社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2005－ＳＬＤＭ－119 (22)
2005／3／18

Responsive Multithreaded Processorの分岐予測器の設計と実装

中村 拓† 伊藤 務†† 新井 誠一†† 山崎 信行†

† 慶應義塾大学 理工学部 情報工学科
〒 223-8522 横浜市港北区日吉 3-14-1
†† 慶應義塾大学大学院 理工学研究科 開放環境科学専攻 コンピュータ科学専修
〒 223-8522 横浜市港北区日吉 3-14-1
E-mail: †hiraku@ny.ics.keio.ac.jp

あらまし 我々はリアルタイム処理を目的として，優先度を持った複数スレッドの同時実行が可能な Responsive Multithreaded Processor （RMT プロセッサ）の研究を行っている．現在の RMT プロセッサは各スレッドに対しそれぞれ独立した分岐予測器が実装されているため，分岐予測器のハードウエア量は大きくなってしまっている．そこで，本研究ではスレッドの優先度情報を利用して最高優先度スレッドが低優先度スレッドから受ける干渉を抑える機構を有したスレッド間共有分岐予測器の設計と実装を行う．これにより最高優先度スレッドの予測精度を維持したままハードウエア量を約 70%削減することに成功した．

キーワード リアルタイムシステム, Responsive Multithreaded Processor, 分岐予測

Design and Implementation of Branch Predictor of Responsive Multithreaded Processor

Hiraku NAKAMURA†, Tsutomu ITOU††, Seiichi ARAI††, and Nobuyuki YAMASAKI†

† Department of Information & Computer Science, Faculty of Science & Technology, Keio University
†† Depertment of Computer Science, Graduate School of Keio University
E-mail: †hiraku@ny.ics.keio.ac.jp

Abstract Responsive Multithreaded Processor (RMT Processor) that can exuecute eight threads simultaneously with priority is researched aiming at real-time processing in our lablatory. Because branch predictors are independently implemented for each thread on a present RMT Processor, the amount of hardware of the branch predictor is increases. Then, we design and inplement the shared branch predictor between threads that have the mechanism that suppresses the interference between threads by using priority information. As a result, it succeeded in about 70% of the amount of hardware reduction with high accuracy of branch prediction of the highest priority thread.

Key words Real-time System, Responsive Multithreaded Processor, Branch Prediction

1. はじめに

リアルタイム処理ではデッドライン内で処理を完了させる必要があり，そのため優先度の高いスレッドを優先的に処理しなければならない．

RMT プロセッサは優先度付きマルチスレッディングにより，高いリアルタイム性能の実現を目標にしている．しかし，Simultaneous Multithreading （SMT) [1] などと同じように複数スレッドのコンテキストなどをハードウエアで保持する必要があるため，ハードウエア量が増加してしまうという問題点も存在する．

そこで，本論文では優先度の高いスレッドに優先的にハードウエア資源を割り当てる機構を実装することで高優先度スレッドの分岐予測ミス率に影響を与えることなく，ハードウエア量の削減が可能なリアルタイムシステム用プロセッサ向けの分岐予測器を設計，実装する．また，この機構を利用してより高精度な分岐予測器をハードウエア量の増加を抑えながら実装する．

2. 背 景

2.1 リアルタイムシステム

リアルタイムシステムでは様々な処理に応じてそれぞれ設定される制限時間内（デッドライン）に処理を終了させる機能が

必要になる．また，リアルタイムシステムはハードリアルタイムとソフトリアルタイムの二種類に分類される．ハードリアルタイムは処理時間がデッドラインを越えてしまうと処理の結果が全く無意味なものになってしまい，システム全体に対して深刻なダメージを与えてしまう．それに対し，ソフトリアルタイムでは処理時間がデッドラインを越えてしまうと結果の有用性は減少してしまうものの，システム全体に深刻なダメージを与えるようなことはない．そのため，頻度の低いデッドラインミスであれば容認される場合もある．

このように，リアルタイムシステムでは処理に時間的制約をつけることでシステム全体の性能を保証することを行っている．

2.2 リアルタイムシステムにおけるプロセッサ

リアルタイム OS では複数のスレッドを切り替えて処理が行われる．そのとき，スケジューラの実行時間やスレッド切り替えにかかる時間は，実際に行われる処理の内容にかかわらず必ず発生するオーバーヘッドとなってしまう．このため，コンテキストスイッチをハードウエアで行うことにより，コンテキストスイッチのオーバーヘッドを削減することが重要であると考えられる．また，根本的に同時実行可能なスレッド数を複数にすることによりコンテキストスイッチの回数自体を削減するという方法も考えられる．同時に，ソフトリアルタイムでは全体のスループットが重要視されるので，プロセッサ全体の処理能力は最低でも汎用プロセッサ程度のものでなければならない．

複数スレッドをプロセッサが保持することが可能であるという点を考えると SMT が候補として挙げられるが，マルチスレッドプロセッサはシステム全体の性能を重視して設計されたアーキテクチャであり，シングルスレッドでの性能に問題がある．そこで，SMT に優先度を導入することで高優先度のタスクから実行することで，この問題を解決しリアルタイム実行を可能にすることができる．また，その際複数スレッドが並列実行されるためプロセッサ使用率の向上や，シングルスレッドの性能向上によるスケジューラビリティの向上といった利点がある．

2.3 Responsive Multithreaded Processor

そこで我々は，実行するスレッドそれぞれに優先度を持たせ，高優先度スレッドのから優先的に実行を行う優先度マルチスレッディングによりリアルタイム性を実現する Responsive Multithreaded Processor（RMT プロセッサ）[2] の研究を行っている．この RMT プロセッサのブロック図を図 1 に，特徴を表 1 に示す．

しかしマルチスレッドプロセッサである RMT プロセッサでは，複数スレッド分のコンテキストなどをハードウエアで保持しなければならないため，ハードウエアの複雑さが増し，ハードウエア量も増加してしまうという問題がある．

また，図 2 に示すように現在の RMT プロセッサは 8 つのアクティブスレッドに対してそれぞれ独立してデコーダーや分岐予測器が実装されている．特に分岐予測器は内部に分岐予測のためのテーブルを持っているため，単独でも比較的サイズが大きく，それが 8 スレッド分実装されることでハードウエア量的にはかなり大きなものになる．また，分岐予測精度を向上させるためテーブルのエントリ数を増加させ，より分岐予測精度の

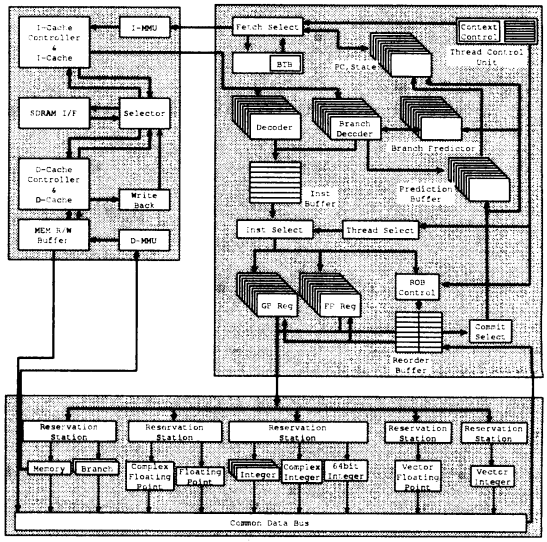

図 1 RMT プロセッサのブロック図

表 1 実装を行う RMT プロセッサの各パラメータ

スレッド数	アクティブ 8，バックアップ 32
命令フェッチ数	8 命令/clock
命令デコード数	8 命令/clock
命令発行数	4 命令/clock
命令コミット数	4 命令/clock
演算ユニット	4 integer 1 complex integer 1 64bit integer 1 FP 1 complex FP 2 branch 1 memory 1 vector integer 1 vector PF
パイプライン	13 ステージ
レジスタ	integer 32bit×32entry×8thread FP 64bit×32entry×8thread
命令キャッシュ	32KB
データキャッシュ	32KB
優先度	256 段階

高い複雑な分岐予測器を実装しようとした場合，ハードウエア量はさらに増加してしまう．

2.4 分岐予測器

分岐予測器は分岐命令がデコードされる際，その分岐命令が分岐条件にマッチして実際に分岐した結果，PC が分岐命令によって指定された番地に飛ぶ（taken）となるか，実際には分岐せずに PC はそのまま（not taken）であるかの予測を行う機構である．RMT プロセッサでは命令のデコードと分岐予測がパイプラインの 12 ステージ目で命令コミットが行われ分岐結果が確定するため，分岐予測が外れた場合 12 クロックサイクル分が無駄になってしまい，大きなペナルティが発生してし

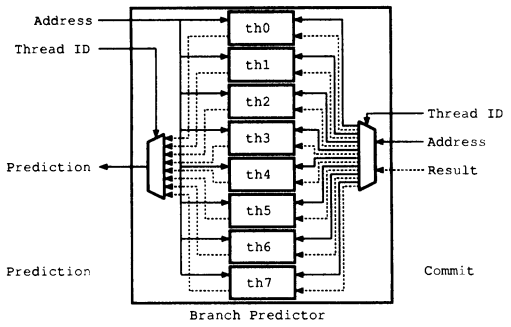

図 2　独立分岐予測器

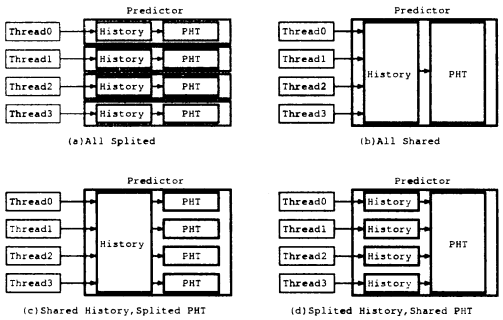

図 3　SMT における分岐予測器の構成

まう．そのため，一般的に分岐予測器には高い予測精度が求められる．

RMT プロセッサのように複数スレッドの同時実行を行う SMT アーキテクチャのプロセッサでは，各スレッドに対する分岐予測器の構成方法が複数存在する [3] [4]．図 3 にその例を示し，その特徴を以下に挙げる．なお，ここでは分岐予測器をヒストリー（グローバルヒストリーやローカルヒストリー）と Pattern History Table（PHT）に分けて考える．

(a)　完全独立

各スレッドがそれぞれ独立したヒストリーと PHT を持ち，スレッド間の干渉を完全に排除することができる．ただし，各スレッドが利用可能な PHT のサイズは小さくなる．

(b)　完全共有

各スレッドが 1 つのヒストリーと PHT を共有するため，スレッド間での干渉が発生し分岐予測精度が低下する．ただし，各スレッドが利用可能な PHT のサイズは大きくなるため，単一スレッド実行時の分岐予測精度は向上する．

(c)　ヒストリー共有，PHT 独立

各スレッドが 1 つの共有ヒストリーと，それぞれが独立した PHT を持ち，スレッド間の干渉はヒストリーのみに発生する．

(d)　ヒストリー独立，PHT 共有

各スレッドがそれぞれ独立したヒストリーと，1 つの共有 PHT を持ち，スレッド間の干渉は PHT のみに発生する．

現在の RMT プロセッサに実装されている分岐予測器は 2-bit Branch Predictor なのでヒストリーレジスタを持たないが，各スレッドにそれぞれ独立して実装されているので，この中では完全独立と考えることができる．

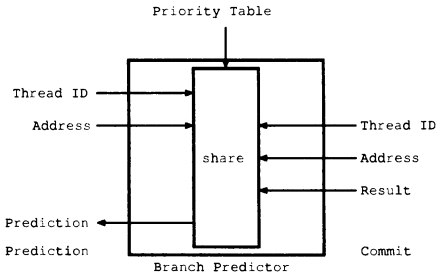

図 4　優先度共有分岐予測器

3.　設計・実装

分岐予測器を複数のスレッドで共有する場合，複数スレッド間で分岐予測器内のテーブルを共有することになる．このとき，テーブルを単純に共有するだけではスレッド間で干渉が発生してしまい，分岐予測精度が低下してしまう．

このスレッド間の干渉は，分岐命令コミット時に複数のスレッドがテーブルの同一のエントリに対して重複して書き込みを行ってしまうために生じるものである．

そこで本研究では低優先度スレッドの分岐命令コミットにより，それ以前に高優先度スレッドが書き込んだエントリに対して書き込みが行われるのを防ぐため，各エントリにスレッド ID 情報を持たせ，低優先度スレッドのコミットが高優先度スレッドのスレッド ID を持ったエントリを上書きするのを禁止する．この優先度共有分岐予測器の概略図を図 4 に示す．独立分岐予測器において入力されていた情報に加え，スレッドの優先度情報を入力し，分岐予測器本体を 1 つにまとめている．

この機構を実装することで優先度の高いスレッドほどテーブルのエントリを自由に使用できるようにし，リアルタイム性を保つため最高優先度スレッドの分岐予測精度を維持したままハードウエア量の削減を行う．

しかしこれだけでは高優先度スレッドの分岐予測精度が保たれる代わりに，低優先度スレッドの分岐予測精度はさらに低下してしまい，全体的な分岐予測精度が低下してしまうものと考えられる．そこでこれを防ぐために優先度情報を持たせたテーブルをさらに set associative 化した共有分岐予測器を実装する．

ここで優先度情報を持たせたテーブルのデータ構造を図 5 に示す．

(a)　優先度で干渉を排除

分岐予測器のテーブルに 1bit の有効/無効の判定ビットと 3bit のスレッド ID を付加している．また，分岐予測用カウンタは PHT であれば通常 2bit，Branch History Table（BHT）であればローカルヒストリーの長さと同じになる．分岐命令コミット時にはスレッド ID からそのスレッドの優先度を参照する．

(b)　優先度と set associative 化で干渉を排除

分岐予測器のテーブルに 1bit の有効/無効の判定ビットと 2bit

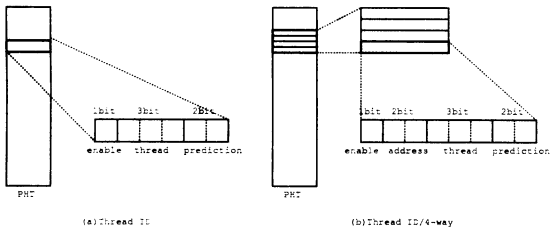

図 5　共有テーブルとエントリのデータ構造

の分岐命令アドレスの下位ビット，3bit のスレッド ID を付加している．分岐予測用カウンタは PHT であれば通常 2bit，BHT であればローカルヒストリーの長さによって決まる．分岐予測，分岐命令コミット時には，アドレスの下位ビットを利用して way を選択し，さらにスレッド ID からそのスレッドの優先度を参照する．

また，実際に図 5(a) を実装した優先度共有分岐予測器は以下のような手順で分岐予測，分岐命令コミットを行う．

- 分岐予測

（1） 分岐命令アドレスの下位ビットをインデックスとしてエントリを参照

（2） そのエントリを使って分岐予測

- 分岐命令コミット

（1） 分岐命令アドレスの下位ビットをインデックスとしてエントリを参照

（2） そのエントリが無効であれば書き込み

（3） 有効であってもそのエントリが持つスレッド ID の優先度が自スレッドの優先度以下であれば上書き

（4） それ以外の場合テーブルの更新は行わない

同様に，図 5(b) を実装した 4-way 優先度共有分岐予測器は以下のような手順で分岐予測，分岐命令コミットを行う．

- 分岐予測

（1） 分岐命令アドレスの下位ビットをインデックスとしてバンクを参照

（2） 参照されたエントリの中からアドレスとスレッド ID が一致するエントリを検索

（3） ヒットした場合そのエントリを使って分岐予測

（4） ヒットしなかった場合その分岐命令は taken と予測

- 分岐命令コミット

（1） 分岐命令アドレスをインデックスとしてバンクを参照

（2） 参照されたエントリの中からアドレスとスレッド ID が一致するエントリを検索

（3） ヒットした場合そのエントリに書き込み

（4） ヒットしなかった場合無効なエントリを検索

（5） ヒットした場合そのエントリに書き込み

（6） ヒットしなかった場合さらに優先度の低いスレッドのスレッド ID を持ったエントリを検索

（7） ヒットした場合そのエントリに書き込み

（8） ヒットしなかった場合テーブルの更新は行わない

表 2　各スレッドで実行するプログラム

th1	th2	th3	th4	th5	th6	th7
matrix	md5	sort	gzip	matrix	md5	sort
md5	matrix	sort	gzip	matrix	md5	sort
sort	matrix	md5	gzip	matrix	md5	sort
gzip	matrix	md5	sort	matrix	md5	sort

4. 評価・考察

4.1 評価方法

本研究は RMT プロセッサ上に実装したものであり，評価についても実装後の RMT プロセッサ上でプログラムを実行することにより行なう．

評価は NC-Varilog を用いた RTL シミュレーションによって行なった．

評価は以下に示す 6 種類の分岐予測器に対して行う．

- 2-bit Branch Predictor [5]
- Gshare Branch Predictor [6](Global History:3bit)
- PAg Branch Predictor [7](Local History:8bit)
- Tournament Branch Predictor [8]
  - 2-Bit / Gshare
  - 2-Bit / PAg
  - Gshare / PAg

また，これらの分岐予測器を以下のような条件で実装してそれぞれを評価する．

- スレッド間独立 256 エントリ × 8 スレッド
- スレッド間単純共有 256 エントリ
- スレッド間優先度共有 256 エントリ
- スレッド間優先度共有（4-way） 256 エントリ

評価はスレッド 1 が最高優先度，スレッド 7 が最低優先度となるような階段状の優先度をもたせた 7 つのスレッドで以下に示す 4 種類のプログラムを同時に実行することで行う．

- matrix: long 型の 32×32 の行列の積を計算
- md5: 8192byte のデータの MD5 を計算
- sort: long 型の 256 エントリのデータをクイックソート
- gzip: 128byte のデータを圧縮（gzip-1.2.4a より圧縮を行うコードを移植したもの）

これらのプログラムを 2 に示すように最高優先度スレッドのプログラムが異なる 4 種類の組み合わせでそれぞれ実行し，各スレッドにおける分岐予測ミス率の計測を行う．なお表中の th1 が最高優先度スレッド，階段状に優先度が低下していき th7 が最低優先度スレッドとなっている．

4.2 評　　価

図 6 に 2-bit Branch Predictor における各プログラムの組み合わせでの最高優先度スレッドプログラムの分岐予測ミス率を示す．

グラフの縦軸は分岐予測ミス率，横軸は最高優先度プログラムを示しており，さらに各グラフは左から独立，単純共有，優先度共有，4-way 優先度共有を示している．

このグラフから全てのプログラムにおいて各スレッドに独立

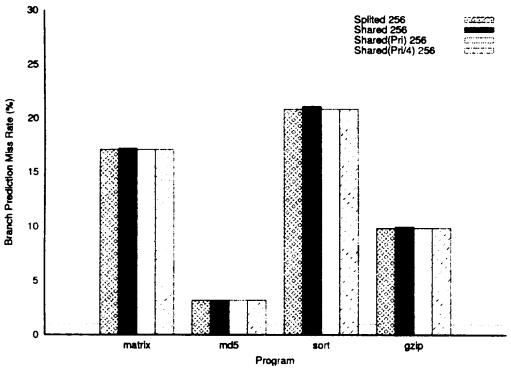

図 6　最高優先度スレッドの分岐予測ミス率（2-bit）

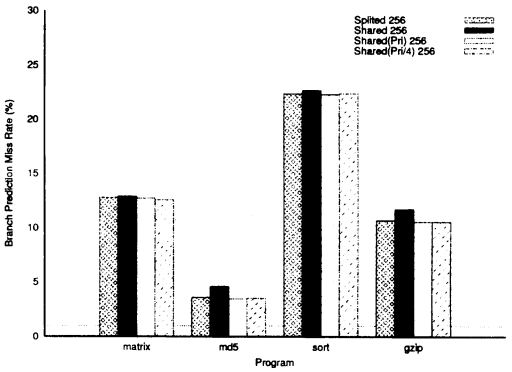

図 8　最高優先度スレッドの分岐予測ミス率（PAg）

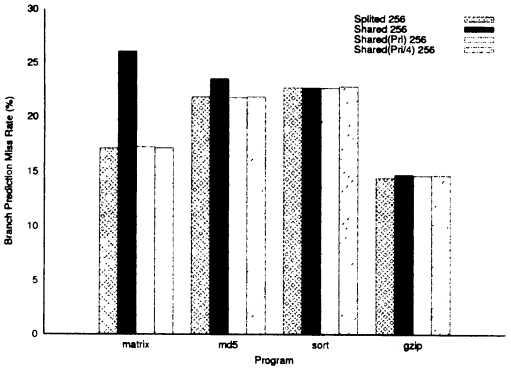

図 7　最高優先度スレッドの分岐予測ミス率（Gshare）

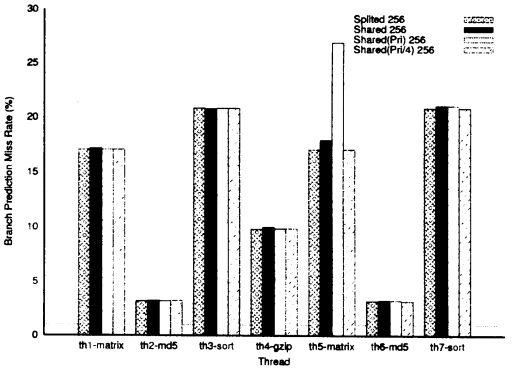

図 9　matrix が最高優先度のときの分岐予測ミス率（2-bit）

して実装した分岐予測器に対し，単純共有の分岐予測ミス率がわずかではあるが 0.1%から 0.5% 高くなってしまっている．それに対し優先度共有 2 種の分岐予測ミス率は独立とほぼ同じ分岐予測ミス率に抑えられている．

次に，Gshare Branch Predictor における各プログラムの組み合わせでの最高優先度スレッドプログラムの分岐予測ミス率を図 7 に示す．

独立タイプと単純共有を比較すると，matrix で約 9%，md5 で約 2%単純共有の方が分岐予測ミス率が高く，2-bit Branch Predictor の場合と比較して共有化による分岐予測ミス率の上昇の割合が非常に大きくなっているのがわかる．それに対し，優先度共有 2 種では独立タイプとほぼ同じ分岐予測ミス率に抑えられている．

さらに，PAg Branch Predictor における各プログラムの組み合わせでの最高優先度スレッドプログラムの分岐予測ミス率を図 8 に示す．

このグラフから全てのプログラムにおいて各スレッドに独立して実装した分岐予測器に対し，単純共有の分岐予測ミス率が 0.1%から 1.0%高くなってしまっている．それに対し優先度共有 2 種の分岐予測ミス率は独立とほぼ同じかそれ以下の分岐予測ミス率に抑えられている．

また，これらを組み合わせた 3 種類の Tournament Branch Predictor でも最高優先度スレッドにおいては，単純共有では独立型と比較して分岐予測ミス率が増加してしまうが，優先度共有にすることによりその増加を抑えることができていた．

次に，最高優先度スレッドを matrix とした場合の 2-bit Branch Predictor の全スレッドにおける分岐予測ミス率を図 9 に示す．

このグラフからは，最高優先度スレッドでは共有分岐予測器に優先度情報を持たせることで分岐予測ミス率の上昇を抑えることができているが，その弊害として低優先度スレッドの Thread5 の matrix において分岐予測ミス率の大幅な上昇が見られる．これは Thread5 の優先度が低いにもかかわらず大量に命令が発行されたため，コミット時に分岐予測器への書き込みの多くがブロックされたため，大幅な分岐予測ミス率の上昇につながったものと考えられる．しかし，優先度共有分岐予測器を 4-way 化することでこの分岐予測ミス率の低下を防ぐことができている．

最後に，各スレッドに独立して実装した場合の各分岐予測器における分岐予測ミス率を図 10 に示す．

グラフの縦軸は分岐予測ミス率，横軸はプログラムとそれらの平均を示している．さらに，各グラフは左側から 2-bit, Gshare, PAg, 2-bit/Gshare, 2-bit/PAg, Gshare/PAg を示している．

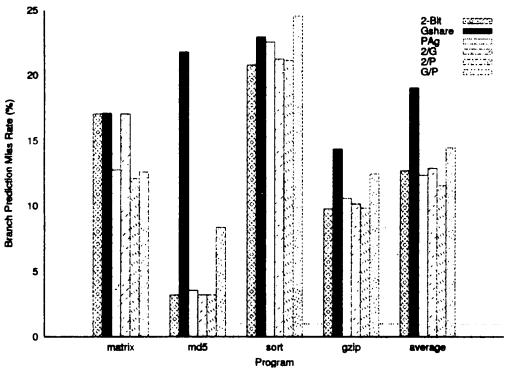

図 10　各分岐予測器の分岐予測ミス率

表 3　論理合成結果

		面積 (μm^2)	遅延時間 (ns)
2-bit	独立	427,089	2.39
	優先度共有	127,819	2.86
	優先度共有 (4-way)	225,341	2.87
2-bit/PAg	独立	2,790,994	3.50
	優先度共有	615,584	3.88
	優先度共有 (4-way)	741,675	5.32

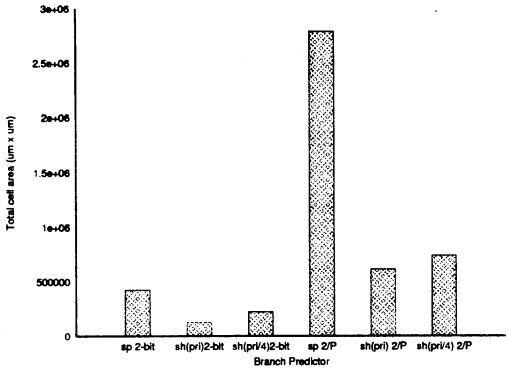

図 11　面　　積

今回評価を行ったプログラムの平均で考えると，もともと RMT プロセッサに実装されていた 2-bit Branch Predictor と比較して 2-bit と PAg を組み合わせた Tournament Branch Predictor では分岐予測ミス率を約 2%削減できているのがわかる．

4.3　論理合成結果

表 3 と図 11 に各スレッドに 256 エントリのスレッド間独立 2-bit Branch Predictor と平均分岐予測ミス率が最も低かった 2-bit と PAg を組み合わせた Tournament Branch Predictor, 256 エントリの優先度スレッド間共有 2-bit Branch Predictor と 2-bit と PAg を組み合わせた Tournament Branch Predictor をそれぞれ論理合成した結果を示す．

このことから，分岐予測器を優先度情報を考慮してスレッド間で共有することにより 2-bit Branch Predictor で 70%, Tournament Branch Predictor で 78% 削減することができた．また，これを 4-way 化した場合でも 2-bit Branch Predictor で 48%，Tournament Branch Predictor で 74%削減することができた．

さらに，もともと RMT プロセッサに実装されていた独立型 2-bit Branch Predictor からの Tournament Branch Predictor 実装による面積の増加を 44% に抑えることができた．

5. ま と め

以上より，スレッド間の干渉を抑える機構を実装したスレッド間共有の分岐予測器により，最高優先度スレッドの分岐予測精度をスレッド間独立の分岐予測器の分岐予測精度とほぼ同じレベルを維持しながら実装面積の削減に成功した．

また，今回実装した Tournament Branch Predictor はこれまで RMT プロセッサに実装されていた 2-bit Branch Predictor よりも高い分岐予測精度を達成することができた．また，もともと実装されていた独立型 2-bit Branch Predictor からの Tournament Branch Predictor 実装によるハードウエア量の増加を 44%に抑えることに成功した．

謝辞　本論文の研究は，文部科学省の科学技術振興調整費の支援による．また本研究の一部は，科学技術振興機構 CREST の支援による．

文　　献

[1] D.M.Tullsen, S.J.Eggers and H.M.Levy.: "Simultaneous multithreading: Maximizing on-chip parallelism.", The 22nd Annual International Symposium on Computer Architecture, pp. 392–403 (1995).

[2] 山崎："Responsive multithreaded processor の全体設計", 電子情報通信学会技術研究報告:実時間処理に関するワークショップ (RTP2004) 電子情報通信学会, pp. 9–14 (2004).

[3] M. H. L. Matt Ramsay, Chris Feucht: "Exploring efficient smt branch predictor design", Workshop on Complexity-Effective Design, in conjunction with ISCA (2003).

[4] M. F. Jayanth Gummaraju: "Branch prediction in multi-threaded processors", The 2000 International Conference on Parallel Architectures and Compilation Techniques, IEEE Computer Society, p. 179 (2000).

[5] D. A. P. John L. Hennessy: "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Publishers (2002).

[6] S. McFarling: "Combining branch predictors", Technical report, Western Research Laboratory (1993).

[7] Y. N. P. Tse-Yu Yeh: "Alternative implementations of two-level adaptive branch prediction", The 19th Annual International Symposium on Computer ArchitectureInternational Conference on Computer Architecture, ACM Press, pp. 124–134 (1992).

[8] R. E. Kessler: "The alpha 21264 microprocessor", IEEE Micro (1999).