2006－ARC－170 (17)
2006／11／30

SMTプロセッサにおけるキャッシュメモリリプレース方式の動的切り替え

小笠原 嘉泰† 佐藤 未来子†
並木 美太郎† 中條 拓伯†

SMTプロセッサは，複数のスレッドで演算器やキャッシュメモリを共有し，性能向上を目指している．ところが，キャッシュメモリの共有が原因で，キャッシュラインにおけるスレッド間競合が発生し，性能が低下する問題がある．スレッド間競合を抑え性能向上を促す手法として，スレッドごとにリプレース可能な領域を制限するキャッシュリプレース方式があるが，キャッシュ容量やプログラムによっては従来の疑似 LRU と比べ性能低下を引き起こす．そこで本論文では，その SMT プロセッサ向けキャッシュリプレース方式の利点と問題点に着目し，プログラム実行中に疑似 LRU との動的切替を行うことで，性能向上を目指す．リプレース動的切替方式として，スレッド間競合ミスを切替パラメータとする動的切替方式と，セットごとにリプレース方式を切り替える動的切替方式を提案し，設計した．評価の結果，各動的切替方式は有効に動作し，SMT プロセッサ向けリプレース方式で発生した性能低下を抑え，さらに疑似 LRU と比べ最大 1.48 倍の性能向上をもたらした．また，各動的切替方式を実装しハードウェアコストを見積った結果，わずかなハードウェア増加量で各方式を実現できることを示した．

Dynamic Switch Strategies of Cache Replacement for an SMT Processor

Yoshiyasu Ogasawara ,† Mikiko Sato ,† Mitaro Namiki †
and Hironori Nakajo †

An SMT processor aims to gain higher processor performance by executing parallel threads. However, the increasing cache misses caused by sharing the cache memory brings performance degradation. There are replacement strategy of cache memory to suppress thread confict misses, but it cause performance degradation depending on circumstances. In this paper, we have proposed dynamic switch strategies of cache replacement to aim higher performance. As a result, dynamic switch strategy shows 1.48 times as high performance as conventional replacement strategy. And dynamic switch strategies can be implemented with low hardware cost.

1. はじめに

近年，プロセッサアーキテクチャとして，スレッドレベル並列性 (TLP : Thread Level Parallelism) を利用したマルチスレッドアーキテクチャに注目が集まっている．マルチスレッドアーキテクチャとして，チップマルチプロセッサ (CMP : Chip Multi Processor) や，Simultaneous MultiThreading (SMT) プロセッサ[1] がある．これらプロセッサは，1チップ上で複数のスレッドを並列に実行し，性能向上を目指している．特に SMT プロセッサは，1チップで複数のハードウェアコンテキストを持ちつつ各種演算器やキャッシュメモリなどスレッド間で共有できる資源をできる限り共用することで，資源の有効活用とプロセッサの性能向上を同時に実現している．

しかし，SMT プロセッサには，スレッド間で共有している資源の競合や枯渇が発生する短所がある．具体的には，スレッドのキャッシュライン競合によるキャッシュミスの増加，各種演算器の枯渇があり，そのため，これらが原因でプロセッサの性能が低下してしまう場合がある．

特にキャッシュラインのスレッド間競合による性能低下は深刻であり，それを解決すべくハードウェア，ソフトウェア両面から様々な解決策が提案されてきた[2][9][10]．その解決策の1つとして，我々はスレッド間競合ミスを抑える SMT プロセッサ向けキャッシュリプレース方式を提案している[2]．これは，スレッドごとにリプレース可能な領域を制限し，スレッドの干渉による競合ミスを抑える働きがある．評価の結果，このリプレース方式は有効に動作し，従来の擬似 LRU と比較し全体的に性能向上を実現した．しかしながら，プログラムやキャッシュ容量によっては，一部性能低下を引き起こした．

そこで，本論文では，擬似 LRU と SMT プロセッサ向けリプレース方式のプログラム実行中における動的切替方式を提案する．プログラム実行中に適切なタイミングで両リプレース方式を切り替えることにより，SMT プロセッサ向けリプレース方式で発生した性能低下を抑え，更なる性能向上を目指す．また，提案する動的切替方式を実装し，具体的なハードウェア量を見積り，その実現可能性を検証する．

なお，本論文では，マルチスレッドプロセッサにおいて1つのスレッドを処理する単位を実スレッド (*AT : Architecture Thread*) と定義する．また，マルチスレッドプログラミングやコンパイラによって生成されるスレッドを論理スレッド (*LT : Logical Thread*) として定義する．

2. 研究背景

2.1 SMT プロセッサのキャッシュメモリの問題点

本論文では，前提とする SMT プロセッサとして *OChiMuS PE* を，スレッドライブラリとして *MULiTh* を用いる．OChiMuS PE については文献 3)，MULiTh については文献 4) を参照されたい．

SMT アーキテクチャでは，各スレッドで共有するデータの有効活用を目指し，L1，L2 キャッシュメモリを共有する．ただし，L1-I (Instruction) -キャッシュメモリは，読み込みしか行わず，コヒーレンス維持の必要がないため，OChiMuS PE では実スレッドごとに L1-I-キャッシュメモリを持つ．本論文ではターゲットとするキャッシュメモリとして，L1-D (Data) -キャッシュメモリを扱う．以降，キャッシュメモリという表記は，L1-D-キャッシュメモリを指す．

SMT プロセッサにおけるキャッシュメモリでは，複数のスレッドが並列実行するため，あるスレッドが他のスレッドのデータをリプレースしてしまう実スレッド間の干渉が発生する．この実スレッド間の干渉によりキャッシュライン競合が多発することを，破壊的干渉 (Destructive

† 東京農工大学大学院 工学府
Graduate School of Technology, Tokyo University of Agriculture and Technology

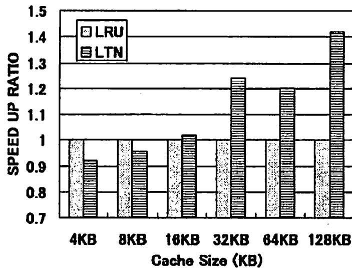


図 1 擬似 LRU と LTN 方式の性能比較

Interference) と定義されており[9)11)]，本論文では，破壊的干渉により発生する競合ミスを破壊的干渉ミス (Destructive Interference Misses) と定義する．SMT プロセッサではキャッシュメモリを各実スレッドで共有するため，破壊的干渉ミスが多発する場合がある．

2.2 LTN 方式の利点と問題点

我々は，SMT プロセッサ向けのキャッシュリプレース方式として，破壊的干渉ミスを抑える LTN (Logical Thread Number) 方式を提案している[2)]．LTN 方式は，実スレッドが保持する LTN (論理スレッド番号) を用いることで，スレッドのリプレース可能なウェイを制限し，実スレッド間の干渉を緩和させる方式である．キャッシュアクセス時は従来と変わらず，全てのウェイに対してアクセスできるようにし，キャッシュミスが発生した場合のみ，リプレース方式として LTN 方式を使用する．そうすることでキャッシュメモリを共有するメリットを損なわず，スレッドの破壊的干渉ミスを抑えることができる．

擬似 LRU (LRU) と LTN 方式 (LTN) の性能比較として，文献 2) の評価結果の一部を図 1 に示す．プログラムは，論理スレッドを 8 個に分割した行列乗算を用いている．また，プロセッサの実スレッド数を 2 とし，L1-D-キャッシュメモリはウェイ数：4，ブロックサイズ：32B とし，キャッシュ容量を 4KB〜128KB☆と変化させ，評価している．詳細なパラメータは文献 2) を参照されたい．

図 1 は擬似 LRU の性能を基準とし，LTN 方式の性能向上率をあらわす．キャッシュ容量 16KB〜128KB の場合で性能向上しており，特に 128KB の場合，1.42 倍と高い性能向上率を示している．これは LTN 方式が有効に働き，破壊的干渉ミスを抑えることができた結果である．

ところが，キャッシュ容量が 4KB，8KB の場合，性能向上率が 0.92 倍，0.95 倍となり，擬似 LRU と比較し性能が低下している．これは，キャッシュ容量が少ない場合，LTN 方式を使用することで，各スレッドのリプレースできる領域が減少し，リプレース領域の制限が欠点として働いてしまったためである．また，行列乗算は破壊的干渉ミスが多いため，LTN 方式の性能向上率が高いが，破壊的干渉ミスが少ないプログラムは，LTN 方式の性能向上率が低く，場合によっては性能低下を引き起こすことがある．

以上のような LTN 方式の利点と問題点に着目し，それらを考慮した有効なリプレース動的切替方式を提案，設計し，OChiMuS PE 全体の性能向上を目指す．

3. 切替対象キャッシュリプレース方式

本章では，まず，動的切替対象のキャッシュリプレース方式である疑似 LRU と LTN 方式の実現方法について示す．次に，両リプレース方式を同時に実現する方法を示す．

3.1 疑似 LRU の実現

キャッシュメモリのリプレース方式として，LRU 方式，ランダム方式，ラウンドロビン方式など[7)] があるが，性能面において有利である LRU 方式が多用されている．しかし，キャッシュメモリのウェイ数が増加すると，LRU を実現するハードウェア増加量が大きくなってしまうため，ウェイ数が 2 の場合以外は，LRU を擬似的に用いている．本研究では，疑似 LRU として pseudo-LRU[7)] を用いる．pseudo-LRU は 2 分木によって，リプレースウェイを決定する．4 ウェイセットアソシアティブの pseudo-LRU の実現方法を図 2 に示す．

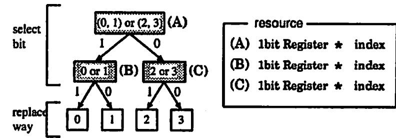


図 2 疑似 LRU の実現方法

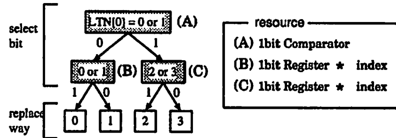


図 3 LTN 方式の実現方法

図 2 中の (A)，(B)，(C) は，1bit ×インデックスのレジスタである．(A)，(B)，(C) のレジスタの更新はキャッシュヒット時に次のように行う．

(1) (A) にヒットしたウェイ番号の上位 1bit を格納する．
(2) 上位ビットが 0 の場合は (B) を，1 の場合は (C) を更新する．格納する数はヒットしたウェイ番号の下位 1bit とする．

リプレースウェイを決定する時，まず (A) のレジスタに 1 が格納されていた場合，(B) を選択し，0 が格納されていた場合，(C) を選択する．次に，(B) が選択された場合，(B) に 1 が格納されていたらウェイ 0 をリプレース，0 が格納されていたらウェイ 1 をリプレースする．(C) が選択されたときも同様であり，(C) に 1 が格納されていたらウェイ 2 をリプレース，0 が格納されていたらウェイ 3 をリプレースする．

このようなアルゴリズムにより，pseudo-LRU は枝の中で最近アクセスしたウェイではない枝を指すことができる．

3.2 LTN 方式の実現

LTN 方式は，論理スレッド番号 (LTN) を用いて破壊的干渉ミスを抑えるリプレース方式である[2)]．リプレースウェイ制限に LTN を用いるが，LTN から n ビットを取り出しリプレースウェイを制限するとき，それを LTN-n 方式と呼ぶ．例えば，LTN から 1 ビットを用いてリプレースウェイを制限するときは LTN-1 方式となり，各実スレッドのリプレース可能ウェイは全てのウェイの半分となる．以降，本論文の LTN 方式という表記は，LTN の最下位 1bit を取り出した LTN-1 方式を指す．

4 ウェイセットアソシアティブの時，LTN 方式の実現方法を図 3 に示す．図 2 の擬似 LRU と比べると，(A) の部分が異なる．LTN 方式の場合，(A) の箇所が，キャッシュアクセスした実スレッドの LTN から取り出したビットの比較器となる．図 3 の場合，LTN の最下位 1bit を判定しているので，論理スレッド (LT) を 8 個に分割する場合，LT0，LT2，LT4，LT6 が (B) を選択し，LT1，LT3，LT5，LT7 が (C) を選択することになる．

比較器を組み込むことになるが，1bit ×インデックス分のレジスタ領域がなくなることになり，ハードウェア量は擬似 LRU に比べ減少する．

3.3 疑似 LRU と LTN 方式の同時実現

疑似 LRU (図 2) と LTN 方式 (図 3) の実現方法の違いは (A) の部分のみとなるため，両リプレース方式を同時に実現することは容易である．両リプレース方式を同時に実現する方法を図 4 に示す．

疑似 LRU と比べると，(D) と (E) の回路が新たに追加される．(D) は LTN 方式を実現するために必要な 1bit 比較器である．また，プログラム実行中に，次章で提案するリプレース動的切替方式から適切なタイミングでリプレースモード (Mode) として，0 または 1 が出力されてくる．それを用い，0 が出力されてきた場合は疑似 LRU を使用するため (A) を選択し，1 が出力されてきた場合は LTN 方式を使用するため (D) を選択する．そのため，(E) として，疑似 LRU と LTN 方式を切り替えるための 1bit 比較器が必要となる．

つまり，疑似 LRU と LTN 方式を同時に実現する場合，疑似 LRU

☆ 文献 2) では，8KB，32KB，128KB を評価している．4KB，16KB，64KB の結果は追加評価による．

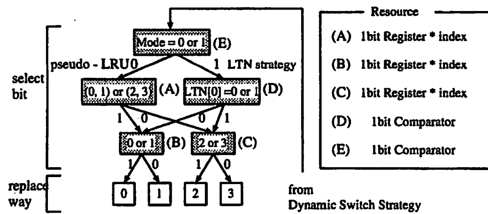


図 4　疑似 LRU と LTN 方式の同時実現方法

と比較し，1bit 比較器が 2 つ追加されるだけであり，ハードウェア増加量は問題とならない．

4. 動的切替方式

本章では，破壊的干渉ミスを切替パラメータとする動的切替方式として **SDI 方式 (Switch by miss ratio of Destructive Interference)** を，セットごとにリプレース方式を切り替える動的切替方式として **SON 方式 (Switch by Occupied Number of ways in set)** を提案し，設計する．

4.1 SDI 方 式

4.1.1 SDI 方式の概要

リプレース動的切替方式として，ここでは破壊的干渉ミスを切替パラメータとする SDI 方式を提案する．破壊的干渉ミス数をプログラム実行中に常に測定し，破壊的干渉ミス率を計算する．計算した結果，破壊的干渉ミス率が一定値以下の場合は疑似 LRU を，一定値を超えた場合は LTN 方式を選択する．

ここで重要となるのが，切替パラメータである破壊的干渉ミス率である．ところが，正確な破壊的ミス率を計算しようとすると除算器を使用しなければならず，それはハードウェア増加量，動作周波数共に多大な悪影響を及ぼす．よって，SDI 方式では，破壊的干渉ミス数と総メモリアクセス数を用いて，以下の計算を行う．

$$Miss_\#_DI > \frac{Total_Memory_Access_\#}{M} \quad (1)$$

$$M = 2^n \quad (2)$$

$Miss_\#_DI$　　　　　　　　: 破壊的干渉ミス数
$Total_Memory_Access_\#$　: 総メモリアクセス数

(1) 式を満たす場合は LTN 方式 (Mode : 1) を選択し，満たさない場合は疑似 LRU (Mode : 0) を選択する．そして，(2) 式の n を変化させることで，切替パラメータである破壊的干渉ミス率を変化させる．また，(1) 式の分母を 2 の指数とすることで，除算器を使用せずに済む．

以上を考慮し，SDI 方式の切替アルゴリズムを以下に示す．

(1)　リプレース方式として，疑似 LRU を初期設定し，プログラムを開始する．
(2)　プログラム実行中，常に破壊的干渉ミス数を測定し，(1) 式を満たすかどうか確かめる．
(3)　(1) 式を満たす場合 (破壊的干渉ミス率がある一定値を超えていた場合)，リプレース方式を疑似 LRU から LTN 方式に切り替える．
(4)　一度，LTN 方式に切り替えたら，プログラム終了まで LTN 方式を使用する．

このような切替アルゴリズムを用いることで，破壊的干渉ミスが多発するプログラムの場合，リプレース方式がすぐに疑似 LRU から LTN 方式に切り替わり，LTN 方式をプログラム開始時から使用した場合に近い性能向上率を実現することができる．一方，破壊的干渉ミスが少ないプログラムの場合，リプレース方式が疑似 LRU から LTN 方式に切り替わることはなく，LTN 方式を使用することによって発生する性能低下を防ぐことができる．

ここで，(2) 式の n を具体的に設定した SDI 方式を **SDI-n 方式**と呼ぶ．例えば，n=6 を設定した場合，SDI-6 方式となり，(1) 式の分母 M は 64 となり，破壊的干渉ミス率が 1/64 (1.56125 %) まで上昇したら，疑似 LRU から LTN 方式にリプレース方式を切り替える．

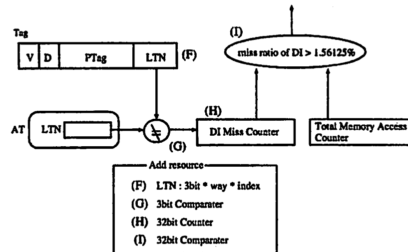


図 5　SDI 方式の実現方法

4.1.2 SDI 方式の設計

SDI 方式の切替アルゴリズムの実現方法を図 5 に示す．

まず，全てのキャッシュラインのタグに LTN を追加する (図 5 の (F))．次に，キャッシュミスとなり，リプレースウェイが決定したら，そのキャッシュラインに保持している LTN とキャッシュアクセスした実スレッドの LTN を比較する (図 5 の (G))．比較した結果，LTN が異なっていたら他のスレッドによるリプレースという判定となり，破壊的干渉ミス数を測定しているカウンタ (図 5 の (H)) をインクリメントする．そして，H が保持している破壊的干渉ミス数とパフォーマンスカウンタが保持している総メモリアクセス数を用いて (1) 式を計算し，設定した破壊的干渉ミス率 (図 5 では 1.56125 %) を超えているかどうかを判別する．超えていたら 1，超えていない場合は 0 を出力する (図 5 の (I))．

この図 5 からの出力は，図 4 の入力となる．SDI 方式のために追加するハードウェア資源は図中の (F)，(G)，(H)，(I) となる．(G)，(H)，(I) の資源はキャッシュメモリにおいて，1 つのみ追加すればよいのでハードウェア増加量は問題ない．唯一，(F) の LTN に注意が必要である．例えば，LT を 8 個に分割した場合は，3bit ×インデックス×ウェイ分用意しなければならないので，ハードウェア増加量は大きくなる．つまり，SDI 方式のハードウェア増加量は (F) に起因する．

4.2 SON 方 式

4.2.1 SON 方式の概要

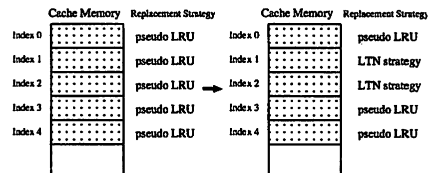


図 6　SON 方式の概要

キャッシュメモリにはホットスポットが存在するため，破壊的干渉ミスが多い場合でも，それが特定のセットのみで多発している場合が考えられる．そこで図 6 のように，セットごとにリプレース方式を動的に切り替える SON 方式を提案する．図 6 では，インデックス 1，2 に多数のスレッドがアクセスしていると仮定し，それらセットのみのリプレース方式を LTN 方式に切り替えている．

セットごとにリプレース方式を切り替える手段として，破壊的干渉ミス数をセットごとに測定する方法が考えられる．しかし，それの実現には，測定用カウンタがインデックス分必要となり，ハードウェア量が大幅に増加してしまうため現実的ではない．

そこで，セットにアクセスしているスレッド数に注目する．SDI 方式と同様にタグに LTN を持たせ，セット中の各ウェイの LTN を比較し，何種類のスレッドのデータがセット中に格納されているか確認する．そのスレッド数に応じて，使用するリプレース方式を決定する．

セット中に格納されているスレッド数 n を切替パラメータとし，n 以上になったらリプレース方式を疑似 LRU から LTN 方式に切り替える SON 方式を SON-n 方式と呼ぶ．4 ウェイセットアソシアティブの場合，SON-2 方式，SON-3 方式，SON-4 方式が考えられ，各方

表 1　セット中のスレッド数によるリプレース方式の切替

スレッド数	ウェイ数	選択リプレース方式		
		SON-2	SON-3	SON-4
1	4	疑似 LRU	疑似 LRU	疑似 LRU
2	3 : 1	LTN 方式	疑似 LRU	疑似 LRU
2	2 : 2	LTN 方式	疑似 LRU	疑似 LRU
3	2 : 1 : 1	LTN 方式	LTN 方式	疑似 LRU
4	1 : 1 : 1 : 1	LTN 方式	LTN 方式	LTN 方式

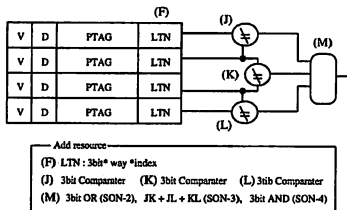


図 7　SON 方式の実現方法

式の切替タイミングを表 1 に示す．セット中のスレッド数が 1 つだけの場合，そのスレッドが 4 つ全てのウェイに格納されていることになる．その場合，LTN 方式を用いる必要が無いため，リプレース方式として疑似 LRU を選択する．そのため，SON-1 方式は存在しない．

以上を考慮した SON 方式の切替アルゴリズムを以下に示す．

(1)　リプレース方式として，疑似 LRU を初期設定し，プログラムを開始する．

(2)　リプレースが必要なキャッシュミスが発生した場合，そのセット中に存在するスレッド数を確認する．

(3)　確認した結果，設定したスレッド数 n 以上のスレッド数がセット中に存在した場合，疑似 LRU から LTN 方式に切り替える．

(4)　一度，LTN 方式に切り替わっても，セット中のスレッド数が設定したスレッド数 n より小さくなった場合，再度 LTN 方式から疑似 LRU に切り替える．

このような切替アルゴリズムを用いることで，ホットスポットやスレッドのアクセス局所性に対応することができ，LTN 方式を適当なセットのみで使用することができる．また，セット中に存在するスレッド数が設定数よりも小さくなった場合，再度疑似 LRU に切り替えることで，適当な期間のみ LTN 方式を活用できる．このように SON 方式は，LTN 方式を適当な場所と期間のみに適用することで，LTN 方式を単体で使用する以上の性能向上率を目指す．

4.2.2　SON 方式の設計

SON 方式の切替アルゴリズムの実現方法を図 7 に示す．図 7 の出力は図 4 の入力となる．

まず，既存のタグに LTN (図 7 の (F)) を追加する．次に，各ウェイの LTN を比較するため，比較器を追加する．ここで，各ウェイの LTN を正確に比較するためには，4 ウェイで 6 個，8 ウェイで 28 個の比較器が必要となる．それに伴って，LTN のポート数を増加させなければならず，多くの LTN を比較することは，ハードウェア量，動作周波数の両方に悪影響を及ぼす．そこで，SON 方式では隣のウェイのみの LTN を比較することで，スレッド数を確認する．そうすることで，正確ではないがおおよそのスレッド数を調べることができ，なおかつ比較器や LTN のポート数を抑えることができる．4 ウェイの場合は，図 7 のように (J), (K), (L) の 3 つの比較器を追加する．

最後に (J), (K), (L) からの入力をもとに，0 (疑似 LRU) または 1 (LTN 方式) を出力する (M) を追加し，SON 方式を実現する．ここでは，(M) に追加する回路を考える．前節で示した表 1 と，(J), (K), (L) から入力をもとに，SON-2, 3, 4 方式の出力を考えると，表 2 のような真理値表が得られる．この真理値表をもとに，回路を簡単化していくと，SON-2 方式は 3bit OR 回路，SON-3 方式は以下の (3) 式，SON-4 方式は 3bit AND 回路が得られる．

$$J \cdot K + J \cdot L + K \cdot L \tag{3}$$

つまり，(M) に追加する回路はどの方式でも簡単な組み合わせ回路となる．SON 方式を実現するためには，(F), (I), (K), (L), (M) の回路が新たに必要となるが，ハードウェア量増加の原因となるのは，SDI 方式同様 (F) の追加 LTN になると考える．

表 2　(M) に追加される回路の真理値表

(J)	(K)	(L)	SON-2	SON-3	SON-4
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1

4.3　SDI 方式と SON 方式の組み合わせ

SDI 方式と SON 方式を同時に実現するためには，図 5 と図 7 の (F)〜(M) の回路が必要となる．

両方式を組み合わせた場合の出力として，本論文では，SDI 方式の出力と SON 方式の出力の AND とし，それを図 4 の入力とする．つまり，どちらかの方式で 0 (疑似 LRU) が出力された場合は 0 を出力し，両方式において 1 (LTN 方式) が出力された場合に限り，1 を出力する．

4.4　動的切替方式の検討

まず，SDI 方式について検討する．切替パラメータである破壊的干渉ミス率の適切な値を求めるため，破壊的干渉ミス率として 0.5 %〜20 % を設定し，予備評価を行った．その結果，1 %〜3 % がよりよい結果をもたらした．その原因として，破壊的干渉ミスが多発するプログラムの場合，そのミス率は 3 %以上になる場合が多く，一方，破壊的干渉ミスが少ないプログラムの場合，そのミス率は 1 %未満となる場合がほとんどであった．そのため，切替パラメータとして 0.5 % を設定した場合，破壊的干渉ミスが少ないプログラムまで疑似 LRU から LTN 方式に切り替わってしまい，性能低下が拡大した．また，切替パラメータとして 5 %以上を設定した場合，破壊的干渉ミスが多いプログラムにおいてもなかなかリプレース方式が切り替わらず，LTN 方式の利点を活かせなかった．これらの結果から，SDI 方式としては，SDI-5 方式 (3.125 %), SDI-6 方式 (1.56125 %) が適切であり，性能低下の抑制および性能向上が見込めると判断した．

次に SON 方式について検討する．切替パラメータとして適切なセット中に存在するスレッド数を求めるため，4 ウェイのキャッシュメモリ上で SON-2,3,4 方式を設計し，予備評価を行った．結果，SON-3,4 方式が高い性能向上率をもたらした．SON-2 方式では多くのセットで切替が発生してしまい，SON-3,4 方式と比べると性能は低かった．そのため，適切な切替パラメータとして，ウェイ数の半分を超える値が妥当と判断した．つまり，4 ウェイでは SON-3,4 方式，8 ウェイでは SON-5,6,7,8 方式が適切であり，性能向上が見込める．

しかしながら，これらの設定はプログラムやウェイ数によって変わる可能性がある．今回の設定は一例であり，プログラム，ウェイ数に応じて，切替パラメータを設定する必要がある．

5.　評　価

本章では，提案したリプレース動的切替方式の性能，ハードウェア量，動作周波数について評価する．

5.1　シミュレーションパラメータ設定

性能評価には，OChiMuS PE をシミュレートする実行駆動型シミュレータ *MUTHASI (MUltiTHreaded Architecture SImulator)*[3] を用いた．評価時のプロセッサパラメータを表 3 に示す．実スレッド数 (AT 数) は 2, 8 であり，各実スレッド数に応じたプロセッサ構成をとる．

評価には，LU 分解 (サイズ : 128 × 128), 行列乗算 (サイズ : 256 × 256), RADIX ソート (個数 : 16384) を用いた．LU 分解，RADIX ソートは SPLASH-2[5] より採用した．これらのプログラムは並列マクロによって並列化し，それぞれ 8 個の論理スレッドを生成する．また，プログラムの作成は MULiTh[4], binutils-2.13☆, gcc-3.2☆☆, newlib1.9.0 を用いた．各プログラムのスレッド分割方法，スレッド

☆ OChiMuS PE のスレッド制御命令を利用可能にしたもの．
☆☆ 最適化オプションは-O2 を設定した．

表 3　シミュレーション時のプロセッサパラメータ

PC (AT#)	2	8
Fetch Buffer Size	16	4
Dispatch Queue Size	32	8
Reorder Buffer Size	128	32
Normal Reservation Station Size	Simple ALU : 8　Complex ALU : 4	
LD/ST Reservation Station Size	8	
Branch History Table Size	1024 (gshare)	
Integer ALU	Simple ALU : 3,　Complex ALU : 2	
FPU	Simple ALU : 2 (delay 4cycle) Complex ALU : 1 (Mult 17 delay,　Div, 30 delay)	
Branch Unit	1	
Fetch Instructions	8	2
Decode Instructions	8	2
Dispatch Instructions	8	2
Retire Instructions	8	2
Finish Instructions	16	
Speculation Depth	4	

表 4　シミュレーション時のキャッシュメモリパラメータ

Capacity	L1-I-Cache	16KB
	L1-D-Cache	8KB, 32KB, 128KB
	L2-Cache	512KB
Way	L1-I-Cache	1
	L1-D-Cache	4
	L2-Cache	8
Line Size	L1-I-Cache	32B
	L1-D-Cache	32B
	L2-Cache	64B
Latency	L1-I-Cache	1 cycle
	L1-D-Cache	2 cycle
	L2-Cache	20 cycle

共有具合，メモリアクセスパターンは文献 2) を参照されたい．

次に，キャッシュメモリのパラメータを表 4 に示す．L1-D-キャッシュメモリとして，ウェイ数 4，ラインサイズ 32B を設定した．4 ウェイ程度の場合，リプレース方式として完全な LRU を実装しても，ハードウェア量は擬似 LRU と大きく変わらない．しかし，完全な LRU と疑似 LRU の性能誤差はわずかであり，ほぼ同一の性能であることが分かっている[2]．よって，本評価では，基準とするリプレース方式として，利用率が高く[6]，なおかつハードウェア量を抑えることのできる擬似 LRU を選択する．

また，本評価では，キャッシュ容量として 8KB，32KB，128KB を選択した．これらのキャッシュ容量は，本評価のプログラム規模を考慮し設定している．詳細は文献 2) を参照されたい．

このとき，L1-D-キャッシュメモリのリプレース方式として，擬似 LRU，LTN-1 方式 (LTN)，SDI-6 方式 (SDI)，SON-3 方式 (SON)，SDI-6+SON-3 方式 (SDI+SON) を実装し，プログラムを実行した．

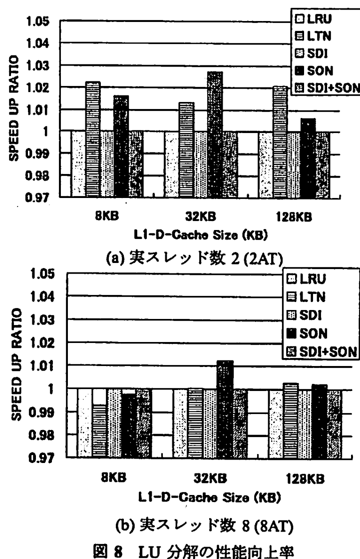
5.2 実行結果

擬似 LRU と LTN 方式および本論文で提案した動的切替方式の性能を比較した．各プログラムの性能向上率を図 8〜図 10 に示す．これらのグラフは，擬似 LRU のサイクル数を基準とし，提案した動的切替方式の性能向上率を示す．

LU 分解は，キャッシュ容量，実スレッド数に関わらず，高い性能向上率が得られず，どの方式も性能に大きな変化があらわれなかった．性能向上率として，実スレッド数 2 の 8KB，32KB，128KB，および実スレッド数 8 の 32KB で 1.01〜1.02 倍程度を示した．

行列乗算は実スレッド数 2，8KB において，LTN 方式使用による性能低下 (0.95 倍) が発生している．しかし，その性能低下を動的切替方式により改善しており，SDI+SON 方式は擬似 LRU よりも高い性能向上率 (1.01 倍) を示した．それ以外では，LTN 方式，各動的切替方式共に有効に動作しており，特に SDI+SON 方式は，LTN 方式単体で用いるよりも高い性能向上率を示した．例えば，実スレッド数 2，32KB における LTN 方式の 1.24 倍の性能向上率に対し，SDI+SON 方式は 1.31 倍の性能向上率を示した．また，128KB は，実スレッド数 2，8 共に性能向上率が非常に高く，最大で 1.48 倍を示した．

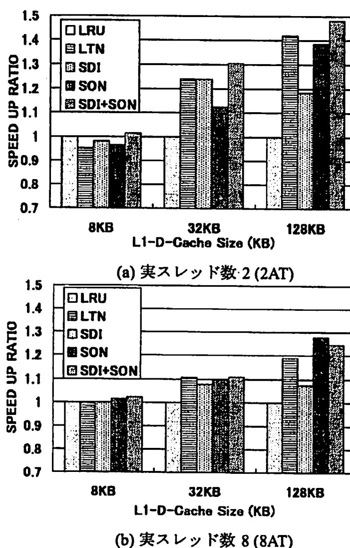
RADIX ソートは 8KB において，各方式を有効に活用し，実スレッド数 2 で 1.04 倍〜1.09 倍，実スレッド数 8 で 1.12 倍〜1.15 倍の性能向上率を示した．特に実スレッド数 8 では，行列乗算と同じく，



(a) 実スレッド数 2 (2AT)

(b) 実スレッド数 8 (8AT)

図 8　LU 分解の性能向上率



(a) 実スレッド数 2 (2AT)

(b) 実スレッド数 8 (8AT)

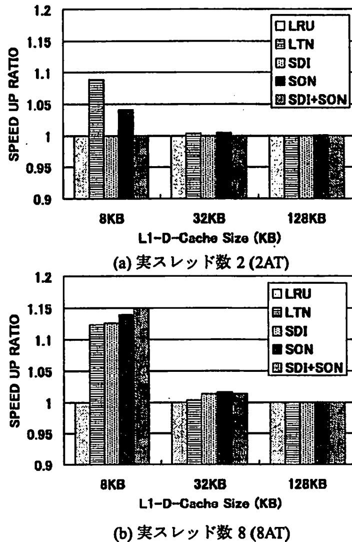
図 9　行列乗算の性能向上率

提案した動的切替方式が，LTN 方式単体よりも高い性能向上率を示した．逆に 32KB，128KB において，各方式は性能に影響を及ぼさず，高い性能向上率が得られなかった．

5.3 考　　察

提案した各動的切替方式は，SMT プロセッサの実スレッド数，キャッシュ容量の大小に関わらず，従来の擬似 LRU 以上の性能向上を示した．各方式は，行列乗算，RADIX ソートのように，破壊的干渉ミスが多発するプログラムの場合，高い性能向上が見込める．

また，実スレッド数 2，8KB の行列乗算，実スレッド数 8，8KB の LU 分解の結果から分かるように，LTN 方式によって発生した性能低下を動的切替方式は抑え，擬似 LRU と同様かそれ以上の性能を実現している．また，行列乗算，RADIX ソートでは，LTN 方式よりも動的切替方式の方が高い性能向上率を示している．つまり，提案したリプレース動的切替方式は，LTN 方式の性能低下を抑え，また LTN 方式よりも高い性能向上を実現することが達成できた．



(b) 実スレッド数 8 (8AT)

図 10 RADIX ソートの性能向上率

表 5 各動的切替方式のハードウェア量

スライス数	LRU	LTN	SDI	SON	SDI+SON
SMT	8366				
Cache	2937	2888	3248	3200	3270
Block RAM #	32	32	32	32	32
SMT + Cache	11303	11254	11614	11566	11636
増加率 (%)	0%	-0.44%	2.68%	2.27%	2.86%

5.4 ハードウェア量と動作周波数

提案した各動的切替方式の具体的なハードウェア量と動作周波数を見積るため，Verilog-2000 と Xilinx 社の ISE6.2.03i を用いて，各方式を実装した．実装したキャッシュメモリ構成は，キャッシュ容量 32KB，ウェイ数 4，ラインサイズ 32B，インデックス数 256 であり，リプレース方式は性能評価と同じく擬似 LRU (LRU)，LTN-1 方式 (LTN)，SDI-6 方式 (SDI)，SON-3 方式 (SON), SDI-6+SON-3 方式 (SDI+SON) を実装した．実装した結果を表 5 に示す．

SMT プロセッサのハードウェア量は現在著者らが設計・開発している FPGA 向け SMT プロセッサ[8] を参考にした．また，どの方式もデータ以外のタグ領域として，スライスを用いて実現する分散 RAM を使用したため，キャッシュメモリのハードウェアスライス数が多くなっている．キャッシュメモリのデータ部分は，各方式とも 32 個の Block RAM を用いた．

SDI 方式は，タグに追加した LTN およびカウンタが主な原因でハードウェア量が増加している．SON 方式も SDI 方式同様に，タグに追加した LTN が増加の主要因となっている．しかし，プロセッサを含めたチップ全体で考えると，SDI 方式のハードウェア増加率は 2.68 %，SON 方式の増加率は 2.27 %となり，擬似 LRU と比較しても大幅な増加量ではないことが分かる．SDI+SON 方式は両方式を同時に実現しているため，ハードウェア増加率が 2.86 %となる．

次に提案した動的切替方式の動作周波数を表 6 に示す．提案方式の実装対象が L1-D-キャッシュメモリであるため，動作周波数の低下は性能に大きな悪影響を及ぼす．しかしながら，擬似 LRU と比較し，SDI 方式の動作周波数低下率は 0.43 %，SON 方式は 0.56 %，SDI+SON 方式は 0.65 %となり，どれも低下率は 1 %未満である．つまり，提案した各動的切替方式の動作周波数の低下について問題はないことが分かる．

性能評価とハードウェア増加量の結果をみると，従来の擬似 LRU と比較し，SDI 方式は 2.68 %のハードウェア増加量に対し，最大 1.24 倍の性能向上を示した．SON 方式は 2.27 %のハードウェア増加量に対し，最大 1.39 倍，SDI+SON 方式は 2.86 %のハードウェア増加量に対し，最大 1.48 倍の性能向上を示した．

表 6 各動的切替方式の動作周波数

	LRU	LTN	SDI	SON	S+S
最長パス (ns)	19.769	19.467	19.77	20.033	20.268
動作周波数 (MHz)	61.389	62.125	61.125	61.047	60.992
低下率 (%)	0%	-1.19%	0.43%	0.56%	0.65%

*S+S : SDI+SON

6. 終わりに

本論文では，SMT プロセッサの性能低下の原因として，キャッシュラインのスレッド競合を取り上げた．スレッドの競合ミスを抑えるキャッシュリプレース方式として LTN 方式があるが，プログラムやキャッシュ容量によっては性能低下を引き起す．そこで，本論文ではプログラム実行中に疑似 LRU と LTN 方式の動的切替を行うことで，LTN 方式の性能低下の抑制，更なる性能向上を目指した．動的切替方式として，破壊的干渉ミス率を切替パラメータとする SDI 方式，セットごとにリプレース方式を切り替える SON 方式，およびそれらを組み合わせた SDI+SON を提案し，設計した．評価の結果，各動的切替方式は有効に動作し，LTN 方式で発生した性能低下を抑え，さらに疑似 LRU と比べ最大 1.48 倍の性能向上をもたらした．また，各動的切替方式を実装しハードウェアコストを見積り，わずかなハードウェア増加量で各動的切替方式を実現できることを示した．

今後の課題として，OChiMuS PE 以外の SMT アーキテクチャにおける各方式の適用，評価がある．また，SMT プロセッサ向けのリプレース方式として，LTN 方式の他に，スレッド間の共有データの有効活用を目指したリプレース方式がある[2]．今後は，そのリプレース方式を含めた 3 つの方式の動的切替の検討，評価を行いたい．

参 考 文 献

1) D. Tullsen, S. Eggers, and H. Levy : Simultaneous multithreading : Maximizing on-chip parallelism, In *ISCA-22*, pp. 392-403 (1995).
2) 小笠原嘉泰，佐藤未来子，笹田耕一，内倉要，並木美太郎，中條拓伯 : SMT プロセッサ向けキャッシュメモリリプレース方式，情報処理学会論文誌, Vol.47, No.SIG12(ACS15), pp.119-132 (2006).
3) 河原章二, 佐藤未来子, 並木美太郎, 中條拓伯 : システムソフトウェアとの協調を目指すオン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 2002, Vol.2002, No.18, pp.1-8 (2002).
4) 笹田耕一，佐藤未来子, 河原章二, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎 : 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現と評価, 情報処理学会論文誌, Vol.44, No.SIG11(ACS3), pp.215-225 (2003).
5) S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta : The SPLASH-2 Programs:Characterization and Methodological Consid-erations, In *ISCA-22*, pp.24-36 (1995).
6) J.L. Hennessy, and D.A. Patterson : Computer Architecture A Quantitative Approach 3rd Edition, Morgan Kaufmann Publishers (2002).
7) J. Handy : the Cache Memory book 2nd Edition, Academic Press (1998).
8) 加藤義人, 大和仁典, 小笠原嘉泰, 佐藤未来子, 笹田耕一, 内倉要，中條拓伯, 並木美太郎 : SMT プロセッサの FPGA への実装と評価, SACSIS 2005, pp.239-240 (2005).
9) 山崎真也, 本多弘樹, 弓場敏嗣 : マルチスレッドアーキテクチャにおけるデータキャッシュ構成方式の提案, 情処研報 (1998-HPC-93), Vol.1998, No.93, pp.79-84 (1998).
10) 内倉要, 笹田耕一, 佐藤未来子, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎 : SMT プロセッサにおけるスレッドスケジューラの開発，情報処理学会論文誌, Vol.46, No.SIG12(ACS11), pp.150-160 (2005).
11) Jack L. Lo, Luiz A. Barroso, Susan J. Eggers, Kourosh Gharachorloo, Henry M.Levy and Sujay S. Parekh : An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors, In *ISCA-25*, pp. 39-50 (1998).