計算機アーキテクチャ 152－1
（２００３．３．１０）

スレッド管理機構を用いたオンチップマルチスレッドのためのキャッシュメモリシステム

大和仁典[†]　河原章二[†]　加藤義人[†]
笹田耕一[†]　佐藤未来子[†]
並木美太郎[†]　中條拓伯[†]

現在，命令レベル並列性 (ILP) に加えスレッドレベル並列性 (TLP) に着目したチップマルチプロセッサ (CMP) や SMT (Simultaneous Multithreading) などのオンチップマルチスレッドアーキテクチャが注目されている．しかし，複数のスレッドが同時に実行されていることから，複数のスレッド間でキャッシュエントリの競合を起こし，キャッシュ効率が低下することが問題となる．本論文では SMT を対象とし，アーキテクチャと OS で管理する論理スレッド番号 (LTN) を利用してスレッド間のキャッシュエントリの競合を抑制したキャッシュ方式である LTN リプレース方式を提案し，プロセッサシミュレータ MUTHASI を用いて行列演算による評価を行ったところ，データ領域が大きいほど LRU によってリプレースを行う方式より性能向上が大きいことが分かった．また，LTN リプレース方式を実現するためのハードウェアリソースの追加は少ないため，チップ面積の増加を最小限に押えながらヒット率の向上を見込める．

Cache Memory System for On-chip Multi-threaded Processor with a Thread Management Mechanism

Masanori Yamato ,[†] Shoji Kawahara ,[†] Norito Kato ,[†]
Koichi Sasada ,[†] Mikiko Sato ,[†] Mitaro Namiki [†]
and Hironori Nakajo [†]

Currently, multi-threaded architectures such as chip multi-processor and SMT (Simultaneous Multithreading), which exploit TLP in addition to ILP, are in a hot topic. In such architecture, however, simultaneously executed threads cause conflicts in cache entries among threads, thus it may degrade efficiency of cache. In this paper, we propose an LTN based replacement strategy that utilizes thread number: Logical Thread Number (LTN) managed by OS in order to control a thread to be replaced in cache entry. We have evaluated our proposed strategy by simulator MUTHASI. The evaluation shows that the larger data size increases the more speed up is gained by LTN replacement strategy against LRU. Since it is not necessary to add so much hardware resources for the LTN replacement strategy, it is expected that the LTN replacement strategy brings high hit ratio without expansion of chip area.

1. はじめに

現在、スーパスカラプロセッサにおける命令レベル並列性 (Instruction Level Parallelism:ILP) 向上の限界から、ILP とともにスレッドレベル並列性 (TLP) に重点がおかれた研究，開発が行われている．オンチップマルチスレッドプロセッサは TLP の向上を目的としたプロセッサであり，１つ，または複数のプログラムから抽出された複数のスレッドを実行することができる．

オンチップマルチスレッドプロセッサとして，チップマルチプロセッサ (CMP)[1] や Simultaneous Multithreading(SMT)[2)3)5)6)] があげられる．CMP は 1 チップ内に複数のプロセッシングエレメント (PE) が実装されており，それぞれの PE で異なるスレッドを実行することができる．通常の対称型マルチプロセッサ (SMP:Symmetric Multi-Processor) と比較すると，チップ内結合により高速に通信できる点や共有レジスタを持つことができる点で優位である．一方，SMT は実行ユニットを共有し，複数のプログラムカウンタ (PC) を用いることにより複数のスレッドから命令をフェッチし，その後デコードを行い，共有している実行ユニットに対して命令をディスパッチする．これらの工夫により、スーパスカラプロセッサより実行ユニットを有効利用できる．

しかし, オンチップマルチスレッドプロセッサは並列実行されている複数のスレッドが同一キャッシュエントリにアクセスした場合，互いにリプレースを行い，キャッシュ効率が低下してしまう．これを解決するためにキャッシュ容量を増加させると，それに伴うチップ面積の増加やアクセス速度の低下が問題となる．

そこで，本論文では具体的な SMT アーキテクチャを対象とし，キャッシュ容量を増すことなく高いヒット率を実現できるキャッシュメモリ方式を提案する．そして，オンチップマルチスレッドアーキテクチャを対象とした命令レベルシミュレータ MUTHASI[3] を用いて提案したキャッシュ方式を評価し，今後の SMT アーキテクチャにおけるキャッシュメモリの構成について考察する．

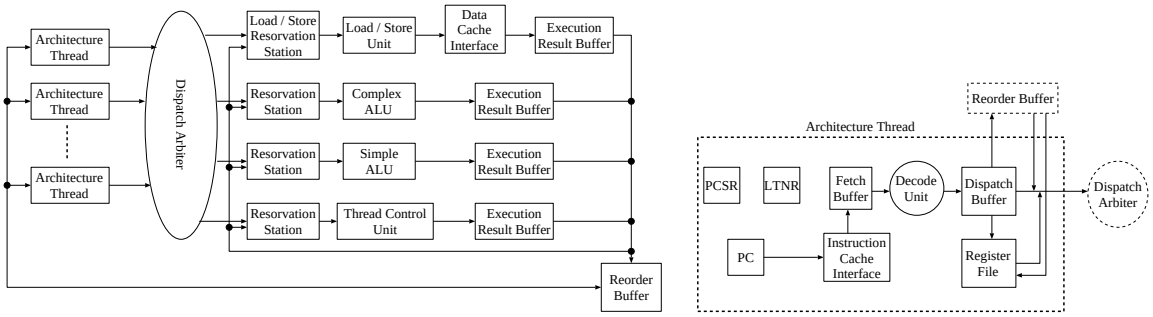
2. 前提となる SMT アーキテクチャ

2.1 スレッドの概念

ここでは，スレッドを一連の命令流れであるものとする．また，プロセスは 1 つ以上のスレッドの集まりとし，同一プロセス内のスレッドはメモリ空間を共有するものとする．

我々はスレッド管理の仮想化をサポートしたアーキ

† 東京農工大学工学部情報コミュニケーション工学科
Department of Computer, Information and Communication Sciences, Tokyo University of Agriculture and Technology



(a) 対象アーキテクチャの要素プロセッサ全体図　　(b) アーキテクチャスレッドに固有な部分

図 1　対象 SMT-PE の構成

テクチャを提案している．つまり，システムソフトウェアは PC と論理スレッドの割当て状態や空き PC を管理する必要がない．そのために，さらにスレッドを論理スレッドとアーキテクチャスレッドに分け，OS と連携して管理する方式を提案する[4]．論理スレッドはシステムソフトウェアが管理するスレッドであり，アーキテクチャスレッドは PC に割り当てられ，実行中のスレッドを指す．

2.2 プロセッサの構成

図 1 に対象 SMT プロセッサ (以下，Processing Element(PE)) の構成を示す．本アーキテクチャではアーキテクチャスレッドごとに以下のようなハードウェアリソースを持つ．

- プログラムカウンタ (PC)
- 実行状態レジスタ (PCSR)
- 論理スレッド番号レジスタ (LTNR)
- 汎用レジスタファイル (RegisterFile)

プログラムカウンタは従来のプロセッサと同様にフェッチする命令のメモリ番地を格納する．実行状態レジスタはスレッドの実行状態を保持している．論理スレッド番号レジスタ (LTNR) はアーキテクチャスレッドの論理スレッド番号 (LTN) を保持しており，スレッド制御命令の際に参照される．汎用レジスタは従来のプロセッサの汎用レジスタと同じものである．以上のものはマルチスレッド化するにあたって必ずアーキテクチャスレッドごとに独立させなければならない．逆に実行ユニットは全アーキテクチャスレッドで共有する．また，キャッシュについては命令キャッシュは PC ごと，データキャッシュは PE 全体で共有する．

2.3 スレッド制御

スレッド制御命令は PC ではなく論理スレッド番号 (LTN) を指定する．たとえば，スレッド割当ての場合はハードウェアが論理スレッドの割当てが行われていない PC を検索し，存在する場合はその PC にスレッドを割り当てる．全ての PC に論理スレッドが割り当てられている場合は失敗となる．またその他のスレッド制御命令の場合は，指定された LTN の割り当てられている PC を検索し，存在する場合はその PC に対してスレッド制御命令を発行する．存在しない場合は失敗となる．

このようにハードウェア側で LTN を管理することにより，システムソフトウェアで PC にスレッドが割り当てられているかどうか，また，どの PC にどのスレッドが割り当てられているかと言った対応関係を管理する必要がない．逆にこのような機構がない場合，メモリ中にどの論理スレッドがどの PC に割り当てられているかといった対応関係を保持する必要があり，スレッド管理の度にこの情報を参照する必要があるため，スレッド管理のオーバーヘッドが大きくなる．以上のことから，LTN を用いたスレッド管理はオンチップマルチスレッドプロセッサ，特に細粒度のアプリケーションにおいて性能向上が見込めると考えられる．

3. SMT アーキテクチャにおけるキャッシュ

3.1 従来キャッシュの問題点

データキャッシュを PE 内のスレッドで共有することによる問題として，PE 上で複数のスレッドが動作している場合，異なるメモリアドレスにてキャッシュ上の同一エントリをアクセスする状態が頻発する．これにより，シングルスレッドプロセッサに比べ，競合ミスによるキャッシュミス率が増え，データアクセスにおけるキャッシュのミスペナルティが大きくなる．

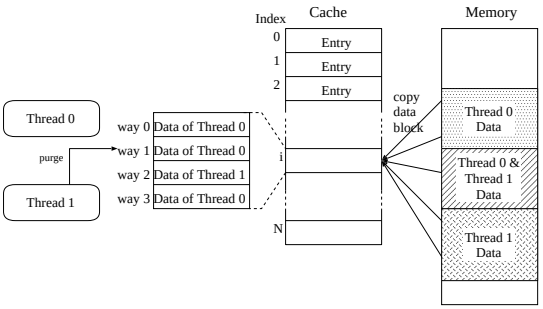


図 2　マルチスレッドにおける他スレッドからのリプレース

図 2 に 4 ウェイセットアソシアティブ方式における例を示す．図 2 では，エントリ i において，あらかじめスレッド 0 がウェイ 0，1，3 のラインにデータブロックをコピーし，スレッド 1 がウェイ 2 のラインにデータブロックをコピーしていたとする．ここでスレッド 1 がエントリ i に対してアクセスを行いキャッシュミスした場合，メモリから新たにデータブロックをコピーするためにスレッド 1 があらかじめコピーしたデータ (図 2 ではブロック番号 1 のラインのデータブロック) をリプレースしてしまう可能性がある．このようにしてシングルスレッドプロセッサでは起こらないようなキャッシュリプレースが発生するため，マルチスレッドプロセッサではキャッシュ効率が低下する．

これを解決するためにキャッシュ容量や連想度を増加させると，チップ面積やアクセスレイテンシの増加が起こり，リソース追加に見合うパフォーマンスは期

待できない．そこで，リソース追加を最小限に押えつつ，エントリの競合を押える方法を次節にて検討する．

3.2 LTN リプレース方式

他のスレッドからのリプレースを制限するために，図 3(a) のように PC ごとにアクセスできるウェイを限定する方法も考えられる．しかし，これは実質的には PC ごとにキャッシュを持たせているのと等価である．よって異なるスレッドからアクセスされるウェイへ同一データブロックをコピーすることで，キャッシュ効率の低下がおこる．また，同一データブロックのコピーを異なるウェイでもつので，ウェイ間の同期によるオーバーヘッドの増加が問題となる．

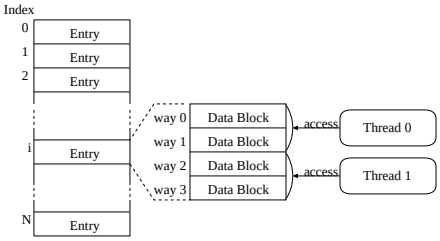


図 3 PC ごとにアクセスできるウェイを限定する方式

これを解決する方式として，どの PC もすべてのウェイにアクセスでき，リプレースについては PC ごとに決められたウェイのラインのみをリプレースするようにする方式が考えられる．これにより，他の PC がメモリからコピーした場合でもアクセスできるのでキャッシュミスを起こさない．よって，先に述べたキャッシュ効率の低下や同期によるレイテンシの増加は起こらない．しかし，この方式ではスレッド切り替えなどの要因で，スレッドが現在割り当てられている PC と異なるスレッドに割り当てられた場合，他のスレッドが先にメモリからコピーしたラインを追い出すことによりキャッシュヒット率が低下する．

そこで，我々は LTN を用いたキャッシュリプレース方式を提案する．リプレース対象となるウェイを決定する際に LTN を用いることで，スレッドに割り当てられる PC に依存することなくリプレースするウェイは一定であるため，スレッド切替えの影響を受けない．以下，この方式を LTN リプレース方式と呼ぶ．また，LTN リプレース方式において 1LTN あたりのリプレース可能ウェイ数を N としたとき LTN リプレース N セット方式と呼ぶ．例えば 4way セットアソシアティブ方式において LTN モジュロを 1 ビット用い，1LTN あたりのリプレース可能ウェイ数が 2 の場合は LTN リプレース 2 セット方式と呼ぶ．また，2 ウェイセットアソシアティブ方式は自明なので単に LTN リプレース方式と呼ぶ．ただし，リプレース可能なウェイが複数存在する場合は LRU にてリプレースを行うウェイを決定する．

4. 命令レベルシミュレータ MUTHASI による評価

我々はマルチスレッドアーキテクチャシミュレータ MUTHASI (MUlti-THread Architecture SImulator)[3] を用いて，LRU によりリプレースを行うキャッシュ方式，LTN リプレース方式ともに実装し，評価を行った．

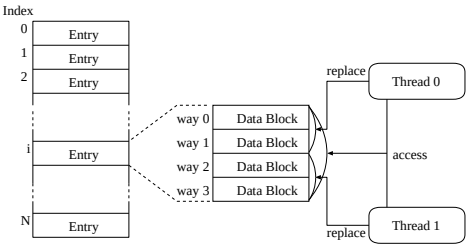


図 4 LTN リプレース方式の概念図

4.1 シミュレーション環境

MUTHASI は命令レベルシミュレータであり，各パイプラインステージの動作を正確にシミュレーションでき，PC 数の変更などを柔軟に行えるようになっている．また，MUTHASI は MIPS R4000 32 ビット整数命令セットと互換性があり，これに対し，スレッド制御命令を追加したものである．キャッシュについても，実際のキャッシュやメモリの遅延などを考慮に入れた構造になっており，キャッシュウェイの構成も実際のキャッシュと同等のものを実装している．

また、スレッド制御命令に対応したスレッドライブラリとして独自に開発したスレッドライブラリ MULiTh を用いた．MULiTh は POSIX スレッド準拠のライブラリとして使うことができ，ここで取り上げる評価プログラムも POSIX スレッド対応のものを使用している．

4.2 行列積による評価

4.2.1 行列積の並列化手法

評価プログラムとして行列積を 8 個のスレッドに処理を分割して求めるプログラムを実行した．並列化の手法として

$$\begin{aligned} C &= A \times B \\ \{c_1 c_2 ... c_n\} &= A \times \{b_1 b_2 ... b_n\}\Gamma \\ c_i &= A \times \ b_i \qquad (i = 1, 2...n) \end{aligned} \tag{1}$$

であることから，スレッド j では

$$\frac{n}{8}(j-1) < i < \frac{n}{8}j \qquad (j = 1, 2...8)$$

の範囲を計算するように処理を分割し並列実行を行う．行列 A がスレッド間で共有するデータであり，データ B はスレッドごとに独立したデータである．よって行列 B のデータ領域がエントリの競合の主な要因となるアクセスになる．

評価する行列サイズは 16×16，32×32，64×64，128×128，256×256 である．また，行列の 1 要素は INT 型であり，当環境では 4Byte である．行列のサイズを変更することによりワーキングセットの変化を考慮に入れ評価を行う．また，キャッシュのパラメータについては表 1 に示す。4 ウェイセットアソシアティブ方式については LTN リプレース 2 セット方式，1 セット方式ともに評価しその比較を行う．

4.2.2 2 ウェイセットアソシアティブ方式における評価結果

評価結果について行列サイズ 16×16，32×32，64×64 は同様の傾向だったため 64×64 の結果を示す．また，行列サイズ 128×128 についてはキャッシュ容量が 16KB，32KB については行列サイズ 256×256 と同様，キャッシュ容量 64KB，128KB については行列サイズ 64×64 と同様の傾向であったため，省略する．4way セットアソシアティブ方式についても同様

である．

まず，64 × 64 の場合の実行結果を図 5 に示す．グラフは、横軸がキャッシュ容量及び各キャッシュ容量における PC 数である．また，棒グラフは実行サイクル数，折れ線グラフは LRU によるリプレース方式に対する LTN リプレース方式の実行速度の向上率を示したものである．これらは以下の全てのグラフで共通である．

この結果より，LTN リプレース方式は LRU 方式に対し性能低下しているいることがわかる．しかし，64 × 64 の PC 数 1，キャッシュ容量 16KB において 5%程度性能低下しているが，その他のパラメータにおいては 1%の性能低下にとどまる．

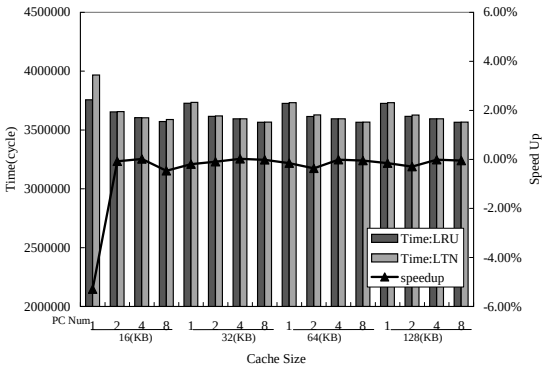


図 5　64 × 64 正方行列の積による評価結果

次に，行列サイズ 256 × 256 における評価結果を図 6 に示す．全体的にみると LTN リプレース方式が有効であることがわかる．キャッシュ容量 16KB における PC 数 8 の場合とキャッシュ容量 32KB の PC 数 2，4，8 の場合を除いて LTN リプレース方式によって性能向上していることがわかる．特にキャッシュ容量 64KB，128 において大きくパフォーマンスが向上していることがわかる．

アクセスするデータ量を考えると行列 1 つにつき 256KB(=4$byte$ × 256 × 256) のデータアクセスがあり，キャッシュ容量 128KB においてさえキャッシュサイズの 2 倍の大きさであり，行列サイズ 64 × 64 以下のときのように収まりきらない容量である．

つまり，PC 数増加によるキャッシュラインのリプレース要因の増加や，ラインのリプレースの方法による差が出ていることになる．注目すべき結果としてはキャッシュ容量 64KB における 4PC の場合であり，キャッシュ容量 128KB における 4PC の場合と同等の実行速度であることがわかる．実行速度比は 0.7%程度しか落ちておらず，倍のキャッシュ容量を搭載した場合と同等の性能を実現しているといってよい．

表 1　シミュレーションにおけるキャッシュパラメータ

キャッシュサイズ	L1 命令キャッシュ	32KB(全 PC 合計)
	L1 データキャッシュ	16KB ～ 128KB
	L2 データキャッシュ	512KB
アクセスレイテンシ	L1 命令キャッシュ	1 サイクル
	L1 データキャッシュ	1 サイクル
	L2 キャッシュ	16 サイクル
	メモリ	80 サイクル
連想度	L1 命令キャッシュ	1 (ダイレクトマップ)
	L1 データキャッシュ	2 / 4
	L2 キャッシュ	8
ラインサイズ	L1 命令キャッシュ	32byte
	L1 データキャッシュ	32byte
	L2 キャッシュ	64byte

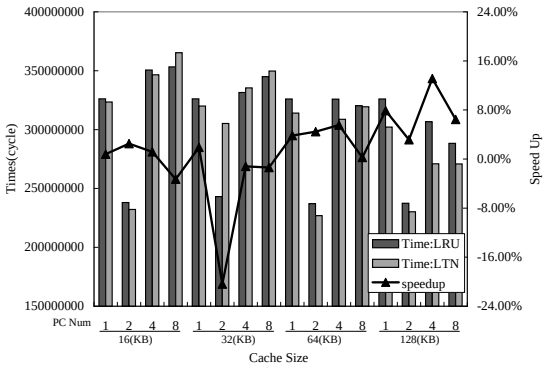


図 6　256 × 256 正方行列の積による評価結果

4.2.3　4 ウェイセットアソシアティブ方式における評価結果

2 ウェイセットアソシアティブ方式とほぼ同様であり，行列サイズが小さい場合は LTN リプレース方式によって性能低下を招いているものの，一部を除いて大きな性能低下ほとんどは見られなかった．

行列サイズ 64 × 64 の評価結果を図 7 に示す．キャッシュ容量 16KB において，1PC，2PC の場合にそれぞれ 14%，6.5%程度の速度低下が見られる．しかし，それ以外では 0%から 1%程度の速度低下であった．また，速度向上率のグラフが重なっており，LTN リプレース 2 セット方式の LTN リプレース 1 セット方式の間に大きな違いがないことがわかる。

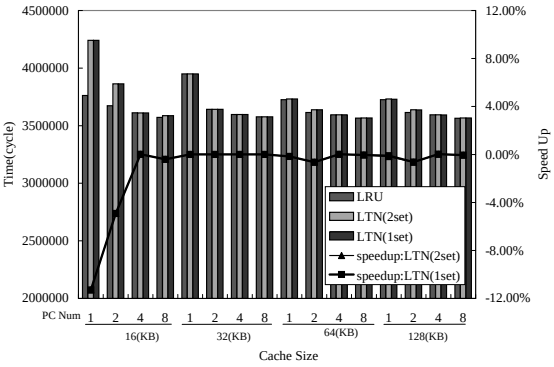


図 7　64 × 64 正方行列の積による評価結果

次に，図 8 に行列サイズ 256 × 256 の評価結果を示す．両者の方式で LRU 方式より LTN リプレース方式が高速である傾向にある．アクセスデータ量と速度の関係は 2 ウェイセットアソシアティブ方式のときと同様の傾向であるといえる．つまり，アクセスデータ量が多いほど LTN リプレース方式が有効であるといえる．

さらに速度向上率のグラフが示す通り，キャッシュ容量 128KB では LTN リプレース 2 セット方式より LTN リプレース 1 セット方式のほうが実行速度が向上している．特にキャッシュ容量 128KB における 4PC では LTN リプレース 2 セット方式が LRU 方式に対

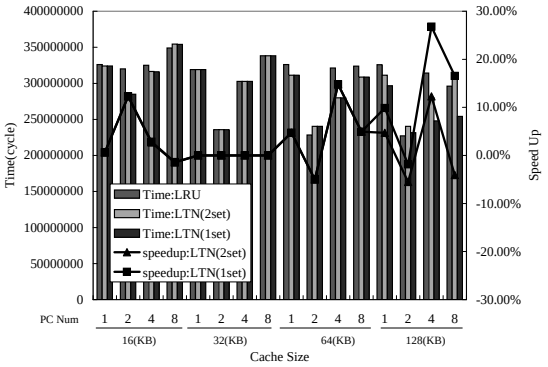


図 8 256 × 256 正方行列の積による評価結果

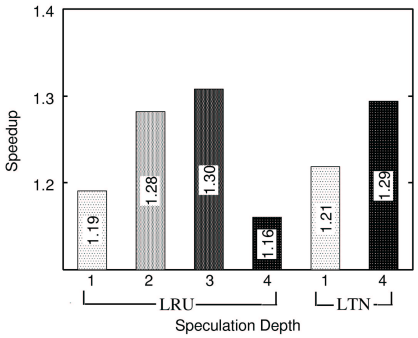


図 9 投機深度変更時の速度変化

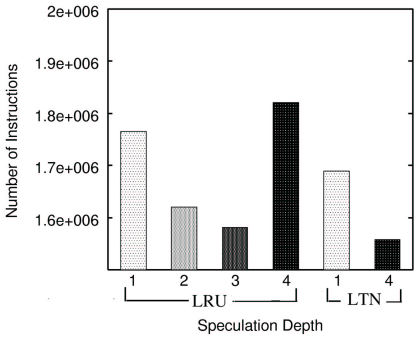


図 10 投機深度変更時のキャッシュミス数

して 12%高速であるのに対し，LTN リプレースし 1 セット方式が 26%高速であることがわかる．

4.3 ヒット率と投機深度

先の評価結果 (4.2.2 及び 4.2.3) から行列サイズ 64 × 64 場合、PC 数を増加させると実行速度が向上する傾向にあることが分かる．これは，複数のスレッドが同時実行されることにより，あるスレッドがメモリアクセス時にキャッシュへデータブロックをコピーし，リプレースされる前に別のスレッドがアクセスした場合，先にアクセスしたスレッドが後にアクセスしたスレッドのためのプリフェッチを行う事と等価である．このようなことからキャッシュヒット率が向上し，実行ユニット数を増加させなくても実行速度が向上したと考えられる．しかし行列サイズ 256 × 256 の場合，PC 数 4，8 の場合は PC 数 2 より性能低下している．これは，スレッド間のエントリ競合の影響でプリフェッチされたデータブロックがアクセスされる前にリプレースされたものと考えられる．

これらのことは PC 数の増加だけでなく，同一 PC 数で分岐時に投機的に実行する段数 (以下，投機深度と呼ぶ) を増加させ，プリフェッチ要因となる命令を増加させた場合も同様の傾向が見られると考えられる．そこで，同一 PC 数にて投機深度を増加させ実行速度やキャッシュに対する影響について表 2 に示す環境で評価を行う．

評価結果を図 9 及び図 10 に示す．図 9 は投機深度と実行速度向上の関係を示しており，横軸が，LRU によるリプレース方式，LTN リプレース方式それぞれにおける投機深度，縦軸は PC 数，投機深度を 1 とした場合の速度比である．また，図 10 は投機深度とメモリロードのキャッシュミス数の関係を示したものであり，横軸が LRU によるリプレース方式，LTN リプレース方式それぞれにおける投機深度，縦軸がキャッシュミスを起こした命令数である．

表 2 シミュレーションにおけるキャッシュパラメータ

キャッシュサイズ	L1 命令キャッシュ	32KB(全 PC 合計)
	L1 データキャッシュ	32KB
	L2 データキャッシュ	512KB
アクセスレイテンシ	L1 命令キャッシュ	ペナルティなし
	L1 データキャッシュ	ペナルティなし
	L2 キャッシュ	20 サイクル
	メモリ	100 サイクル
連想度	L1 命令キャッシュ	1 (ダイレクトマップ)
	L1 データキャッシュ	2
	L2 キャッシュ	8
ラインサイズ	L1 命令キャッシュ	32byte
	L1 データキャッシュ	32byte
	L2 キャッシュ	64byte

まず，従来の LRU によるリプレース方式を実装したキャッシュでは投機深度 3 までは性能向上しているが投機深度 4 では投機深度 1 よりも性能が低下している．これは，投機深度を増加させたことにより複数のスレッドから，メモリアクセスが増加し，その結果キャッシュミス率が増加したものと思われる．実際に，キャッシュミス数を調べてみると図 10 のように投機深度 4 で，キャッシュミス率が増加していることが分かる．これは，メモリアクセスの増加によりキャッシュエントリの競合が起こったからであると考えられる．

そこで，LTN リプレース方式を用いて投機深度 1, 4 の評価を行ったところ，どちらにおいても LRU によるリプレース方式に対し，性能向上が見られた．図 10 に示す通りキャッシュミス数が減少していることが分かる．特に，投機深度 4 の場合は実行速度において 10%以上の性能向上が見られ，LTN リプレース方式によるリプレースの制限が性能向上につながっていることが分かる．

5. シミュレーション結果からの考察

5.1 アクセスパタンとヒット率

評価結果から，キャッシュサイズに対してデータアクセス量が大きいほど，LTN リプレース方式が有効であり，逆にキャッシュサイズに対し，データアクセス量が少ない場合は LTN リプレース方式は有効ではないことが分かった．これは，キャッシュラインのリプレースの原因と，アクセスするエントリの範囲の関係によるものだと思われる．図 11 にアクセス範囲の

狭いスレッドを 2 つ実行した例を示す．図 11 において，スレッド 0，スレッド 1 が PE 上で実行されているが，エントリ i についてはスレッド 1 のアクセス対象にはならず，スレッド 0 しかアクセスしていない．この場合，LTN リプレース方式ではスレッド 0 はウェイ 2，3 のラインのリプレースを行うことができない．ウェイ 2,3 のラインに対して，一度メモリからのコピーが行われるとスレッドの実行が終了するまでリプレースができない．ウェイ 2,3 のラインのデータがスレッド 1 がスレッド終了するまでアクセスされないデータだった場合，ウェイ 2,3 のラインは存在しない状態と変らず，キャッシュの利用効率を下げることになる．つまり，ある程度，スレッドの処理が進むと，使用されないラインが存在することになる．この状態では他スレッドからのリプレースではなく，自スレッド内でのリプレースが問題になる．

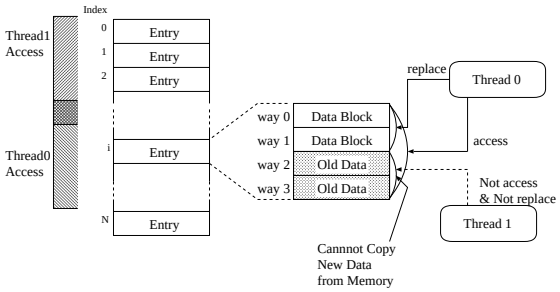


図 11 あるエントリを単一スレッドしか参照しない場合

以上のことから，PC 数 1 の場合に LTN リプレース方式を用いると常にリプレース不可能なウェイが存在する．リプレース不可能なウェイが実行中のスレッドで使われないデータブロックを保持している場合，LRU によるリプレース方式よりキャッシュの利用効率が下がる．しかし，実際には PC 数 1 でも LTN リプレース方式が有効な場合も存在する．これは，先に実行されたスレッドがキャッシュに保持したデータブロックが有効かつ利用頻度が高いものと考えられ，現在実行中のスレッドのキャッシュリプレースの対象にならないことが逆にパフォーマンス向上につながったものと考えられる．

このような LTN リプレースの制限を利用し，プリフェッチ機構として応用することができる．例えば，モジュロが 0 である LTN をプリフェッチ用スレッド，それ以外のモジュロを通常のスレッドとする．プリフェッチ用のスレッドを同時実行し，通常のスレッドがアクセスするアドレスに対し，あらかじめロード命令を実行する．この場合，アクセスするデータブロックは通常スレッドがリプレースできないウェイに格納されるため，通常スレッドがアクセスする際に必ずキャッシュに存在する．このようにすることでキャッシュミスペナルティを隠蔽することで実行ユニットの利用率を向上させる事ができると考えられる．

5.2 トレードオフ

LTN リプレース方式を実現するためには以下のハードウェアの追加が必要である．

- LTN のモジュロ (下位 1〜2bit) をロードストアユニットへ送るためのバス．
- ディスパッチキュー，ロードストアリザベーションステーションにて LTN のモジュロを捧持するバッファ (各エントリ)
- データリクエストバスにおける LTN モジュロ用の信号線

特にタグに保持するための領域は必要ないため，低コストで実装できる．以上のハードウェアの追加はコアの面積を考えれば十分に小さいものである．メモリアクセス命令に関するバスはアドレス，データそれぞれ 32bit ずつあり，制御信号を無視しても 64bit ある．それに対して追加するバスは 1bit から 3bit なのでメモリアクセスに関係するバスのみに着目しても 3%以下の追加である．プロセッサコア全体では 1%以下で済む．これは速度向上を考えれば充分小さいものである．

6. まとめと今後の展望

本論文では，SMT アーキテクチャにおけるデータキャッシュのリプレース方式として，LTN リプレース方式を提案した．他スレッドによるキャッシュラインのリプレースを抑制するため，LTN のモジュロによってリプレース可能なウェイを制限することで他スレッドからのリプレースを制限し，パフォーマンスの向上を目指す．プロセッサシミュレータ MUTHASI により，LRU リプレース方式，LTN リプレース方式の両方を比較し次に示す結果を得た．

- アクセスするデータ領域が大きくなる場合において LTN リプレース方式は LRU リプレース方式よりパフォーマンスが高い．
- 4 ウェイセットアソシアティブ方式において LTN のモジュロを 1bit にした場合よりも 2bit にした場合にの方が優位である．
- 同一 PC 数にて，投機深度を増加させることにより命令供給量を増加させた場合，LTN リプレース方式の方が優位である．
- LTN リプレース方式は LRU によるリプレース方式と比較して，2 ウェイセットアソシアティブ方式で最大 15%，4 ウェイセットアソシアティブ方式では最大で 26%高速であった．
- LTN リプレース方式の実装はハードウェアの追加は非常に少なく，プロセッサコアへのハードウェアの追加は 1%程度の増加で可能である．

以上より，LTN リプレース方式は SMT アーキテクチャにおいて，キャッシュ容量やハードウェアの追加を最小限に押えた上で高いパフォーマンスを実現するキャッシュ方式であるといえる．

今後，複数の SMT プロセッサコアを 1 チップに搭載したオンチップマルチ SMT(OChiMuS:On-chip Multi SMT)[3] について研究を行い，Ochimus に適したキャッシュについて検討する．

参 考 文 献

1) 小林, 小川, 磐田, 安藤, 島田: 非数値計算応用向けスレッドレベル並列処理マルチプロセッサ SKY 情報処理学会論文誌、Vol.42, No.2, pp.349-366, Feb, 2001
2) 中條, 河原, 上原, 並木: Simultaneous Multithread(SMT) アーキテクチャの現状と今後, 情報処理 Vol.43 No.3 Mar.2002
3) 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)
4) 佐藤, 河原, 並木, 中條: SoC 時代に向けた SMT 用 OS の構想, システムソフトウェアとオペレーティング・システム, No.91-5,pp.31-38(2002)
5) D. Tullsen, S. Eggers, and H. Levy, Simultaneous multithreading: Maximizing onchip parallelism, Proceedings of the 22rd Annual International Symposium on Computer Architecture, pp392-403, 1995.
6) Hyper-Threading Technology: http://www.intel.com