情報処理学会論文誌　コンピューティングシステム　Vol. 2　No. 3　12–25 (Sep. 2009)

SMT プロセッサにおける L1/L2 キャッシュアクセス動的切替え方式

小笠原　嘉泰[†1,†2,*1]　三　輪　　忍[†1]
中　條　拓　伯[†1]

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，性能が低下するという問題がある．そこで本論文では，キャッシュアクセスとして L2-ダイレクトアクセスを可能にし，それを適切な条件で適用することで L1-キャッシュメモリを使用するスレッド数を調節し，スレッド間競合を抑える．L1/L2 キャッシュアクセスの動的切替え方式として，ヒット率を切替えパラメータとする方式とセットごとにキャッシュアクセスを切り替える方式を提案し，設計した．評価の結果，提案方式は通常のキャッシュアクセスと比較し，最大 1.106 倍，平均 1.022 倍の性能向上をもたらした．また，各提案方式を実装した結果，どちらの方式も，プロセッサとキャッシュメモリを含んだチップ全体で 3%未満とわずかなハードウェア増加量で実現できることを示した．

Dynamic Switch Strategies of Accessing L1/L2 Cache for an SMT Processor

Yoshiyasu Ogasawara,[†1,†2,*1] Shinobu Miwa[†1]
and Hironori Nakajo[†1]

An SMT processor aims to gain higher performance by sharing resources such as ALUs and cache memory among several threads. However, sharing cache memory causes thread conflict miss which degrades its performance. This paper proposes two dynamic switching strategies of accessing L1/L2 cache in order to improve performance. One uses the number of cache miss as switching, and the other switches accessing algorithm in each set. Dynamic switching strategies adjust number of thread in L1 Cache memory in order to reduce thread conflict miss. As a result, dynamic switching strategies show 1.022 times as high performance in average and 1.106 times in max as a conventional cache access. Furthermore, both dynamic switching strategies can be implemented with small additional hardware cost in less than 3%.

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，CMP（Chip Multi Processor），SMT（Simultaneous Multi-Threading）プロセッサ[1)]，その両方を用いたプロセッサが代表的である．たとえば，Intel の Core i7[2)] は CMP と SMT の両方を採用しており，1 コアが 2 スレッド同時実行可能な SMT プロセッサであり，それを 1 チップに複数コア実装している．

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 コア内で複数のスレッドを同時実行するため，コア内の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している．そのため，SMT プロセッサには，それらスレッド間共有資源の競合や枯渇が発生するという短所がある．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いる[3)–7)]．その解決策の 1 つとして，SMT プロセッサ内で扱うスレッド数を制御するスケジューラがある[6)]．このスケジューラは，キャッシュメモリの状況を監視し，キャッシュミスが多くなった場合，同時実行するスレッドを減らすことで，キャッシュラインのスレッド間競合を抑えている．その概要を**図 1** に示す．図 1 は 4 つのスレッドが同時実行可能な SMT を想定しており，キャッシュミスの増加により，1 つのスレッドの実行を停止させている．この方式によりキャッシュラインのスレッド間競合を抑えることができるが，一方で，仮にキャッシュメモリ以外の資源は多数のスレッド実行を許容できる状態であっても，並列実行するスレッド数が減ってしまうことになり，プロセッサ全体のスループットが低下してしまう．現に文献 6) では，スレッド数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ている．さらにこの問題

†1 東京農工大学
Tokyo University of Agriculture and Technology
†2 日本学術振興会特別研究員 PD
Research Fellow of the Japan Society for the Promotion of Science
*1 現在，任天堂株式会社
Presently with Nintendo Co., Ltd.

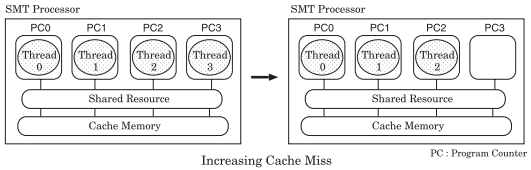

図 1　状況に応じスレッドを停止させる方法
Fig. 1　A thread halts by the condition of cache memory.

は，SMT プロセッサ内で並列実行するスレッド数が少ない場合，より顕著となる．そのため，キャッシュラインのスレッド間競合を抑える場合，スレッド停止範囲をキャッシュメモリだけに限定させれば効果的であり，性能向上が期待できる．

そこで，本論文では，ハードウェアによるスレッドの L1/L2 キャッシュアクセス動的切替え方式を提案する．キャッシュメモリの状況に応じ，通常の L1-キャッシュアクセスと L2-キャッシュへのダイレクトアクセスを動的に切り替え，スレッド停止をキャッシュメモリのみに限定する．そうすることで，プロセッサ内で実行しているスレッド数を変えないまま，キャッシュメモリのスレッド間競合を抑え性能向上を実現する．また，提案する方式を実装し，具体的なハードウェア量を見積もり，その実現可能性を検証する．

なお，本論文では，SMT 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*）と定義する．

次章では，研究背景を示す．3 章では，L1/L2 キャッシュアクセスの動的切替の概要について示す．4 章では，プログラム実行中に適切なタイミングでキャッシュアクセスを切り替える動的切替え方式を提案，設計，検討する．5 章では評価を，6 章では関連研究との比較を行う．

2. 研 究 背 景

本章では本論文で前提とする SMT プロセッサアーキテクチャを示す．また，事前評価を行いスレッド停止範囲の違いによる性能を比較する．

2.1　前提とする SMT プロセッサアーキテクチャ

本研究では SMT プロセッサとして *OChiMuS*（*On Chip Multi SMT*）*PE*[8] を使用する．OChiMuS PE の特徴の 1 つとして，スレッドに，**論理スレッド番号（LTN: Logical Thread Number）**という ID を付加し，スレッド管理の効率化を図っている．本論文のキャッシュメモリでは，この LTN を使用するが，OChiMuS PE と LTN でなくても，論理スレッドを一意に特定できるスレッド ID のようなものが存在すれば，他の SMT プロセッサにおいても対応が可能である．

また，SMT プロセッサアーキテクチャでは，資源の有効活用を目指し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．

2.2　スレッド停止範囲の違いによる性能比較

スレッドの停止範囲の違いによる性能比較として，プロセッサ全体を範囲にした場合と，キャッシュメモリのみに限定した場合を調査する．そのため，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*）[8] を用い，事前評価を行う．プロセッサの AT 数を 2，4 とし，ある 1 つの AT をまったく使わない場合（1AT Stop）と，ある AT のキャッシュアクセスをすべて L2-キャッシュにダイレクトアクセスさせる場合（Cache only Stop）の性能を評価する．2AT の 1AT Stop は，SMT をまったく活用していないことを意味する．プログラムは，スレッド間競合が比較的少ないプログラムとして LU 分解，比較的多いプログラムとして行列乗算を用いており，各プログラムは 8 個の論理スレッドに分割している．L1-キャッシュはウェイ数：4，ブロックサイズ：32 B，レイテンシ：2，キャッシュ容量：16 KB とし，L2-キャッシュはウェイ数：8，ブロックサイズ：64 B，レイテンシ：10，キャッシュ容量：512 KB とした．その他のプロセッサパラメータは 5.1 節に示す．

スレッド停止範囲の違いによる性能比較を**図 2** に示す．図 2 は，2AT，4AT において通常実行したときの性能を基準とする．1AT Stop，Cache only Stop どちらも極端な条件を設定しているため，通常実行と比較し全体的に性能が低下している．しかし，両者を比較すると，すべてにおいて 1AT Stop の性能が低い．特に，2AT やスレッド間競合ミスが少ない LU 分解で性能低下が著しい．一方，Cache Only Stop では性能低下が抑えられており，4AT の行列乗算に限っては，スレッド間競合を抑える利点により，通常実行よりも性能が高くなっている．このように，キャッシュメモリのスレッド間競合を抑える場合，スレッド停止をプロセッサ全体よりもキャッシュメモリに限定して行った方が，より効果的である．

一方，最近のプロセッサではキャッシュメモリ構成として，L3-キャッシュを用意し，L2-キャッシュをコアごとに保持させるようになってきている．そのため，L1-キャッシュと L2-

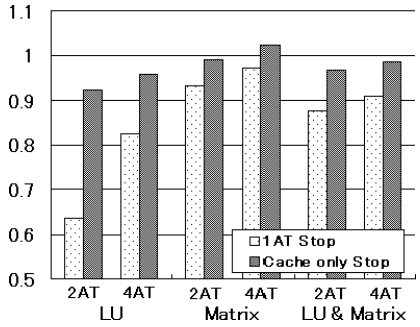

図 2　スレッド停止範囲の違いによる性能比較
Fig. 2　The performance comparison by area of halting a thread.

キャッシュのレイテンシ比が縮まりつつあり，たとえば，Core2 Quad QX9700 では，L1-レイテンシ：3，L2-レイテンシ：15（比率 1：5）なのに対し，Core i7 965 では，L1-レイテンシ：4，L2-レイテンシ：11（比率 1：2.75）である[2)]．そのため，L2-ダイレクトアクセスするコストが相対的に小さくなりつつある．図 2 の Cache only Stop は，ある 1 つの AT をプログラム開始から終了まですべて L2-ダイレクトアクセスさせたため，さすがに通常実行と比較して性能が低下する場合が多かった．本研究では，この L2-ダイレクトアクセスを，L1-キャッシュメモリの状況に応じ，適当なタイミングのみ使用することで性能向上を目指す．

3. L1/L2 キャッシュアクセスの動的切替え

L1/L2 キャッシュアクセス動的切替えの概要を**図 3** に示す．まず，プロセッサから L2-キャッシュメモリへのダイレクトアクセスを可能にする．そして，L1-キャッシュメモリの状況に応じて，通常の L1-キャッシュへのアクセスと，L2-ダイレクトアクセスを切り替える．図 3 では，4 つのスレッドが実行している状況で L1-キャッシュメモリのミス数が増加した場合を仮定し，スレッド 3 のみ L2-キャッシュメモリへダイレクトアクセスさせている．このように適切なタイミングで，スレッドを L2-キャッシュメモリへダイレクトアクセスさせることにより，プロセッサ内で実行しているスレッド数を変えないまま，L1-キャッシュメモリのみスレッド数を減らすことができ，スレッド間競合を抑えることができる．

L2-ダイレクトアクセスの実現方法を**図 4** に示す．プロセッサからの要求，またキャッシュメモリからのレスポンスとして，アドレス線，制御線，データ線などが必要であるが，図 4 ではそれを区別しておらず，まとめて 1 つの太線として記述している．

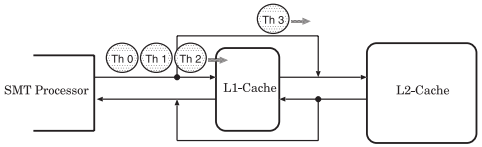

図 3　L1/L2 キャッシュアクセス動的切替えの概要
Fig. 3　The overview of dynamic switching of accessing L1/L2 cache.

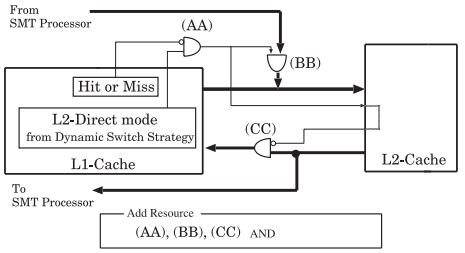

図 4　L2-ダイレクトアクセスの実現方法
Fig. 4　The realization method of L2-direct access.

次章で提案する動的切替え方式より，適切なタイミングで L2 ダイレクトモード（L2-Direct mode）として，0 または 1 が出力されてくる．その L2 ダイレクトモードと L1-キャッシュメモリの Hit/Miss 信号の反転を用い，AND を行う (AA)．さらに，その信号とプロセッサからの要求で AND (BB) を行い，(AA) からの出力信号が 1 の場合のみプロセッサからの要求を直接 L2-キャッシュメモリに送る．L2-キャッシュメモリでは，その信号をプロセッサからの要求とともに保持しておき，データなどのレスポンスを返すときにその信号を出力する．その信号の反転と L2-キャッシュメモリからのレスポンスを (CC) で AND を行い，L2-ダイレクトモードが 1 のときは，レスポンスを L1-キャッシュメモリに送らず，プロセッサにのみ送る．なお，L2-ダイレクトアクセス要求と L1-キャッシュメモリからの通常ミス要求が同時に発生した場合は，通常ミス要求を優先して処理する．

また，L2-キャッシュメモリにおいて，直接プロセッサからのメモリアクセス要求を処理する場合，L2-キャッシュメモリのラインサイズにあったビット列をアドレスの下位部分より抜き出し，L1-キャッシュメモリと同様に，それをセレクタの制御線とし，要求があったデータ部分のみロードもしくはストアすればよい．そのため，L2-キャッシュメモリにおい

てもラインから該当するデータを選択するセレクタが必要となる．しかし，新たに追加しなければならないセレクタは L2-キャッシュメモリにおいて 1 つのみでよいため，キャッシュラインサイズによるハードウェア増加量は大きくない．

以上をまとめると，プロセッサからのメモリアクセス要求は以下のように処理される．

(1)　L1-キャッシュメモリにアクセスする．

(2)　(2-1) ヒットした場合，L1-キャッシュメモリで通常のヒット処理を行う．
　　(2-2) ミスした場合，L2-ダイレクトモードを参照し，(3) の処理を行う．

(3)　(3-1) L2-ダイレクトモードが「0」の場合，L1-キャッシュメモリで通常のミス処理を行う．
　　(3-2) L2-ダイレクトモードが「1」の場合，プロセッサからの要求を L2-キャッシュメモリに伝播させ，L1-キャッシュメモリの通常キャッシュミス処理をキャンセルさせる．

以上のように，ヒットした要求はすべて L1-キャッシュメモリで処理し，ミスでなおかつ L2-ダイレクトモードが 1 の場合のみ，L2 へダイレクトアクセスさせる．

キャッシュアクセス処理をこのように実現することで，データのコヒーレンスが確保できる．L2-ダイレクトモードになっているスレッドでも，L1-キャッシュメモリにヒットした場合は L1-キャッシュメモリにおいてメモリアクセス要求を処理するため，L1-キャッシュメモリのデータが最新の値ではないという状況は発生しない．一方，L2-ダイレクトモードになっているスレッドが L1-キャッシュメモリをミスした場合，L2-キャッシュメモリにおいて，メモリアクセス要求を処理することになる．この場合，ストア命令により，データを直接 L2-キャッシュメモリに書き込むことになるが，その処理は L1-キャッシュメモリでミスしているため，最新のデータが L2-キャッシュメモリに存在しても問題はない．そのデータがスレッド間で共有されており，他のスレッドがそのデータを要求する場合，L2-キャッシュメモリに存在する最新データが L1-キャッシュメモリへ格納される．

4. 動的切替え方式

本章では，L1-キャッシュヒット率を切替えパラメータとする動的切替え方式として，**CHR 方式（switch by Cache Hit Ratio）**を，セットごとに L1/L2 キャッシュアクセスを切り替える動的切替え方式として，**SAC 方式（switch by occupied thread number of ways in Set And Cache hit ratio）**を提案し，設計する．

4.1 CHR 方式

4.1.1 CHR 方式の概要

L1/L2 キャッシュアクセス動的切替え方式として，ここでは L1-キャッシュメモリのヒット率を切替えパラメータとする CHR 方式を提案する．プログラム実行中に，プロセッサが保持するパフォーマンスモニタからのキャッシュヒット率をつねに参照する．参照した結果，キャッシュヒット率が一定値未満の場合，あるスレッドを L2-キャッシュにダイレクトアクセスさせる．また，その状態でキャッシュヒット率が一定値を超えた場合，キャッシュアクセスを通常の L1-アクセスに戻す．

このように，キャッシュヒット率をパラメータとしてとらえることで，適当な期間のみ L2-ダイレクトアクセスを採用でき，L1-キャッシュメモリのスレッド間競合を抑えることができる．また，スレッド間競合があまり発生せず，ヒット率が閾値まで低下しないプログラムの場合，L2-ダイレクトアクセスが発生することはなく，それによって引き起こされるスループット低下を防ぐことができる．また，1 度，L2-ダイレクトアクセスになったとしても，ヒット率が回復した場合，通常の L1-アクセスに戻す．そうすることで，L1-キャッシュメモリでキャッシュミスが特に発生している期間のみ，L2-ダイレクトアクセスを適用することができる．

CHR 方式では，動的切替えのための閾値として 2 種類用意する．まず，通常アクセス状態から L2-ダイレクトアクセス状態に切り替える閾値を **ND 値（from Normal to L2-Direct）**と定義する．また，L2-ダイレクトアクセスの状態において通常アクセスに切り替える閾値を **DN 値（from L2-Direct to Normal）**と定義する．たとえば，ND 値として 95%，DN 値として 97%を設定する場合，通常アクセス状態でヒット率が 95%未満になったら，あるスレッドのキャッシュアクセスを通常から L2-ダイレクトに切り替える．その状態で，ヒット率が 97%より大きくなった場合，再度 L2-ダイレクトアクセスを通常アクセスに切り替える．

L2-ダイレクトアクセスを発生させるとき，どのスレッドを対象とするか選定する必要がある．そのスレッド選定方法として，各スレッドのキャッシュミスの発生回数をもとに選ぶ方法，各スレッドのキャッシュメモリ使用率をもとに選ぶ方法，ランダムに選ぶ方法など様々なものが考えられる．しかし本論文では，動的切替え方式として，L1/L2 キャッシュアクセス動的切替えの切替え条件に焦点を当て性能を調査する．そのため，本論文では，スレッド選定方法として，実装が簡単であり，なおかつヒット率に悪影響を与えていることが容易に分かるキャッシュミスの発生回数で選ぶ方法を採用する．そのため，CHR 方式では各ス

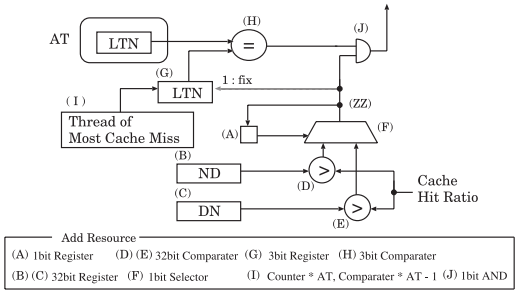

図 5　CHR 方式の実現方法
Fig. 5　The realization method of CHR strategy.

レッドごとにキャッシュミス回数をカウントし，最もキャッシュミスを起こしているスレッドを L2-キャッシュにダイレクトアクセスさせる．

以上を考慮した CHR 方式の切替えアルゴリズムを以下に示す．

(1)　初期設定として，すべてのスレッドのキャッシュアクセスを通常状態にし，プログラムを開始する．

(2)　プログラム実行中，つねにヒット率を参照し，設定した ND 値と比較する．

(3)　ヒット率が ND 値を下回っていた場合，最もキャッシュミスを発生させているスレッドを L2-ダイレクトアクセスにする．

(4)　1 度，L2-ダイレクトアクセスに切り替わっても，この状態でヒット率が DN 値を上回った場合，再度 L2-ダイレクトアクセスから通常アクセスに切り替える．

以上のように，CHR 方式は，L1-キャッシュメモリのヒット率を参照して L1/L2 キャッシュアクセスを切り替え，L2-ダイレクトアクセスを適当な期間のみ適用することで性能向上を目指す．

4.1.2　CHR 方式の実現方法

CHR 方式の切替えアルゴリズムの実現方法を**図 5** に示す．

まず，現在のキャッシュアクセス状態を保持する 1 bit のレジスタ (A) と，切替え閾値である ND 値と DN 値を保持する 32 bit のレジスタ (B)，(C) を用意する．初期値として，(A) には 0 を設定する．次に，プロセッサのパフォーマンスモニタから出力されるヒット率と，設定した ND 値と DN 値を (D)，(E) でそれぞれ比較する．そして，(F) のセレクトにより，現在の状態が通常状態（(A) の値が 0）の場合，(D) の判定結果を，ダイレクトアクセス状態（(A) の値が 1）の場合，(E) の判定結果を採用する．この信号 (ZZ) を (A) のレジスタの入力値とする．

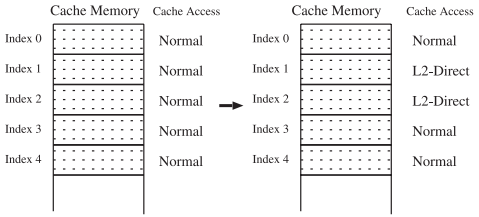

図 6　SAC 方式の概要
Fig. 6　The overview of SAC strategy.

一方，(G) のレジスタを用意し，(I) の回路から出力されてくる最もキャッシュミスを起こしている論理スレッド番号（LTN）を保持する．本論文では (I) の回路として，AT 数分のキャッシュミスカウンタと AT—1 数分の比較器が必要となるが，スレッド選定方法を異なる方法にする場合，この (I) の回路のみを変更すればよい．(I) からの出力値はプログラム実行中に変化するが，(ZZ) の信号が 1 のとき，つまりヒット率が ND 値未満になった場合，(G) の値を固定する．最後に，(G) とキャッシュミスを起こしたスレッドの持つ LTN を (H) で比較し，その結果と (ZZ) の信号の AND を (J) で行い，それを L2-ダイレクトモードとする．この図 5 からの出力値は，図 4 の L2-Direct mode となる．

CHR 方式を実現するためには，図 5 の (A)〜(J) の資源が必要となるが，どれも簡単な論理回路でキャッシュメモリ中に 1 つのみ追加すればよいので，ハードウェア増加量は大きくない．また，(G) のレジスタのビット幅はプロセッサ中で扱うスレッド数によって異なる．たとえば，8 つのスレッドを動作させる場合，(G) は 3 ビットのレジスタとなる．

4.2　SAC 方式

4.2.1　SAC 方式の概要

キャッシュメモリには，アクセスが集中するホットスポットが存在するため，キャッシュ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，セットごとにキャッシュアクセスを動的に切り替える SAC 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，キャッシュアクセスを L2-ダイレクトアクセスに切り替えている．このように特定のセットのみ，キャッシュアクセスを L2-ダイレクトとす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も対応することができる．

セットごとにキャッシュアクセスを切り替える手段として，キャッシュ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，その手段による実現には，測定用カウンタがインデックス分必要となり，ハードウェアが大幅に増加してしまうため現実的ではない．

そこで，以前我々が文献 4) のキャッシュリプレース動的切替え方式で提案した手法を採用する．それは，セットごとにキャッシュアクセスを切り替える手段として，あるセットに格納されているスレッド数に注目する．キャッシュタグに 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を確認する．そのスレッド数に応じて，キャッシュアクセスを切り替える．

このように，セット中に格納されているスレッド数を切替え閾値とし，それを本論文では，**OTN 値（Occupied Thread Number）**と定義する．たとえば，4 ウェイセットアソシアティブのキャッシュメモリ上で OTN 値として 4 を設定した場合，セットの各ウェイのデータが，すべて異なるスレッドだったときのみ，キャッシュアクセスとして L2-ダイレクトアクセスを選択する．

しかし，この手法は，キャッシュメモリのスレッド間競合について，適切なセット（空間軸方向）を発見することができるが，その採用期間（時間軸方向）の精度があまり適切ではないことが分かっている．文献 4) では切替え対象がキャッシュリプレース方式だったため，切替えの精度が多少荒く，切替えが多発しても問題は少なかった．ところが，本研究は L1/L2 キャッシュアクセスの切替えのため，不用意に L2-ダイレクトアクセスを多発させると，性能低下を招く恐れがある．

そこで，SAC 方式では，文献 4) の手法と CHR 方式を組み合わせ，両方の条件がともに有効になった場合のみ，スレッドを L2-キャッシュにダイレクトアクセスさせる．また，どちらかの条件が無効になった場合，L2-ダイレクトアクセスを通常アクセスへ切り替える．なお，L2-ダイレクトアクセスを発生させるスレッドの選定方法は，CHR 方式と同様にする．

以上を考慮した SAC 方式の切替えアルゴリズムを以下に示す．

(1)　初期設定として，すべてのスレッドのキャッシュアクセスを通常状態にし，プログラムを開始する．

(2)　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と CHR 方式の L2-ダイレクトモードを確認する．

(3)　確認した結果，OTN 値以上のスレッドがセット中に存在し，なおかつ CHR 方式の L2-ダイレクトモードが 1 の場合，最もキャッシュミスを発生させているスレッドを L2-キャッシュにダイレクトアクセスさせる．

(4)　1 度，L2-ダイレクトアクセスに切り替わっても，セット中のスレッド数が OTN 値より小さくなった場合，もしくは CHR 方式の L2-ダイレクトモードが 0 になった場合，再度 L2-ダイレクトアクセスから通常アクセスに切り替える．

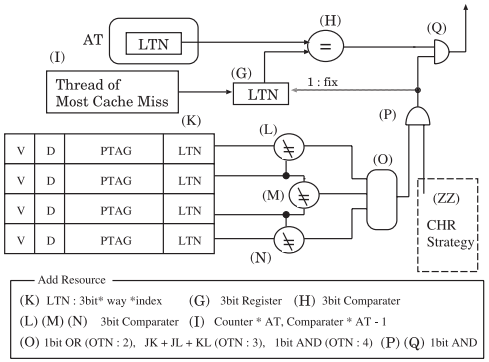

図 7　SAC 方式の実現方法
Fig. 7　The realization method of SAC strategy.

以上のように，SAC 方式は，切替え条件をより限定することで，適当な期間に適当な場所のみを L2-ダイレクトアクセスさせ，性能向上を目指す．

4.2.2　SAC 方式の実現方法

SAC 方式の切替えアルゴリズムの実現方法を**図 7** に示す．図 7 の出力値は，図 4 の L2-Direct mode となる．

まず，既存のタグに LTN として，(K) を追加する．そして，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．それにともなって，LTN からの出力ファンアウ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，SAC 方式では隣のウェイのみの LTN を比較することで，擬似的なスレッド数を確認する．ただし，すべての LTN 比較と，隣のウェイのみの LTN 比較で性能に大きな差が生じるようでは意味がない．そこで，2.2 節の事前評価と同じ環境とプログラムで，すべての LTN 比較と隣のウェイのみの LTN 比較の性能誤差を調査した．

調査結果を**図 8** に示す．このようにすべてのプログラムにおいて，性能誤差は＋− 0.1%未

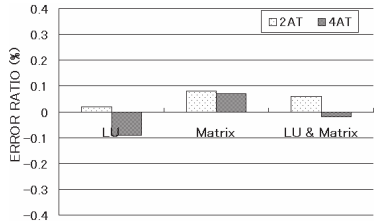

図 8　すべての LTN 比較と隣のウェイのみの LTN 比較の性能誤差
Fig. 8　The performance error ratio between comparing complete LTN and comparing pseudo LTN.

満であり，すべての LTN 比較による正確なスレッド数と隣のウェイのみの LTN 比較による疑似スレッド数の性能誤差はわずかであることが分かった．そのため，4 ウェイの場合は，図 7 のように (L)，(M)，(N) の 3 つの比較器が必要となる．そして，(L)，(M)，(N) からの入力をもとにした (O) を追加する．(O) からの出力は，そのセットにスレッドが OTN 値以上入っていたら 1，入っていなかったら 0 を出力する．よって，(O) は設定する OTN 値により異なり，OTN 値が 2 の場合は 1 bit OR 回路，OTN 値が 3 の場合は JK+JL+KL の組合せ回路，OTN 値が 4 の場合は 1 bit AND 回路となる．

さらに，CHR 方式の L2-ダイレクトモードを参照するため，(O) からの出力と図 5 の (ZZ) からの出力の AND を (P) で行う．そして，CHR 方式と同様に (G)，(H)，(I) を用いて，キャッシュミスを起こしたスレッドの LTN と最もキャッシュミスを起こしている LTN の値を比較する．最後に，その比較結果と (P) からの出力値で AND を行い (Q)，それを SAC 方式の L2-ダイレクトモードとする．

SAC 方式を実現するためには，図 7 の資源が必要となるが，この中で，(K) の追加 LTN に注意が必要である．たとえば，プロセッサで 8 個の論理スレッドを実行する場合，3 bit ×ウェイ×インデックス分の領域を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．

4.3　動的切替え方式の予備評価と検討

提案した 2 つの動的切替え方式について，本評価の前に予備評価[⋆1]を行い，適切なパラメータを検討する．予備評価に用いたプログラムは，スレッド分割方法，スレッド分割数，プログラム規模などすべて本評価と同様である．

⋆1 定量的な評価を行ったが，紙面の都合で，予備評価のまとめのみを掲載している．

4.3.1　CHR 方式の予備評価と検討

CHR 方式について検討する．まず，通常アクセスから L2-ダイレクトアクセスへの切替え閾値である ND 値の適切な値を求めるため，ND 値として，99%〜80%を設定し，予備評価を行った．その結果，95%〜90%の範囲がより高い性能をもたらした．スレッド間競合ミスが多いプログラムの場合，ヒット率が 90%以下になる場合が多く，一方，スレッド間競合ミスが少ないプログラムの場合，ヒット率が 95%以上になることが多かった．そのため，ND 値として，95%より大きい値を設定した場合，そもそもスレッド間競合が少ないプログラムまで L2-ダイレクトアクセスが頻発してしまい，同時に，スレッド間競合が多いプログラムでは，L2-ダイレクトアクセスする期間が長すぎてしまい，性能に悪影響を及ぼした．また，ND 値として 90%未満の値を設定した場合，スレッド間競合が少ないプログラムにおいて，切替えがまったく発生しなくなってしまった．また，スレッド間競合が多いプログラムでは，L2-ダイレクトアクセスする期間が短すぎる状況になり，大きな性能向上は得られなかった．

次に，L2-ダイレクトアクセスから通常アクセスへの切替え閾値となる DN 値の適切な値を求めるため，ND 値と DN 値の差として 0%〜5%を設定し，予備評価を行った．その結果，ND 値と DN 値の差は，2%〜3%程度が最適であることが分かった．これは，ND 値と DN 値の差が大きいと L2-ダイレクトアクセス期間が長くなりすぎてしまい，逆に差が小さいと短すぎてしまうことが原因である．

これらの結果より，ND 値は 95%〜90%，DN 値は ND 値との差が 2.5%程度が最適であり，性能向上が見込めると判断した．そのため，本評価では ND 値として 95%〜90%の中間値である 92.5%，DN 値として 95%を設定する．

4.3.2　SAC 方式の予備評価と検討

SAC 方式について検討する．SAC 方式では，セットごとにキャッシュアクセスを切り替えるための切替え閾値として OTN 値を用いる．その OTN 値の適切な値を求めるため，4 ウェイのキャッシュメモリ上で，OTN 値として 2〜4 を設定し予備評価を行った（OTN 値：1 も設定はできるが，すべてのセットで L2-ダイレクトアクセスを選択してしまうため意味がない）．その結果，3，4 のときの性能が良く，4 の性能向上率が最も高かった．2 では，多くのセットで切替えが発生してしまい，3，4 と比べると性能は低かった．

そのため，適切な OTN 値として，ウェイ数の半分を超える値が妥当と考え，4 ウェイならば 3 または 4，8 ウェイならば 5，6，7，8 が適切であり，性能向上が見込めると判断した．本評価では 4 ウェイの L1-キャッシュを使用するので，OTN 値として，4 を設定する．

5. 評　　価

本章では，提案した動的切替え方式の性能，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* [8] を用いた．評価時のプロセッサパラメータを**表 1** に示す．実スレッド数は 2，4 である．

評価には，LU 分解（サイズ：150 × 150），行列乗算（サイズ：200 × 200），RADIX ソート（個数：10,000）を用いた．LU 分解，RADIX ソートは SPLASH-2 [9] より採用した．これらのプログラムを並列化し，それぞれ単一プログラムから 8 個の論理スレッドを生成し，評価する．次に，異種プログラムとして，LU 分解と RADIX ソート，LU 分解と行列乗算，RADIX ソートと行列乗算を同時実行し，評価する．異種プログラムの評価では，単一プログラムから 4 個の論理スレッドを生成し，計 8 個のスレッドを実行する．なお，異種プログラムの同時実行では，一方のプログラムのみが実行しているという状況を極力避けるため，行列乗算の計算サイズを 150 × 150 にしている．また，プログラムの作成はスレッドライブラリ MULiTh [10]，binutils-2.13 ⋆1，gcc-3.2 ⋆2，newlib1.9.0 を用いた．各プログラムのスレッド分割方法，スレッド間のデータ共有具合，メモリアクセスパターンは文献 3) 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 2** に示す．本評価では，L1-D-キャッシュ容量として 8 KB，16 KB，32 KB を選択した．これらの容量は，本評価のプログラム規模を考慮し設定している[3]．また，キャッシュレイテンシとして，L1 を 2，L2 を 10 と設定した．このキャッシュレイテンシは本論文で提案する方式にとって，重要な値となる．そこで，文献 11)，12) や製品として登場している SMT プロセッサ[2] を参考にしてレイテンシとして妥当な値を設定している．

このとき，キャッシュアクセスとして，通常状態，CHR 方式，SAC 方式を選択し，プログラムを実行した．

5.2 実 行 結 果

本論文で提案した動的切替え方式の性能向上率を**図 9** に示す．このグラフは，キャッシュアクセスとして通常状態を基準とし，提案した動的切替え方式の性能向上率を示している．

本評価のプログラム特性として，LU 分解はスレッド間の競合が少ないプログラムであり，逆に行列乗算はスレッド間競合が多いプログラム，また RADIX ソートはその中間程度のプログラムである．そのため，LU 分解や RADIX ソート単体において提案方式の性能向上率が低い．しかし，性能低下に至っているプログラムは 8 KB，2AT の LU 分解のみである．逆に行列乗算が関わるプログラムでは，提案した動的切替え方式の性能が高くなっており，特に 8 KB では CHR 方式が，16 KB，32 KB では SAC 方式が主に有効となっている．

表 1　シミュレーション時のプロセッサパラメータ
Table 1　The processor configuration of simulation.

PC (AT#)	2	4
Fetch Buffer Size	16	8
Dispatch Queue Size	32	16
Reorder Buffer Size	128	64
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	

表 2　シミュレーション時のキャッシュメモリパラメータ
Table 2　The cache memory configuration of simulation.

Capacity	L1-I-Cache	16 KB
	L1-D-Cache	8 KB，16 KB，32 KB
	L2-Cache	512 KB
Way	L1-I-Cache	1
	L1-D-Cache	4
	L2-Cache	8
Line Size	L1-I-Cache	32 B
	L1-D-Cache	32 B
	L2-Cache	64 B
Latency	L1-I-Cache	1 cycle
	L1-D-Cache	2 cycle
	L2-Cache	10 cycle

⋆1 OChiMuS PE のスレッド制御命令を利用可能にしたもの．
⋆2 最適化オプションは-O2 を設定した．

表 3　単一プログラムの実行結果
Table 3　The result of all experiments of a program.

		LU			MATRIX			RADIX		
		cycle	SUR	L1D_H_R	cycle	SUR	L1D_H_R	cycle	SUR	L1D_H_R
L1-D cache size 8 KB	2AT Normal	24,723,542	1.000	95.57%	67,119,865	1.000	76.54%	24,107,053	1.000	99.11%
	2AT CHR	24,856,843	0.995	96.02%	63,089,180	1.064	84.69%	24,107,053	1.000	99.11%
	2AT SAC	24,809,831	0.997	96.56%	63,598,664	1.055	83.77%	24,107,053	1.000	99.11%
	4AT Normal	17,485,162	1.000	94.22%	70,070,801	1.000	53.32%	22,540,158	1.000	95.52%
	4AT CHR	17,368,486	1.007	95.26%	63,863,881	1.097	74.93%	22,468,111	1.003	96.09%
	4AT SAC	17,268,942	1.013	95.51%	64,829,496	1.081	72.82%	22,368,998	1.008	96.26%
L1-D cache size 16 KB	2AT Normal	23,582,499	1.000	98.14%	55,128,480	1.000	89.38%	24,057,835	1.000	99.16%
	2AT CHR	23,582,499	1.000	98.14%	53,828,952	1.024	93.65%	24,057,835	1.000	99.16%
	2AT SAC	23,582,499	1.000	98.14%	53,489,248	1.031	94.09%	24,057,835	1.000	99.16%
	4AT Normal	16,530,310	1.000	96.81%	54,409,762	1.000	83.32%	21,229,509	1.000	98.84%
	4AT CHR	16,530,310	1.000	96.81%	51,683,977	1.053	88.98%	21,229,509	1.000	98.84%
	4AT SAC	16,530,310	1.000	96.81%	52,039,810	1.046	87.72%	21,229,509	1.000	98.84%
L1-D cache size 32 KB	2AT Normal	22,462,504	1.000	99.21%	50,077,622	1.000	94.45%	24,008,386	1.000	99.24%
	2AT CHR	22,462,504	1.000	99.21%	49,276,434	1.016	95.65%	24,008,386	1.000	99.24%
	2AT SAC	22,462,504	1.000	99.21%	49,136,898	1.019	95.78%	24,008,386	1.000	99.24%
	4AT Normal	15,838,703	1.000	98.85%	48,818,699	1.000	93.08%	21,201,864	1.000	99.23%
	4AT CHR	15,838,703	1.000	98.85%	47,489,198	1.028	93.89%	21,201,864	1.000	99.23%
	4AT SAC	15,838,703	1.000	98.85%	47,183,838	1.035	93.98%	21,201,864	1.000	99.23%

*SUR: Speed Up Ratio, **L1D_H_R: L1 D-cache Hit Ratio

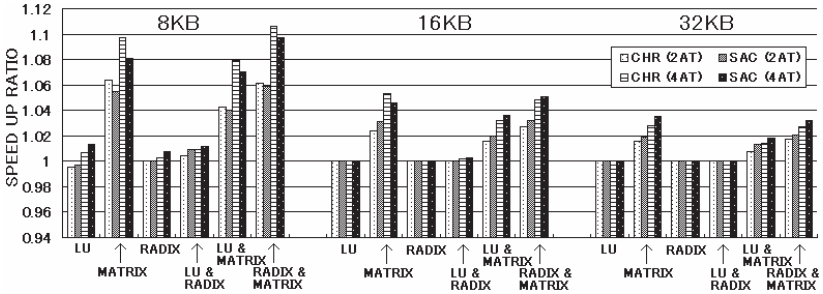

図 9　L1/L2 キャッシュアクセス動的切替え方式による性能向上率
Fig. 9　The speed up ratio by dynamic switch strategies of accessing L1/L2 cache.

2AT，4AT ともに性能向上を実現しているが，スレッド間競合が多く発生する 4AT の方が性能向上率が良い．また，キャッシュ容量を大きくすると，それにともなってスレッド間競合の発生が低くなるので各提案方式の性能向上率が緩くなっている．

性能向上率の平均値として，8 KB では 1.038 倍，16 KB では 1.018 倍，32 KB では 1.011 倍となり，2AT では 1.016 倍，4AT では 1.028 倍となり，全体の平均値は 1.022 倍となる．

5.3 考　　察

実行したプログラムのすべてのサイクル数，性能向上率，キャッシュヒット率を**表 3**，**表 4** に示す．本節では，実行結果をふまえて CHR 方式，SAC 方式について考察する．

まずは，実行プログラムの違いによる影響を示す．L2-ダイレクトアクセスはスレッド間競合を抑える働きがあるが，一方でスループットが低下してしまうという問題がある．そのため，プログラムの特性に応じて，L1/L2 キャッシュアクセスを適切に切り替える必要がある．スレッド間競合が多いプログラムの場合，両方式を用いることで適切なタイミングやセットで L1/L2 キャッシュアクセスを切り替えることが可能になり，性能向上を実現することができた．一方，スレッド間競合が少ないプログラムの場合，L2-ダイレクトアクセスをまったく採用しないことで，性能低下を防いでいる．そもそも，スレッド間競合が少ないプログラムでは SMT プロセッサでもヒット率が高く，L2-ダイレクトアクセスさせる必要性はない．このように，両方式はプログラムの特性に応じて動的切替えを柔軟に変化させ，性能向上を実現している．

ところが，評価結果を見ると，LU 分解のキャッシュ容量 8 KB，2AT のみ性能低下が発生している．LU 分解ではスレッド間競合が少なくキャッシュヒット率は全般的に高いため，2AT の SMT プロセッサ上においても，あるスレッドを L2-ダイレクトアクセスさせる必要

表 4　異種プログラムの実行結果
Table 4　The result of all experiments of programs.

		LU & RADIX			LU & MATRIX			RADIX & MATRIX		
		cycle	SUR	L1D_H_R	cycle	SUR	L1D_H_R	cycle	SUR	L1D_H_R
L1-D cache size 8 KB	2AT Normal	37,416,129	1.000	94.65%	49,359,770	1.000	80.42%	34,782,973	1.000	73.54%
	2AT CHR	37,269,860	1.004	95.01%	47,336,786	1.043	85.68%	32,786,483	1.061	83.08%
	2AT SAC	37,083,698	1.009	95.23%	47,469,863	1.040	84.33%	32,853,698	1.059	82.22%
	4AT Normal	26,457,060	1.000	93.94%	52,751,944	1.000	71.55%	37,288,137	1.000	51.48%
	4AT CHR	26,218,036	1.009	94.25%	48,888,967	1.079	79.23%	33,699,863	1.106	71.98%
	4AT SAC	26,136,909	1.012	94.57%	49,299,369	1.070	78.71%	33,998,648	1.097	70.72%
L1-D cache size 16 KB	2AT Normal	35,922,369	1.000	96.45%	43,644,826	1.000	89.92%	28,735,531	1.000	88.38%
	2AT CHR	35,922,369	1.000	96.45%	42,969,869	1.016	92.09%	27,990,360	1.027	93.56%
	2AT SAC	35,922,369	1.000	96.45%	42,769,893	1.020	93.30%	27,853,684	1.032	94.03%
	4AT Normal	25,066,694	1.000	95.21%	41,126,941	1.000	87.70%	28,230,709	1.000	83.96%
	4AT CHR	25,013,698	1.002	95.50%	39,869,806	1.032	89.53%	26,936,986	1.048	87.93%
	4AT SAC	24,987,860	1.003	95.76%	39,698,179	1.036	89.72%	26,869,863	1.051	88.01%
L1-D cache size 32 KB	2AT Normal	35,209,239	1.000	97.85%	40,081,754	1.000	94.90%	25,850,939	1.000	94.45%
	2AT CHR	35,209,239	1.000	97.85%	39,776,183	1.008	95.79%	25,428,981	1.017	95.46%
	2AT SAC	35,209,239	1.000	97.85%	39,563,689	1.013	95.93%	25,329,686	1.021	95.68%
	4AT Normal	24,211,676	1.000	97.13%	37,986,246	1.000	93.71%	25,294,047	1.000	92.88%
	4AT CHR	24,211,676	1.000	97.13%	37,469,860	1.014	94.70%	24,636,983	1.027	93.98%
	4AT SAC	24,211,676	1.000	97.13%	37,303,698	1.018	94.82%	24,518,618	1.032	94.08%

*SUR: Speed Up Ratio, **L1D_H_R: L1 D-cache Hit Ratio

性はない．しかし，キャッシュ容量が 8 KB と小さかったため，キャッシュヒット率が，キャッシュアクセスの切替え閾値である ND 値（本評価では，92.5%を設定）を一瞬下回ってしまった．その後，キャッシュヒット率が DN 値まで回復し，キャッシュアクセスは通常状態に戻ったが，本来なら適用すべきではないプログラムで L2-ダイレクトアクセスを適用してしまったため，L1-キャッシュヒット率の改善よりも，L2-ダイレクトアクセスによるスループット低下の影響が大きく反映されてしまい，総サイクル数が通常状態と比較し増加してしまった．しかし，このように切替え閾値を一瞬のみ下回るようなきわどい場合においても，どちらの方式も性能低下率は 0.5%以下であり，大きな性能低下とはならないことが分かった．

次に，CHR 方式と SAC 方式の特性について示し，比較する．CHR 方式は，ヒット率を切替えパラメータととらえることで，時間軸方向に対し，適切に L1/L2 キャッシュアクセスを切り替えることができる．一方，SAC 方式は，セットごとにキャッシュアクセスを切り替えることができ，さらに CHR 方式と組み合わせることで，空間軸方向と時間軸方向に対し，適切に L1/L2 キャッシュアクセスを切り替えることができる．そのため，SAC 方式の方が切替え精度が高く，性能向上率もすべてにおいて CHR 方式よりも高くなるはずである．ところが，8 KB では CHR 方式の性能の方が高い場合が多い．そこで，8 KB の該当プログラムにおける通常状態のキャッシュヒット率をみると，ヒット率が非常に低いことが分かる．このような場合，切替え条件の限定が有効に働かず，CHR 方式のように切替え条件を緩くすることで SAC 方式以上の性能向上率を実現できる．それならば，このような場合に限っては，いっそ 2.2 節の事前評価のようにある 1 つの AT を使用するスレッドをプログラム実行開始から終了まで L2-ダイレクトアクセスさせた方が性能が良くなるのではないかと考えた．しかし，このようなプログラムとキャッシュ容量でも，ある 1 つの AT をプログラム実行開始から終了まで L2-ダイレクトアクセスをしたときと比べると，CHR 方式の方が性能が高くなった．

ND 値，DN 値の設定について考察する．提案した動的切替え方式は，キャッシュアクセスの切替え閾値として ND 値，DN 値を用いており，これらの設定は性能に大きな影響を及ぼす．そこで，キャッシュ容量 16 KB の場合で，ND 値を 95%〜90%の間で 1%ずつ変化させたときの性能向上率を調査する．DN 値は，ND 値＋ 2.5%を設定し，本評価で用いた ND 値：92.5%の性能向上率も参考のため掲示する．ND 値を変化させたときの性能向上率を**図 10**，**図 11** に示す．図 10 はスレッド間競合ミスが多いプログラムとして行列乗算を，図 11 はスレッド間競合ミスが少ないプログラムとして LU 分解を用いている．これらの結果より，行列乗算の 2AT では 93%，4AT では 91%，LU 分解では 94%が ND 値として最

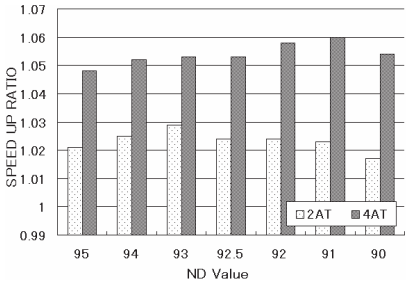

図 10　ND 値を変化させたときの性能向上率（スレッド間競合が多いプログラム）
Fig. 10　The performance comparison by ND value (program of high thread conflict misses).

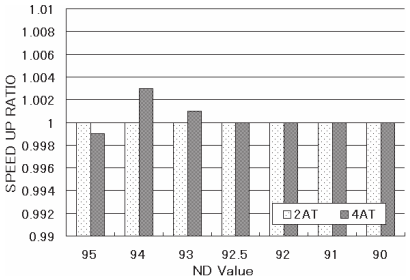

図 11　ND 値を変化させたときの性能向上率（スレッド間競合が少ないプログラム）
Fig. 11　The performance comparison by ND value (program of low thread conflict misses).

適値であり，性能向上率が最も高くなることが分かる．本評価では，予備評価により，すべてのプログラムの平均的な値として，ND 値：92.5%，DN 値：95.0%を設定している．本評価で用いた ND 値，DN 値は，各プログラムの最適値とは異なるものだが，最適値と比較しても大きな性能低下は示していない．また，図 10，図 11 より，ND 値として 95%〜90%の値を設定すれば，スレッド間競合の多いプログラムからスレッド間競合の少ないプログラムまで全般的に，性能に致命的な問題を与えないことが分かる．行列乗算などで L2-ダイレクトモードになった状態でキャッシュヒット率が DN 値まで回復しない場合も存在するが，そのような場合でも通常キャッシュアクセス状態と比較するとスレッド間競合の減少によりキャッシュヒット率が改善し，性能向上を実現している．

一方，本論文で提案した方式は ND 値，DN 値を静的に設定しているため，プログラム実行中に急激にメモリアクセスパターンが変化するプログラムには正確に対処できない恐れがある．また，L2-ダイレクトアクセスの対象としたスレッドが適切なものでなかったことが切替え後に判明する場合，本論文の提案方式は，それをリカバリする手段を持っていない．これらの問題を解決する手段として，プログラム実行状況に応じ，ND 値，DN 値を動的に設定し，これらの値をつねに最適な値に変化させていく方法がある．この ND 値，DN 値の動的設定は，ハードウェア単体で行う手法，システムソフトウェアと協調して行う手法など様々な手法が考えられるが，これら手法の実現は今後の課題とする．

表 5　各動的切替え方式のハードウェア量
Table 5　The hardware cost of dynamic switch strategies.

スライス数	Normal	CHR	SAC
SMT	8366		
Cache	2987	3056	3326
Block RAM #	32	32	32
SMT + Cache	11353	11422	11692
増加率 (%)	0	0.61	2.98

以上の考察より得られた知見をまとめる．提案した CHR 方式，SAC 方式はスレッド間競合ミスが多いプログラムにおいて性能向上を実現できる．一方，スレッド間競合ミスが少ないプログラムでも性能低下はほぼ発生しない．全体的に，切替え精度の高い SAC 方式の性能向上率が CHR 方式よりも高くなる．しかし，スレッド間競合ミスの多いプログラムを容量の小さいキャッシュメモリで動作させた場合のみ，CHR 方式が SAC 方式よりも有効となる．CHR 方式，SAC 方式ともに ND 値，DN 値の設定が性能に大きな影響を及ぼすが，ND 値を 95%〜90%，DN 値を ND 値 +2.5%に設定すれば，スレッド間競合ミスが多いプログラムからスレッド間競合ミスの少ないプログラムまで，全般的に性能向上を実現でき，性能低下を引き起こす場合でも大幅な性能低下とはならない．

5.4　ハードウェア量と動作周波数

提案した動的切替え方式の具体的なハードウェア量と動作周波数を見積もるため，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 32 KB，ウェイ数 4，ラインサイズ 32 B，インデックス数 256 であり，キャッシュアクセスは性能評価と同じく通常状態（Normal），CHR 方式（CHR），SAC 方式（SAC）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[13] を参考にした．また，どの方式もキャッシュメモリのタグ領域として，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スライス数が多くなっ

表 6　各動的切替え方式の動作周波数
Table 6　The frequency of dynamic switch strategies.

	Normal	CHR	SAC
最長パス (ns)	19.769	19.987	20.003
動作周波数 (MHz)	67.956	67.621	67.298
低下率 (%)	0	0.49	0.97

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．

CHR 方式は，切替え閾値を格納するレジスタおよび実スレッドごとのキャッシュミスカウンタが主な要因でハードウェア量が増加している．SAC 方式は，CHR 方式に加え，タグに追加した LTN が増加主要因となり，CHR 方式よりもハードウェア増加率が高くなっている．しかし，プロセッサを含めたチップ全体で考えると，CHR 方式のハードウェア増加率は 0.61%，SAC 方式のハードウェア増加率は 2.98%となり，通常状態と比較しても大幅な増加量ではないことが分かる．

次に提案した動的切替え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，通常状態と比較し，CHR 方式の動作周波数低下率は 0.49%，SAC 方式は 0.97% となり，どちらも低下率は 1%未満である．つまり，提案した動的切替え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．

各提案方式のハードウェア増加率・動作周波数低下率に対する性能向上率の結果をまとめる．5.2 節の性能向上率はシミュレーションによる結果だが，Xilinx 社のツールを使用し，実際に提案方式を実装して性能を評価した結果，性能向上率はシミュレーションの結果とほぼ同様となることが分かった．そのため，現在のキャッシュ容量として一般的な 32 KB において，CHR 方式は，0.61%のハードウェア増加率と 0.49%の動作周波数低下に対し，平均 1.01%，最大 2.8%の性能向上を実現できることが分かった．また，SAC 方式は，2.98%のハードウェア増加率と 0.97%の動作周波数低下に対し，平均 1.23%，最大 3.5%の性能向上を実現できることが分かった．なお，本評価ではキャッシュメモリのデータ部分を Block RAM を用いて実現しているため，ハードウェア増加率にこのキャッシュのデータ部分が考慮されていない．そのため，このキャッシュメモリのデータ部分をスライスを用いて実現する分散 RAM で実現しハードウェア増加量に加味した場合，ハードウェア増加量は本評価よりもさらに小さくなる．

6. 関連研究

関連研究として，Suh らによる Partitioning Decision 方式[5] がある．本論文ではキャッシュアクセスを動的に切り替えているが，Suh らはマルチスレッドプロセッサの L2-キャッシュにおいて，スレッドの実行状況に応じて扱うウェイ数を動的に変化させ，最適化している．これは，キャッシュ容量の大きい L2-キャッシュのあるセットにおいて，そこにアクセスする確率の高いスレッドに多くのウェイを割り当て，性能向上を目指している．この方式は，比較的空き容量の多い L2-キャッシュにおいて効果がある．しかし，キャッシュ容量が小さくなると，各スレッドは多くのセットにアクセスするため，動的にウェイを変化させる効果が薄くなる．たとえば，L2-キャッシュ容量が 1 MB の場合，14.2%の性能向上率を示しているが，256 KB の場合はわずか 0.1%の性能向上しか示していない[5]．よって，この方式は本論文で対象とする L1-D-キャッシュのように，キャッシュ容量が小さいキャッシュメモリについて効果は薄いと考える．

スレッドをハードウェアでコントロールし，キャッシュミスと削減する方法として文献 14), 15) がある．佐藤ら[14]，Sigmund ら[15] はスレッドに優先度をつけて，その優先度に基づいてスレッドのキャッシュアクセスをコントロールし，優先度の高いスレッドのキャッシュヒット率を改善している．これらの研究は，組み込みやメディア処理に特化しており，特定スレッドの処理時間の向上を目標としている．これに対し，我々の動的切替え方式はすべてのスレッドを対象とし，プログラム実行全体の性能向上を実現している．

また，キャッシュメモリのヒット率を改善する手法として SMT プロセッサ向けスケジューラ[6),7)] を活用する方法がある．プロセッサに流入するスレッド数を制限するスケジューラやスレッドの相性や親和性を監視するスケジューラについて提案し，ヒット率を改善している．しかし，システムソフトレベルのスケジューラでは，キャッシュメモリのみに限定してスレッド制御を行うことは難しい，本論文では，スケジューラよりもさらに細粒度のスレッド制御をキャッシュメモリに限定して行うことで性能向上を実現している．

7. 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間競合を取り上げた．スレッドの競合ミスを抑える方法として，L2-ダイレクトアクセスを適切なタイミングで適用し L1-キャッシュメモリのスレッド数を調節する L1/L2 キャッシュアクセス動的切替え方式を提案した．動的切替え方式として，キャッシュヒットを切替えパラ

メータとする CHR 方式，CHR 方式上でさらにセットごとにキャッシュアクセスを切り替える SAC 方式を提案し，設計した．評価の結果，各動的切替え方式は有効に動作し，通常のキャッシュアクセスと比較し，最大 1.106 倍，平均 1.022 倍の性能向上を示した．また，各動的切替え方式を実装しハードウェアコストを見積もった結果，どちらの方式も，プロセッサとキャッシュメモリを含んだチップ全体で 3%未満とわずかなハードウェア増加量で実現できることを示した．

今後の課題として，プログラム実行状況に応じた ND 値，DN 値の動的設定の実現がある．ND 値，DN 値をプログラム実行中に動的設定することで，CHR 方式，SAC 方式をあらゆるプログラムに適用可能にし，さらなる性能向上を目指す．また，本論文では L1/L2 キャッシュアクセスの動的切替えとして，切替え条件に焦点を当て，動的切替え方式を提案した．一方，L2-ダイレクトアクセスさせるスレッドの選定方法には，議論の余地が十分にある．本論文では，ヒット率に悪影響を与えていることが容易に分かるキャッシュミスの発生回数でスレッドを選定している．しかし，この方法以外にも，スレッドの各スループットの公平さで選ぶ方法，スレッドのキャッシュメモリの使用率で選ぶ方法など様々な選定方法が考えられる．以上のように，今後は L1/L2 キャッシュアクセスの動的切替えとして，ND 値と DN 値の動的設定およびスレッド選定方法に着目し，研究を進める．

謝辞　本研究の一部は，日本学術振興会科学研究費補助金（No.19・8474）および文部科学省共生情報工学推進経費による．

参 考 文 献

1) Tullsen, D., Eggers, S. and Levy, H.: Simultaneous multithreading: Maximizing on-chip parallelism, *Proc. 22nd Annual International Symposium on Computer Architecture* (*ISCA-22*), pp.392–403 (1995).
2) Intel Core i7 Processor. http://www.intel.com/products/processor/corei7/index.htm
3) 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：コンピューティングシステム，Vol.47, No.SIG12(ACS15), pp.119–132 (2006).
4) 小笠原嘉泰，佐藤未来子，並木美太郎，中條拓伯：SMT プロセッサにおけるキャッシュリプレース動的切替え方式，情報処理学会論文誌：コンピューティングシステム，Vol.48, No.SIG13(ACS19), pp.70–83 (2007).
5) Suh, G.E., Rudolph, L. and Devadas, S.: Dynamic Partitioning of Shared Cache Memory, *The Journal of Supercomputing Architecture*, pp.7–26 (2004).
6) 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：コンピューティングシステム，Vol.46, No.SIG12(ACS11), pp.150–160 (2005).
7) Snavely, A., et al.: Symbiotic jobscheduling for a simultaneous multithreading processor, *ASPLOS-2000*, pp.234–244 (2000).
8) 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).
9) Woo, S.C., Ohara, M., Torrie, E., Singh, J.P. and Gupta, A.: The SPLASH-2 Programs: Characterization and Methodological Consid-erations, *ISCA-22*, pp.24–36 (1995).
10) 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：コンピューティングシステム，Vol.44, No.SIG11(ACS3), pp.215–225 (2003).
11) Hennessy, J.L. and Patterson, D.A.: *Computer Architecture A Quantitative Approach*, 3rd Edition, Morgan Kaufmann Publishers (2002).
12) Handy, J.: *The Cache Memory book*, 2nd Edition, Academic Press (1998).
13) 加藤義人ほか：SMT プロセッサの FPGA への実装と評価，先進的計算基盤システムシンポジウム SACSIS 2005, pp.239–240 (2005).
14) 佐藤純一，内山真郷，伊藤　務，山崎信行，安西祐一郎：リアルタイム処理用マルチスレッディングプロセッサの優先度に基づくキャッシュサブシステム，情報処理学会研究報告 2001-ARC-143，Vol.2001, pp.37–42 (2001).
15) Sigmund, U. and Ungerer, T.: Memory Hierarchy Studies of Multimedia — enhanced Simultaneous Multithreaded Processors for MPEG-2 Video Decompression, *MTEAC-2000*, pp.1–9 (2000).

(平成 21 年 1 月 27 日受付)
(平成 21 年 4 月 28 日採録)

小笠原嘉泰（正会員）

1982 年生まれ．2003 年育英（現，サレジオ）工業高等専門学校情報工学科卒業．2005 年東京農工大学工学部情報コミュニケーション工学科卒業．2006 年同大学大学院工学教育部情報コミュニケーション工学専攻博士前期課程修了．2007 年日本学術振興会特別研究員 DC2．2008 年東京農工大学大学院工学府電子情報工学専攻博士後期課程修了．同年 4 月，日本学術振興会特別研究員 PD．2009 年 4 月，任天堂株式会社入社．博士（工学）．マルチスレッドプロセッサ，キャッシュメモリ，FPGA，再構成技術に興味を持つ．

三輪　　忍（正会員）

1977 年生まれ．2000 年京都大学工学部情報学科卒業．2002 年同大学大学院情報学研究科通信情報システム専攻修士課程修了．2005 年同大学院情報学研究科通信情報システム専攻博士課程学習認定退学．同年 4 月京都大学法学研究科助手．2008 年 1 月東京農工大学大学院工学府特任助教．博士（情報学）．計算機アーキテクチャ，ニューラルネットの研究に従事．電子情報通信学会，人工知能学会各会員．

中條　拓伯（正会員）

1961 年生まれ．1985 年神戸大学工学部電気工学科卒業．1987 年同大学大学院工学研究科修了．1989 年同大学工学部助手の後，1998 年より 1 年間 Illinois 大学 Urbana-Champaign 校 Center for Supercomputing Research and Development（CSRD）にて Visiting Research Assistant Professor を経て，現在，東京農工大学大学院共生科学技術研究院准教授．プロセッサアーキテクチャ，並列処理，クラスタコンピューティング，高速ネットワークインタフェースに関する研究に従事．電子情報通信学会，IEEE CS 各会員．博士（工学）．