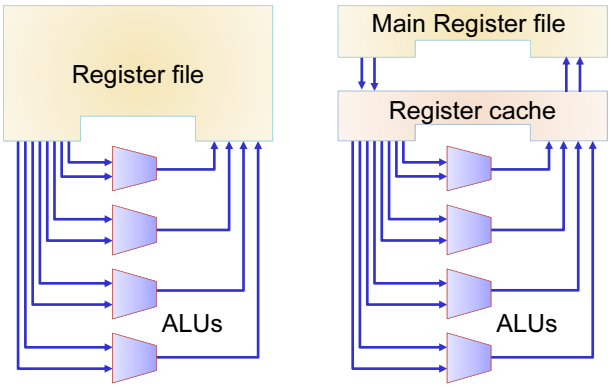

図 5 レジスタ・キャッシュ・システムによるポート数削減

レッド・プロセッサへ適用することを提案する．

4.1 レジスタ・キャッシュ・システムによるレジスタ・ファイルの面積削減

レジスタ・キャッシュ・システムは，多ポート少エントリのレジスタ・キャッシュと少ポート多エントリのレジスタ・ファイルを組み合わせにより構成される．ここで，レジスタ・ファイルの面積はポート数の 2 乗とエントリ数に比例して大きくなる．レジスタ・キャッシュは少エントリであり，レジスタ・ファイルは少ポートであるため，全体として回路面積を大きく削減することができる．

図 5 はレジスタ・ファイル（左）と，レジスタ・キャッシュ・システム（右）の模式図である．ここで，双方のレジスタ・ファイル エントリ数は同一とする．左図のレジスタ・ファイル は 8-read /4-write の合計 12 ポートであるが，右図では 2-read /2-write の 4 ポートとなっており，面積が大きく削減される．また，レジスタ・キャッシュ はエントリ数がレジスタ・ファイル の 1/10 程度であるため，レジスタ・ファイル とレジスタ・キャッシュ を合わせた全体としての回路面積も大幅に削減されていることがわかる．

以下ではポートを削減する手法を，読み出しと書き込みのそれぞれについて説明する．

- 読み出しポートの削減
  レジスタ・キャッシュを搭載しているプロセッサにおいて，レジスタ・キャッシュにヒットし続ける限り，メイン・レジスタ・ファイルへのアクセスは省略できる．これにより大幅な性能低下なく，メイン・レジスタ・ファイルのポート数を削減することができる．
- 書き込みポートの削減
  レジスタ・ファイルへの書き込みは，一時的にライト・バッファに保持される．ライト・バッファを介したレジスタ・ファイルへの書き込みによって，メインレジスタの書き込みポートを命令の平均実行スループットにまで下げることが可能である．

レジスタ・ファイルの面積はポート数の 2 乗に比例して大きくなる．そのためポート数の削減は，レジスタ・ファイルの面積を大きく減らす．

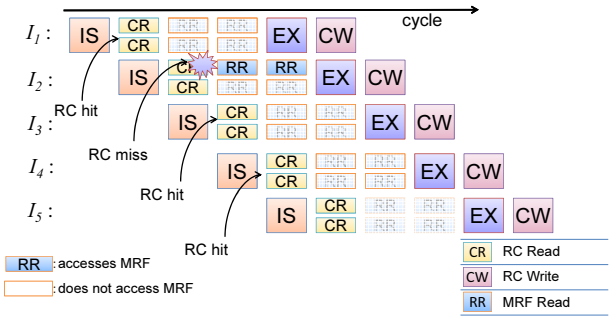

図 6 NORCS のパイプライン

4.2 NORCS

NORCS (Non-latency-Oriented Register Cache System) は，**レジスタ・キャッシュ・ミスを仮定した**パイプラインを持つ．一般的なレジスタ・キャッシュ・システムでは，レジスタ・キャッシュ・ミスの際にバックエンドをストールし，レジスタ・ファイルへのアクセスを行う．一方 NORCS は，レジスタ・キャッシュへのアクセスと同時に，メイン・レジスタ・ファイルにアクセスするステージを持つ．**図 6** は NORCS のパイプラインを表した図である．この図において命令 I_2 が，レジスタ・キャッシュにアクセスする CR ステージにおいて，レジスタ・キャッシュ・ミスを引き起こしている．だがレジスタ・キャッシュ・ミスが発生しても，メイン・レジスタ・ファイルにアクセスする RR ステージが確保されているため，ストールは発生しない．

NORCS では，メイン・レジスタ・ファイルへのポート数より多くのレジスタ・キャッシュ・ミスが 1 サイクルあたりに生じた時にのみ，ストールが発生する．このように NORCS では，ある程度のレジスタ・キャッシュ・ミスを起こしても，パイプラインが乱れず，性能低下を 2%程度に抑えることに成功している．

4.3 マルチスレッド・プロセッサへの適用にあたって

我々は NORCS を SoF-MT に適用し，レジスタ・ファイルの回路面積を削減する手法を提案する．レジスタ・ファイルのエントリ数はスレッド数に比例するため，レジスタ・ファイルの面積には (レジスタ・ファイルの面積) $\propto$ (ポート数)^2 $\times$ (スレッド数) という関係が成り立っている．これによるとポート数が減ることで，スレッド数の増加によるレジスタ・ファイルの面積の増加を抑えることができる．これによりマルチスレッド・プロセッサにレジスタ・キャッシュを適用することで，レジスタ・ファイルの面積の問題を大幅に緩和できる．

ここで，マルチスレッド・プロセッサはスレッド数に応じてコンテキストが増大するため，ワーキング・セット・サイズが増加し，レジスタ・キャッシュ・ミスも増加する可

能性がある．前述のように，NORCS ではレジスタ・ファイルのポート数より多くのレジスタ・キャッシュ・ミスが発生したときに初めてストールが発生する．そのため，レジスタ・キャッシュ・ミスの増加が即，性能低下に繋がるわけではない．レジスタ・キャッシュ・ミスが頻繁に発生するようであれば性能低下を避けられないため，何らかの対策が必要となる．

しかし，SoF-MT はスレッドの切り替えタイミングが delinquent 命令のフェッチ時であり，通常の SMT プロセッサなどと比べて比較的粒度が粗い．そのため，レジスタ・キャッシュの局所性が保たれ，レジスタ・キャッシュ・ミスはほとんど増加しないのではないかと予想される．

つまり我々の提案する SoF-MT に NORCS を適用することで，大幅な性能低下なく，レジスタ・ファイルの大幅な面積削減が可能である．次章では SoF-MT に NORCS を適用し評価を行った．

5. 評価

表 1 プロセッサの構成

パラメータ	値
ISA	Alpha 21164A
logical thread	4 way
fetch width	4 inst.
execution unit	int : 2, fp : 2, mem : 2.
instruction window	int : 32, fp : 16, mem : 16
register file	int : 512, fp : 512
register file ports	read: 2, write: 2
branch prediction	8KB g-share
miss penalty	10 cycle
BTB	2K entry, 4-way
L1C	32KB, 4-way, 64B/line, 2 cycle
L2C	4MB, 8-way, 64B/line, 10 cycle
main memory	400 cycle

表 2 評価に用いたベンチマーク

ベンチマークセット	アプリケーション
SPECCPU 2006[3]	bzip2, mcf, milc, hmmer, lbm

SoF-MT に NORCS を組み合わせたマルチスレッド・プロセッサについての性能評価を行った．

5.1 評価環境

サイクル・アキュレートなプロセッサ・シミュレータ鬼切弐 [11] に対して，SoF-MT と NORCS を組み合わせたプロセッサを実装し，評価を行った．評価したプロセッサのパラメータは**表 1** の通りである．評価に用いたベンチマークを**表 2** に示す．プログラムのコンパイルには gcc4.3.3 をベースとし，SoF-MT に必要な専用命令を挿入することができるコンパイラを作成し用いた．コンパイラの最適化オプションは-O3 を使用する．各ベンチマークでは最初の 1G 命令をスキップし，続く 100M 命令を実行した．なお我々が作成したコンパイラは現状プログラマの手でソースコードのループを指定することにより専用命令を挿入する仕様である．そのため，ベンチマークは delinquent 命令の特定が出来ており，かつループの指定が可能なものを選んでいる．

5.2 評価結果

ここでは以下の 2 つのパラメータを変化させて IPC，レジスタ・キャッシュ・ヒット率を評価を行った．

- SoF 実行されるスレッド数
- レジスタ・キャッシュのエントリ数

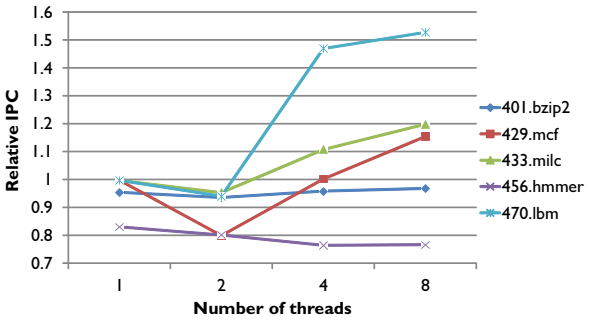

図 7 スレッド数による相対 IPC 変化

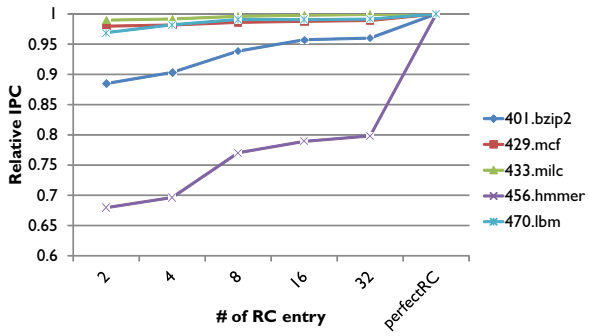

図 8 レジスタ・キャッシュのエントリ数による相対 IPC の変化

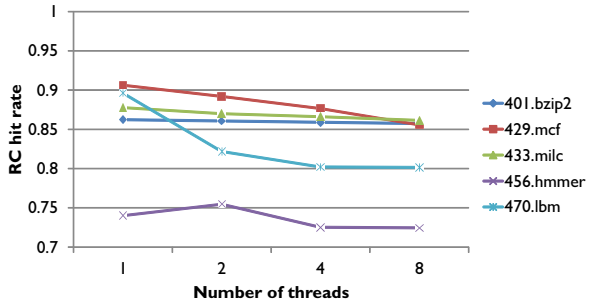

図 9 スレッド数によるレジスタ・キャッシュ・ヒット率の変化

図 7 はレジスタ・キャッシュのエントリ数を 16 に固定し，SoF-MT を行うスレッド数を変化させたときの，シング

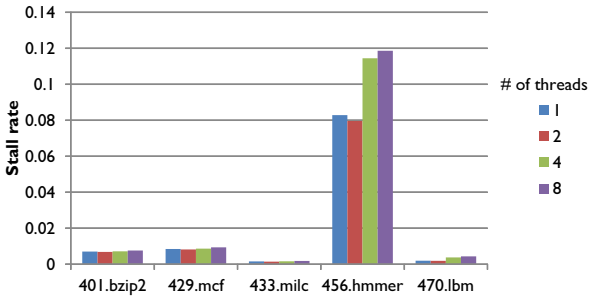

図 10 NORCS におけるストール発生率

ルスレッド・プロセッサに対する相対 IPC である．スレッド数が 2 のときはいずれのベンチマークについても性能が低下している．これは，切り替えるスレッドが少ないため遅延の隠蔽が十分に行われず，SoF-MT のオーバーヘッドの影響が大きいためである．429.mcf，433.milc，470.lbm に関しては，スレッドが 4，8 と増加するにつれて性能が向上している．これはスレッド数の増加に伴い，分岐予測ミスやキャッシュ・ミスの遅延が隠蔽されていることによるものである．一方 401.bzip2 と 456.hmmer に関しては，スレッド数が増加しても性能が向上していない．これらについてより詳細に調査するため，レジスタ・キャッシュのエントリ数を変化させたときの性能について評価を行った．

スレッド数を 8 と固定し，レジスタ・キャッシュのエントリ数を変化させたときの相対 IPC を表した図が**図 8** である．ここで，ベースラインはレジスタ・キャッシュが perfect にヒットするモデルである．429.mcf，433.milc，470.lbm については，2 エントリ程度でも相対 IPC がほとんど下がらない．このことからスレッド数を増加させても，少量のエントリ数で性能が保たれることが分かる．また 401.bzip2 についても，エントリ数が 16 程度であれば性能低下は 5%未満に抑えられる．一方，456.hmmer に関してはレジスタ・キャッシュが 32 エントリでも性能が約 20%低下している．

図 9 はレジスタ・キャッシュのエントリ数を 16 に固定し，SoF-MT を行うスレッド数を変化させたときの，レジスタ・キャッシュ・ヒット率の変化である．456.hmmer を除く全てのベンチマークについて，スレッド数が増加するにしたがってキャッシュ・ヒット率は低下している．一方，456.hmmer は特にヒット率が低く，またスレッド数に依存していない．

レジスタ・ファイルのポート数よりも多くのレジスタ・キャッシュ・ミスが同時に発生すると，ストールが起こり，性能低下の要因となる．**図 11** は，レジスタ・キャッシュのエントリ数を 16 に固定し，SoF-MT を行うスレッド数を変化させたときの，レジスタ・キャッシュによって生じるストール率の変化を表したグラフである．456.hmmer は他のベンチマークに比べてストール率が高く，特に 8 スレッドの場合には約 11.9%ものストールがレジスタ・キャッシュによって起こっていることが分かる．456.hmmer を除いたベンチマークにおいては，ストールの発生がスレッド数によらず，低い水準で推移していることが分かる．これはレジスタ・キャッシュ・ヒット率がスレッドの増加に応じて低下しようが，IPC 低下に大きな影響を及ぼさないことを示している．

結論としては，スレッド数の増加によりたしかにレジスタ・キャッシュ・ヒット率は低下するが，性能に影響が出るレベルではないことがわかった．性能が低下しているベンチマークについてはスレッド数に関わらず（スレッド数が 1 の場合でも）キャッシュ・ヒット率が低く，ベンチマーク特有の問題と考えられる．以下ではこのベンチマーク 456.hmmer について詳細に調査した結果を示す．

5.3 NORCS により性能低下するベンチマーク

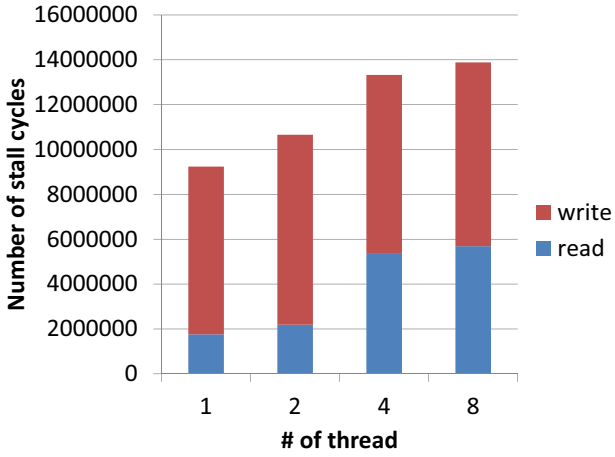

図 11 リードによるストールとライトによるストール

456.hmmer は隠れマルコフアルゴリズムを行うベンチマークで，実行時のスループットが大きく，シングル・スレッド実行においてもワーキング・セット・サイズの大きいベンチマークであった．そのためライト・ポートの不足による性能低下の影響が大きいと思われる．そこで，レジスタ・キャッシュ・ミスが原因のストールのうち，リード・ポート由来のものとライト・ポート由来のものの割合を調査した．図 11 は，エントリ数を 16 に固定し，スレッド数を変化させたときのレジスタ・キャッシュ・ミスが原因で発生したストールのサイクル数である．

これを見ると，シングル・スレッドのときはライト・ポートの不足が主な原因でストールが発生していることがわかる．これはベンチマークのスループットが高いため，ライト・バッファへの平均書き込み速度がライト・ポート数よりも高く，書き込みが追いついていないためである．この問題については Write Squash Buffer によって解決可能である [12]．

一方，スレッド数を増加させるとストール発生率も増加

するが，その原因はリード・ポートの不足によるものである．この問題に対してはベンチマークのリードアクセスの特性を調べて，その対応策を考える．

6. おわりに

本論文では，マルチスレッド・プロセッサにおける，レジスタ・ファイルの面積の問題を解決する手法としてレジスタ・キャッシュに注目し，レジスタ・キャッシュにマルチスレッド・プロセッサを適用することを提案した．その適用にあたって delinquent 命令の性質を注目したマルチスレッディング・プロセッサである SoF-MT と，パイプラインの発生しにくいレジスタ・キャッシュの NORCS を適用し評価した．

評価では，8 スレッド程度であれば delinquent 命令による遅延を隠蔽し，性能が向上するプログラムがある一方，レジスタ・キャッシュ・ミスによりストールが発生し，性能が低下してしまっているプログラムもいくつか存在した．

だが 1 つのベンチマークを除いて，レジスタ・キャッシュ・ヒット率の低下が，必ずしも性能低下につながるわけではないことが分かった．加えてスレッド数を増加させても，少量のエントリ数で性能が保たれることが分かる．

その一方で 456.hmmer はスレッド数の増加に従い，リード・ミスの増加によりストール発生率が高くなる，他のベンチマークにみられない傾向を示した．

今後の予定としては，456.hmmer のリードによるストール発生の原因をレジスタへのアクセス特性を調べ，その対応策を考える．そして 456.hmmer のようなワーキング・セット・サイズの大きなプログラムにおいても，性能を落とさず面積を削減する手法を考える．

7. 謝辞

本論文の研究は一部，文部科学省科学研究費補助金 No. 23300013 による．

参考文献

[1] Collins, J., Wang, H., Tullsen, D., Hughes, C., Lee, Y.-F., Lavery, D. and Shen, J.: Speculative precomputation: long-range prefetching of delinquent loads, *ISCA*, pp. 14–25 (online), DOI: 10.1109/ISCA.2001.937427 (2001).

[2] 塩谷亮太，五島正裕，坂井修一：分岐プレディシジョン，情報処理学会研究報告 2008-ARC-179，pp. 67–72 (2008).

[3] The Standard Performance Evaluation Corporation: *SPEC CPU2006 suite http://www.spec.org/cpu2006/*.

[4] 塩谷亮太，倉田成己，中島　潤，五島正裕，坂井修一：Switch-On-Future-Event マルチスレッディング，先進的計算基盤システムシンポジウム SACSIS2010，pp. 157–165（オンライン），入手先 ⟨http://www.mtl.t.u-tokyo.ac.jp/publications/paper/2010/J10-kenkyukai-shioya.pdf⟩ (2010).

[5] Farrens, M. and Pleszkun, A.: Strategies for achieving improved processor throughput, *ISCA*, pp. 362–369 (1991).

[6] McNairy, C. and Bhatia, R.: Montecito: a dual-core, dual-thread Itanium processor, *Micro, IEEE*, Vol. 25, No. 2, pp. 10–20 (online), DOI: 10.1109/MM.2005.34 (2005).

[7] Collins, J., Tullsen, D., Wang, H. and Shen, J.: Dynamic speculative precomputation, *MICRO*, pp. 306–317 (online), DOI: 10.1109/MICRO.2001.991128 (2001).

[8] Roth, A. and Sohi, G.: Speculative data-driven multithreading, *HPCA*, pp. 37–48 (online), DOI: 10.1109/HPCA.2001.903250 (2001).

[9] Chappell, R., Stark, J., S. Kim, S. R. and Patt, Y.: Simultaneous Subordinate Microthreading (SSMT), *ISCA*, pp. 186–195 (1999).

[10] Shioya, R., Horio, K., Goshima, M. and Sakai, S.: Register Cache System not for Latency Reduction Purpose, *IEEE Int'l Symp. on Microarchitecture (MICRO-43)*, pp. 301–312 (online), available from ⟨http://www.mtl.t.u-tokyo.ac.jp/publications/paper/2010/E10-conference-shioya.pdf⟩ (2010).

[11] 塩谷亮太，五島正裕，坂井修一：プロセッサ・シミュレータ「鬼斬弐」の設計と実装，先進的計算基盤システムシンポジウム SACSIS，pp. 120–121 (2009).

[12] 山田淳二，倉田成己，塩谷亮太，五島正裕，坂井修一：レジスタ・キャッシュ・システムの省電力化手法，情報処理学会　SWoPP(発表予定) (2013).