2003－ARC－154　(2)
2003／8／5

SMT プロセッサにおけるアウトオブオーダ実行に必要なリソースの共有方式の評価

加藤義人[†]　大和仁典[†]　笹田耕一[†]
佐藤未来子[†]　並木美太郎[††]　中條拓伯[††]

SMT プロセッサでは ILP（ILP）だけでなく，TLP（Thread Level Parallelism）を利用することにより高い並列度が実現できる．しかし，物理レジスタファイルに保持すべきコンテキストの増加により，物理レジスタファイルに必要なエントリ数が増加する．また，実行ユニットの数が増えると，それにともない物理レジスタファイルに必要な読み出しポート数が増加する．レジスタファイルのポート数の増加はレジスタファイルの面積の増加，レジスタアクセス時間の増加などを引き起こす．これらの影響は，多くの実行ユニットを持ち，多数のスレッドを実行可能な SMT プロセッサの実装の障害になる．本研究では，これらの問題を緩和する手法として，物理レジスタファイルの分割と，その動的な割り当てによるポート数の削減方式を提案する．

本方式によって最大で，レジスタファイルの総面積を約 73%，アクセス時間を約 70% 減少させることができた．また，提案方式導入による IPC の低下は最大でも 20%程度に抑えることができた．

An Evaluation of Method for Sharing Resources Needed to Perform Out-Of-Order Execution in an SMT Processor

Norito Kato,[†] Hironori Yamato,[†] Kouiti Sasada,[†]
Mikiko Sato,[†] Mitaro Namiki[††] and Hironori Nakajo[††]

SMT processors can perform high parallelism by exploiting TLP (Thread Level Parallelism) as well as ILP (Instruction Level Parallelism). However, additional threads increase number of physical register file entries for their context to be kept in the register file. Moreover, a physical register file in an SMT processor requires more ports for the increasing number of execution units. Increasing number of ports of a register file causes significant growth of area, access time and power consumption of a register file. These problems are significant hurdles to implement an SMT processor which executes the more number of threads with the more execution units. In this paper, we propose a strategy of dividing a physical register file into some sets of registers and dynamic allocation of divided register sets in order to reduce the number of port of a register file.

We have accomplished reduction in area of a register file up to about 73% and in access time of a register file up to 70% by using a proposed strategy. On the contrary, IPC degradation can be limited up to 20% by this strategy.

1. はじめに

近年，プログラム中の ILP(Instruction Level Parallelism) を見出すことへの限界が指摘され，TLP(Thread Level Parallelism) への注目が増してきている．TLP を利用するアーキテクチャの一つに SMT[9](Simultaneous Multithreading) がある．SMT とは，複数のスレッドが実行ユニットなどの命令実行に必要なリソースを共有し，効率よく並列度を上げることのできるアーキテクチャである．Intel 社の Hyper-Threading[4] はこの技術の実装例の一つである．

SMT プロセッサでは，TLP により並列度があがることが期待されるが，各スレッドが別のコンテキストを保持する必要があるので，必要なレジスタファイルのエントリ数は増加する．また，高い並列度を出すためには多くの命令を同時にフェッチし，かつ多くの実行ユニットを同時に使用できる必要があるため，必要になるレジスタファイルのポートの数は増加する．これらのレジスタファイルのエントリ数，ポート数の増加はレジスタファイルの面積，アクセス時間，消費電力を増加させる．これらの影響は，大規模な SMT プロセッサを実装する際の障害となることが予想される．本稿では，特にレジスタファイルのポート数に注目し，これらの問題を緩和する方法として，SMT プロセッサにおけるレジスタファイルの複製の作成によるポー

† 東京農工大学大学院工学研究科
Graduate School Technology, Tokyo University of Agriculture and Technology
†† 東京農工大学
Tokyo University of Agriculture and Technology

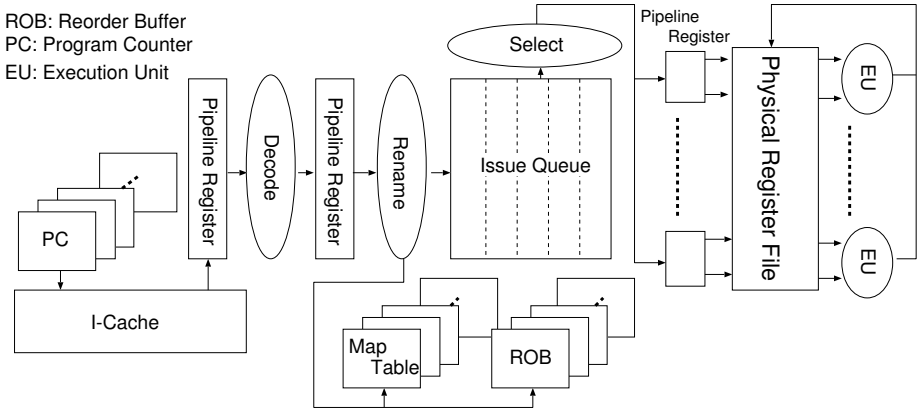


図 1 統合物理レジスタファイル方式

ト数の削減手法を提案する．

第 2 節で本方式が対象とする SMT プロセッサの構成について説明する．第 3 節では従来型の SMT プロセッサでの問題点を説明する．第 4 節では本稿で提案する分割物理レジスタファイルの動的割り当てについて説明する．第 5 節では本方式のシミュレータを使用した予備評価について述べる．第 6 節で関連研究を挙げ，第 7 節でまとめる．

2. 対象とする SMT プロセッサ

2.1 リネーミング実現方式

本方式が対象とするリネーミング実現方式は物理レジスタファイル中にリネーミングレジスタと論理レジスタを混在させ，その対応付けをマップテーブルによって管理するアーキテクチャとする．この方式は MIPS[12] や Alpha[2]，Pentium4[1] などで採用されている方式であり，ROB 中に投機実行された命令の結果を格納する方法に比べ，最新のレジスタの値を得るための連想検索が必要ないという利点がある．また，リネーミングレジスタとアーキテクチャレジスタを別のレジスタとして持つものに比べ，リネーミングレジスタからアーキテクチャレジスタへの書き戻し処理が必要ないため，ポート数が少なくてすむという利点がある．

本稿が前提とする SMT プロセッサは，このアーキテクチャを単純に拡張し，全てのスレッドが一つの物理レジスタファイルを共有するものとする．また，マップテーブルは各スレッド別のものを持つとする．本稿ではこの方式を**統合物理レジスタファイル方式**と呼ぶ．この方式の概念図を**図 1** に示す．

2.2 オペランドフェッチのタイミング

レジスタファイルからのオペランドのフェッチは演算を行う直前とする．この方式では，フロントエンドへの発行時にオペランドのフェッチを行う方法に比べ，発行待ち領域への演算結果のブロードキャスト，最新の値を保持しておくための記憶領域が必要ないという利点がある．また，この方式ではレジスタファイルのポート数は実行ユニットの数に依存する．

3. 統合物理レジスタファイル方式の問題点

3.1 レジスタファイルのエントリ数の増加

統合物理レジスタファイル方式の SMT プロセッサでは，同時に複数のスレッドのコンテキストを保持する必要があるため，レジスタファイルの総エントリ数が増加する．この方式で必要とされるレジスタファイルのエントリ数 E_p は 1 スレッドの論理レジスタの個数を E_l，最大動作スレッド数を T，リネーミングレジスタの個数を E_r とした場合

$$E_p = E_l T + E_r$$

となる．この方式は，物理レジスタファイルのエントリの割り当てを柔軟に行えるという特徴がある．例えば，プロセッサ上で 1 スレッドのみが走行中のとき，そのスレッドは全ての物理レジスタファイルのエントリを使用することができる．しかし，使用可能な物理レジスタファイルのエントリの増加は，ある程度の数までは利点となるが，プロセッサ上に存在できる命令数やフェッチ幅などが変わらなければ性能はさほど向上しない．

そのため，走行スレッド数が少ない場合非常に無駄が多い．この傾向はアーキテクチャレジスタのエントリ数が大きいアーキテクチャほど顕著である．例えば $T = 8$，$E_l = 32$，$E_r = 32$ のプロセッサの場合，物理レジスタファイルのエントリ数は 288 になるが，1 スレッドのみが走行中の場合，実際に使用されるエントリ数は，同時に必要とされるリネーミングレジスタ数の最大値が 32 であるとすると，64 のみである．

3.2 レジスタファイルのポート数の増加

TLP を生かすには，十分な実行ユニットとオペランド供給が必要になる．これらを実現するためには，レジスタファイルに多くのポートが必要になる．オペランドを二つ必要とし，出力が一つの演算パイプラインが n あった場合，調停を行わなければ，リードポートは $2n$，ライトポートは n 必要になる．レジスタファイルのポート数の増加はレジスタファイルの，面積，アクセス時間，消費電力の全てに悪い影響を与える．特に，レジスタファイルの面積はポート数の 2 乗に比例して増加し[7]，ポート数が与える影響が非常に強い．

このように，大規模な SMT プロセッサを統合物理

レジスタファイル方式で実現する場合，非常に大規模なレジスタファイルが必要になり，その面積，遅延，消費電力が問題になる．

4. 分割物理レジスタファイルの動的割り当て方式

本論文では，物理レジスタファイルを，リードポートの少ない複数の物理レジスタファイルに分割し，それらを動的にスレッドに割り当て，レジスタファイルの複製として使用することでリードポートの削減を実現する．この方式により，一つの物理レジスタファイルを全スレッドで共有する場合と比較し，物理レジスタファイルの総エントリ数は増加するものの，物理レジスタファイルの総面積，アクセス遅延が改善される．本稿ではこの方式を**分割物理レジスタファイルの動的割り当て方式**と呼ぶ．この方式の概念図を**図 2** に示す．なお，この図はリネームステージ以前などの従来の方式と変更のない部分は省略している．

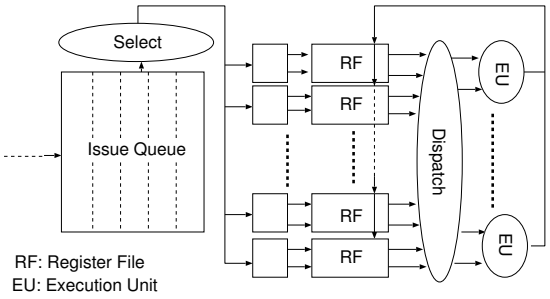


図 2 分割物理レジスタファイル方式

動的に割り当てを行う利点は，物理レジスタファイルのリードポートを削減しても性能の低下が起こりにくいという点である．例えば，各物理レジスタファイルがリードポートを 2 ずつ持っていたとする．動的に割り当てを行わず，スレッドが物理レジスタファイルを固定で所有していた場合，各スレッドはレジスタの値を必要とする命令を同時に 1 命令しか発行することができない．一方，動的に割り当てを行う場合，スレッドが所有している物理レジスタファイルの個数分命令を同時に発行することができる．この差が性能に与える影響は，走行スレッド数が少ない場合に大きくなる．

このように，分割物理レジスタファイルの動的割り当て方式は各スレッドに物理レジスタファイルを持たせる方式に比較し，走行スレッド数が少ない場合の性能低下が少ない．

4.1 物理レジスタファイルの分割

本方式では，物理レジスタファイルをいくつかに分割し，その分割された物理レジスタファイルをスレッドに割り当てる．以降，この物理レジスタファイルの使用権を持つスレッドを**オーナー**と呼ぶ．各物理レジスタファイルのオーナーは 1 スレッドに限るため，最低でも，最大同時走行スレッドと同数に分割する必要がある．

各物理レジスタファイルが持つリードポート数の決定は任意であるが，多すぎると本方式のメリットがほとんどなく，少なすぎると，各スレッドが一サイクルに読み出せるオペランドの数が少なくなるため，競合により性能が低下することが予想される．

実行ユニットからの結果は各物理レジスタファイルへブロードキャストされ，レジスタファイルのオーナースレッド番号と演算結果を生成した命令のスレッドが等しい場合書き込まれる．

4.2 Wakeup Logic

本方式では，Wakeup Logic の変更が必要になる．Wakeup Logic とは，ある命令が発行可能かどうかを決定する Logic である．例えば MIPS R10000 ではリザーベーションステーションの各エントリが命令が発行可能かどうかを表すビット (Rdy bit) を持っているが，これを各物理レジスタファイル用に複製する必要がある．**図 4** に統合物理レジスタファイル方式，**図 5** に本方式でのオペランド一つあたりの Rdy bit 更新ロジックの概略を示す．

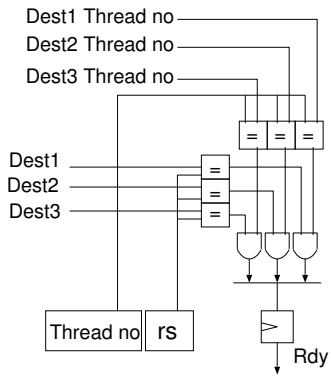


図 4 統合物理レジスタファイル方式用 Wakeup Logic

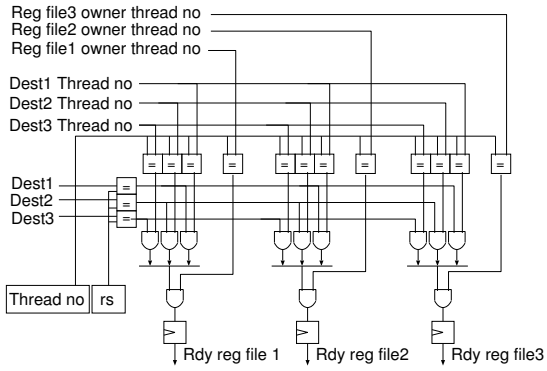
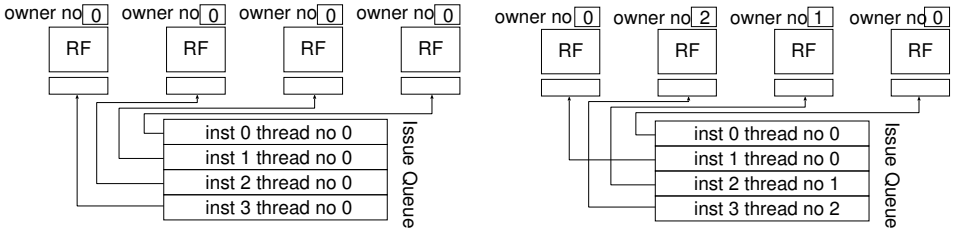


図 5 分割物理レジスタファイル方式用 Wakeup Logic

本方式では，各物理レジスタファイルごとに Rdy 信号が必要になる．ある命令のオペランドのレジスタ番号が結果のディスティネーションレジスタ番号と等しく，その命令が所属しているスレッドが所有しているレジスタファイルへ書き込みが行われた場合，対応する Rdy 信号が有効になる．また，MIPS R10000 以外



(a) 走行スレッド数 1　　(b) 走行スレッド数 3

図 3　動的物理レジスタファイル割り当て

の Wakeup Logic でも同様の複製が必要である．

本方式では，統合物理レジスタファイル方式に比べ，Wakeup Logic のハードウェア量が増加する．しかし，Issue Queue のエントリ数がさほど大きくない場合，本方式で削減できるハードウェア量に比べ，十分小さいと予想される．

4.3 Selection Logic

Selection Logic とは，発行する命令を決定する Logic であるが，本方式では Selection Logic も変更する必要がある．統合物理レジスタファイル方式では，発行可能な命令は実行ユニットに対して発行されたが，本方式では物理レジスタファイルに発行される．つまり，最も高い優先度をもつ数命令がレジスタファイルのリードポートの使用権を得る．実行ユニットへの発行と，物理レジスタファイルへの発行はさほど違いはないため，動作周波数の低下は無いと予想される．

4.4 実行ユニットへの Dispatch Logic

本方式では，物理レジスタファイルから命令をフェッチするのと同時に実行ユニットへの振り分けが行われる．実行ユニットの振り分けに必要なロジックは統合物理レジスタファイル方式とほぼ変わらないが，1 サイクルで行う処理が増える．しかし，本方式により，レジスタファイルからの値の読み出しにかかる時間が削減されるため，動作周波数への影響はないと予想される．

本方式では，物理レジスタファイルへの振り分けと，実行ユニットの振り分けを行う必要があるため，命令の振り分けを行う Logic が約 2 倍になる．しかし，この Logic の追加によるハードウェア量の増加は本方式で削減されるハードウェア量に比べ十分小さいと思われる．

4.5 レジスタファイルのオーナーの動的な変更

レジスタファイルのオーナーの変更は動的に行われるものとする．レジスタファイルがスレッドから取り上げられた場合，そのレジスタファイルに対しての発行可能ビット (4.2 節参照) は全て無効化される．つまり，その物理レジスタファイルへその命令が発行されることはなくなる．

また，レジスタファイルの割り当ての変更はスレッドの生成，消去時に行われる．この割り当ての決定には，様々なアルゴリズムが考えられるが，今回はできるだけ，各スレッドの所有物理レジスタファイル数が平等になるような割り当てアルゴリズムを用いた．

ただし，本方式では，スレッドが生成されてから消去されるまでの間所有しつづけるレジスタファイルが，少なくとも一つ必要になる．これは，各物理レジスタファイルは所有スレッドが変更された後の演算結果のみを持つためである．

図 3 に動的割り当ての概念図を示す．図 3 の (a) はスレッド 0 のみが走行している場合のプロセッサの状態を示している．スレッド 0 が全ての物理レジスタファイルを使用している様子がわかる．(b) はスレッド 0，1，2 が同時に走行中のプロセッサの状態である．スレッド 1，2 が一つずつ，スレッド 0 が二つの物理レジスタファイルを所有している様子がわかる．

4.6 書き込みポートの制限

本方式は，スレッド番号により，使用可能な実行ユニットを制限することでさらにレジスタファイルのポートを削減することができる．

本方式では動的にオーナーが変化するため，物理レジスタファイルの 0 番から値を読み出す命令は 0〜2 番までの ALU を使うことを許可するといった，発行元バッファによる制限を課すことができない．そのため，スレッド番号による実行ユニットの使用制限を行う．概念図を**図 6** に示す．

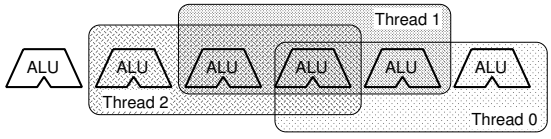


図 6　スレッド番号による実行ユニットの使用制限

この例では各スレッドは三つの ALU の使用が許可されている．このような制限を実現するには，実行ユニットへの発行の際にスレッド番号の判定が必要になる．また，実行ユニットへの命令の振り分けはレジスタリードと同サイクルで行うため，場合によっては動作周波数に影響が出ることも考えられるが，ポート数の削減によりレジスタファイルへのアクセス時間が削減されるため問題にならないと予想される．

5. 予 備 評 価

ここではレジスタファイルのハードウェア，IPC の評価を行う．全評価で固定のパラメータは**表 1** に示す．

なお，ICOUNT.2.8 とは現在実行中の命令が最も少ない 2 スレッドから 8 命令ずつフェッチするというフェッチポリシーである．今回は話を簡単化するため，整数演算器のみを評価した．評価中で可変であるパラメータは**表 2** のようなものがある．本論文中ではこ

表 1 プロセッサの固定パラメータ

最大動作スレッド数	8
実行ユニット	ALU 6, Load/Store Unit 3, Complex ALU 2, Thread Control Unit 1, Register Transfer Unit 1
合計演算パイプライン本数	13
分岐予測器	gshare, PHT 2K entry, BTB 512 entry 4 set assosiative, GHR 4bit
リネームレジスタ	32 entry
Issue Queue	32 entry
Reorder Buffer	32 entry/thread
Retire	4/thread
フェッチポリシー	ICOUNT.2.8[10]

れらのパラメータの組み合わせを，*Cc.Ee.Rr.Ww* と表記する．例えば，統合物理レジスタファイル方式は *C1.E288.R26.W13* と表記する．

表 2 可変パラメータ

c	物理レジスタファイルの分割数
e	各物理レジスタファイルあたりのエントリ数
r	各物理レジスタファイルあたりのリードポート数
w	各物理レジスタファイルあたりのライトポート数

5.1 ハードウェアの評価

ここでは本方式を適用した場合の物理レジスタファイルの総面積と遅延の評価を行う．評価には CACTI3.2[8] を使用し，プロセスルールは $0.13\mu m$ を仮定した．評価結果は**表 3** に示す．

レジスタファイルの総面積

レジスタファイルの総面積は物理レジスタファイル一つあたりの面積に物理レジスタファイルの分割数 c を乗じたものである．

統合物理レジスタファイル方式に比較し，提案方式の全てのパラメータの組み合わせで，レジスタファイルの面積が減少している．特に *C8.E64.R2.W8* では 73%削減されている．パラメータにより削減効果に差はでるものの，本方式はレジスタファイルの総面積の削減に効果があるといえる．

レジスタファイルのアクセス時間

全てのパラメータの組み合わせで従来方式より改善されている．最大で統合物理レジスタファイル方式に比較し，70%短縮されている．レジスタファイルのアクセス時間の改良には本方式は十分な効果があるといえる．

5.2 IPC の評価

ベンチマークプログラムを走らせることで各パラメータでの IPC を測定した．評価には SPLASH-2[11] の radix ソートを使用した．プログラム中の浮動小数点演算は gcc の soft-float を使用しエミュレートした．また，プロセッサコアの性能の変化を調査するため，キャッシュは 100%ヒットすると仮定した．radix ソートのキー数は 1000，radix は 16 とした．なお，gcc のバージョンは 3.2，最適化オプションには “-O2 -mips2” を使用した．

5.2.1 測定環境

IPC の測定は命令レベルシミュレータによって行った．シミュレータは C 言語で記述した．

命令セットは OChiMuS/PE[13] のものを使用した．この ISA は MIPS32 をベースにしたもので，スレッド制御命令を持つ．また，各スレッドは LTN(Logical Thread Number) という各スレッドに一意に振られた番号で管理される．

また，評価にはスレッドライブラリ MULiTh[14] を使用した．

5.2.2 評価結果

評価結果を**図 7** に示す．分割物理レジスタファイル方式では，統合物理レジスタファイル方式に比較し，IPC が最大で約 20%低下している．これは，各スレッドが柔軟に命令を発行することができないことが原因と考えられる．例えば 8 スレッドが走行中の場合，分割物理レジスタファイル方式では，各スレッドは同サイクルでリードポート数/2 の命令しか発行できない．一方，統合物理レジスタファイル方式では，同時に走行中のスレッド数にかかわらず，実行ユニットがあいていれば命令を発行できる．このような制限が性能に影響を与えていると考えられる．

しかし，レジスタファイルのアクセス時間がクリティカルパスになると仮定した場合，表 3 を見ると分かる通り，統合物理レジスタ方式に比べ 3 倍程度の動作周波数を出すことができるため，全体の処理能力は分割物理レジスタファイル方式の方が優れているといえる．

物理レジスタファイル一つあたりのリードポートが多い方が高い性能が出ていることが分かる．しかし，リードポートを増加させるとレジスタファイルの総面積が増加する．

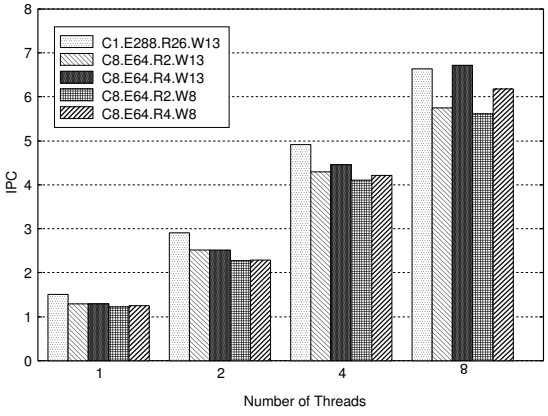


図 7 各構成での IPC

6. 関連研究

Compaq 社の Alpha21264[2] では物理レジスタファイルや実行ユニットなどのリソースを二つのクラスタへ分けることで，レジスタファイルのポート数の削減を行っている．本方式はこの技術を SMT 用へ拡張したものである．

Washinton 大学の Redstone らは，物理レジス

表 3 ハードウェア見積もり結果

	Total Area (cm^2)	Relative Area (%)	Access Time (ns)	Relative Access Time (%)
C1.E288.R26.W13	0.65	100	2.93	100
C8.E64.R2.W13	0.37	57	1.05	36
C8.E64.R4.W13	0.57	87	1.09	37
C8.E64.R2.W8	0.17	27	0.87	30
C8.E64.R4.W8	0.24	38	0.91	31

タファイルの規模を減少させる方法として，Mini-thread[6] を提案している．しかし，この方法ではコンパイラや OS の変更が必要であるという欠点がある．本方式ではそのようなものは必要がないため，優位であると考える．

本方式では，大規模なプロセッサのレジスタファイルの改良案として，ポートを削減することを提案したが，文献 3)5) 15) では物理レジスタファイルのエントリ数を削減する方法を提案している．なお，本稿で提案した方式はこれらと排他的なものではないので，併用が可能である．

7. おわりに

本稿では，SMT プロセッサにおけるレジスタファイルのリードポートを削減する方法として，動的なレジスタファイルの割り当てを提案した．この方式により，レジスタファイルの総面積，アクセス時間を大幅に改善できることが分かった．また，本方式は実行ユニットの利用制限によるライトポートの削減と組み合わせることで，さらに効果を上げることができることを示した．また，本方式による IPC の低下は最大でも 20%程度であった．

本方式では，リネーミングレジスタが複製されてしまうため，統合物理レジスタファイル方式に比較し物理レジスタファイルの総エントリ数が増加する．このエントリ数の増加を軽減するため，物理レジスタファイルのエントリ数を減らす技術との組み合わせを検討したい．

本方式によって追加されるハードウェアが，プロセッサへ与える影響をさらに追及したい．

参考文献

1) Hinton, G., Sager, D., Upton, M., Boggs, D., Carmean, D., Kyker, A. and Roussel, P.: The Microarchitecture of the Pentium4 Processor, *Intel Technology Journal*, Vol. 1st quarter (2001).
2) Kessler, R. E.: The Alpha 21264 Microprocessor, *IEEE Micro*, Vol. 19, No. 2, pp. 24–36 (1999).
3) Lo, J. L., Parekh, S. S., Eggers, S. J., Levy, H. M. and Tullsen, D. M.: Software-Directed Register Deallocation for Simultaneous Multithreaded Processors, *IEEE Transactions on Parallel and Distributed Systems*, Vol.10, No.9, pp. 922–933 (1999).
4) Marr, D., Binns, F., Hill, D., Hinton, G., Koufaty, D., Miller, J. and Upton, M.: Hyper-Threading Technology Architecture and Microarchitecture: A Hypertext History, *Intel Technology Journal*, Vol. 6, No. 1 (2002).
5) Monreal, T., Gonzalez, A., Valero, M., Gonzalez, J. and Vinals, V.: Delaying Physical Register Allocation through Virtual-Physical Registers, *MICRO-32:*, pp. 186–192.
6) Redstone, J., Eggers, S. and Levy, H.: Mini-threads: Increasing TLP on Small-Scale SMT Processors, *Proc. of the Ninth HPCA*, pp. 19–30 (2003).
7) Rixner, S., Dally, W. J., Khailany, B., Mattson, P., Kapasi, U. J. and Owens, J. D.: Register Organization for Media Processing, *Sixth HPCA*, pp. 8–12 (2000).
8) Shivakumar, P. and Jouppi, N.P.: CACTI 3.0: An Integrated Cache Timing, Power, and Area Model, *WRL Research Report* (2001).
9) Tullsen, D. M., Eggers, S. J. and Levy, H. M.: Simultaneous Multithreading: Maximizing On-Chip Parallelism, *Proc. of the 22nd ISCA*, pp. 392–403 (1995).
10) Tullsen, D., Eggers, S., Emer, J., Levy, H., Lo, J., and Stamm, R.: Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor, *Proc. of the 23rd ISCA*, pp. 191–202 (1996).
11) Woo, S. C., Ohara, M., Torrie, E., Singh, J.P. and Gupta, A.: The SPLASH-2 Programs: Characterization and Methodological Considerations, *Proc. of the 22nd ISCA*, pp. 24–36 (1995).
12) Yeager, K.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, Vol. 16, No. 2, pp. 28–40 (1996).
13) 河原, 佐藤, 並木, 中條: システムソフトウェアとの協調を目指すシングルチップマルチスレッドアーキテクチャの構想, コンピュータシステムシンポジウム, Vol. 2002, No. 18, pp. 1–8 (2002).
14) 笹田, 佐藤, 河原, 加藤, 大和, 中條, 並木: マルチスレッドアーキテクチャにおけるスレッドライブラリ の実装と評価, *SACSIS2003*, pp. 13–20 (2003).
15) 大熊, 片山, 小林, 安藤, 島田: 頻出値を利用した物理レジスタの静的共有化手法, *SACSIS2003*, pp. 291–298 (2003).