計算機アーキテクチャ 132－17
システムソフトウェアとオペレーティング・システム 80－17
ハイパフォーマンスコンピューティング 75－17
（1999．3．5）

マルチスレッドアーキテクチャ用データキャッシュ －動的スレッドアソシアティブ方式－の評価

山崎 真矢　本多 弘樹　弓場 敏嗣

電気通信大学大学院 情報システム学研究科

概要

本稿では，マルチスレッドプロセッサのキャッシュ構成として，各スレッドで使用できるキャッシュラインをスレッド処理数に応じて制限する動的スレッドアソシアティブ(Dynamically Thread-Associative)方式[1]の評価報告をする。本方式では，従来のセットアソシアティブ方式の置き換え動作を変更しキャッシュ内にスレッド専用領域を確保することで，複数のスレッド間での干渉によって起こるキャッシュミスを低減することが期待できる。シミュレータを用いて本方式の評価を行った結果，本方式により複数スレッド間での干渉を低減できることがわかった。

An Analysis of A Data Cache : Dynamically Thread-Associative for Multithread Architecture

Shinya Yamazaki　Hiroki Honda　Toshitsugu Yuba

The Graduate School of Information Systems,
The University of Electro-Communications

Abstract

We have presented a new replacement algorithm in set-associative cache adapted to multithread architecture. By restricting the replacement candidate blocks to the sub-set in a set that exclusively assigned to each thread, the cache miss rate caused by the interference among threads can be kept low. This paper shows the result of the measurements on the cache simulator.

1 はじめに

マルチスレッドアーキテクチャは，パイプラインストールによるプロセッサ利用率の低下を回避する手段，スパースカラプロセッサにおいて，プログラムの特性で命令並列度が不足することによる命令発行数の低下の回避する手段，として現在盛んに研究されている方式である。このアーキテクチャは，予め用意された複数個の命令列から実行可能な命令列を選択して処理することにより，プロセッサの利用率を高く保とうとするものである。この命令列のことをスレッドと呼ぶ。

プロセッサをマルチスレッド化することのメリットを以下に示す。

1．プロセッサ資源の利用効率が上がるため，スループットが向上する
2．OLTP(On-Line Transaction Processing)のように，入力データによって処理の流れが変化する non-deterministic な問題に対しても有効な手段となり得る[2]
3．マルチプロセッサと異なり1次キャッシュメモリはプロセッサ 1 個あたり 1 個なので，複雑なキャッシュコヒーレンシ(cache coherency)問題の発生を避けることができる

マルチスレッドプロセッサでは，各プログラムの処理時間は変わらないが，単位時間内の処理量を増やすことができる。

2 データキャッシュ構成

2．1　従来方式

これまでマルチスレッドアーキテクチャで評価されたデータキャッシュ構成方式は，以下のものがある。

(a) スレッド共有キャッシュ
(b) スレッド専用キャッシュ(プライベートキャッシュ)

(c) スレッド専用キャッシュ＋スレッド共有キャッシュ

これまでの評価結果[3][4][5][6]によって次のことがわかっている。(a)の方式では，ダイレクトマップ方式，セットアソシアティブ方式で評価を行っている。連想度を上げればキャッシュミスの低減が可能で，メモリのスループットを十分用意できれば，さらにプロセッサ利用率(CPI，Cycle Per Instruction)の向上がはかれる。(b)の方式では，スレッド間による干渉は全くないが，(a)の方式より総キャッシュ容量を十分用意しなければならなく，スレッド数が不足する場合，全く使用されないキャッシュがあるので資源の無駄となる。(c)の方式では，(b)の方式でのスレッド間でメモリ空間を共有する場合に起こる問題を解決しているが，スレッド数が不足する場合の問題点が解決されていない。

2．2　動的スレッドアソシアティブ方式(DTA方式)

動的スレッドアソシアティブ方式(以下 DTA 方式)[1]とは，従来のセットアソシアティブ方式の置き換え動作を変更した方式で，キャッシュミスにおけるキャッシュラインの置き換え動作以外は，従来のセットアソシアティブ方式と同じ動作を行う(図1)。

DTA の置き換えアルゴリズム

図2に DTA 方式の置き換えアルゴリズムを示す。例えば，状態 n＝2 は，現在スレッドが2つ処理されていることを表し，セット内に 4 ウェイあるうち，それぞれのスレッドは，ウェイ数を2割り当てられる。その割り当てられた領域以外は，置き換え対象とならい。また，2 ヶ所のうちどちらか選択するためには，LRU ビットを使用して選択する。

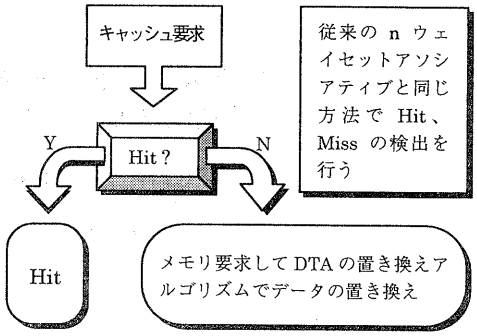

図 1：DTA 方式の動作

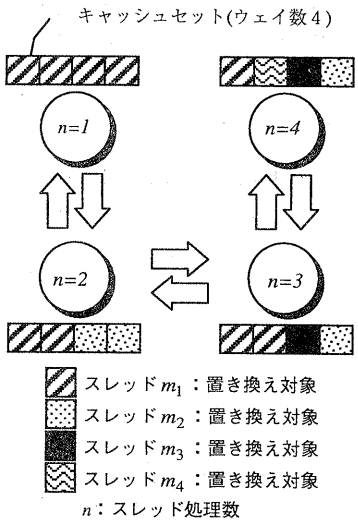

図 2：DTA の置き換え動作

3 評価

3．1　評価条件

アーキテクチャ

DTA 方式をマルチスレッドプロセッサ環境もとで評価を行うために，[6]で開発された fine-grained(Interleave)方式のマルチスレッドシミュレータ(MTSIM)に DTA 方式を組み込んで評価を行う。Interleave とはパイプラインにインターロック機構を設けることにより，同一スレッドの命令をパイプラインステージ以下の間隔で発行できるようにした方式である。表1に評価に用いる基本アーキテクチャを記す[7]。

キャッシュモデル

データキャッシュの評価を目的としているので，命令キャッシュは，100%ヒットするものと仮定した。データキャッシュモデルを表2に示す。容量はシミュレーション時に指定可能である。

メモリメモリモデル

メインメモリモデルを表 2 に示す。アクセス方式以外のパラメータはシミュレーション時に指定が可能である。なお，アドレス空間の大きさは制限がない。

キャッシュミスの内訳

本稿では，キャッシュミスを起こした原因を以下の4つに分類する。

- 初期参照
- スレッド内での競合
- 破壊的干渉[1][3]
- 建設的干渉[1][3]の消滅

表 1：プロセッサパラメータ

スレッド切り替え方式	ラウンドロビン
パイプライン	IF,ID, EX, MEM,WB
コンテクスト	4
命令発行数	1/cycle

表 2：キャッシュメモリモデル

タイプ	Write back
ラインサイズ	32 bytes
構成	DTA or n-way set associative
容量	8KB to 64KB
アソシアティビティ	1 to 8
ヒットレイテンシ	1 cycle
置き換え方式	LRU or LRU&DTA
ライトミス時の動作	Fetch on write
アドレス	Physically-index/physically-tagged

キャッシュミスを起こしたライン上でのミス前後のスレッド番号を表4のように決める(ただし，$n \neq m$)。キャッシュミスを起こした原因の内訳は表4を使って説明し，以下に定義したものとする。

スレッド内での競合：
ミス前後でスレッド番号が同じで，その後スレッドmが上書きされたメモリブロックを必要としミスが起きる
破壊的干渉：
ミス前後でスレッド番号がnからmと変わり，その後スレッドnが上書きされたメモリブロックの内容を必要としミスが起きる
建設的干渉の消滅(1)：
ミス前後でスレッド番号が同じで，その後スレッドm以外のスレッドが上書きされたメモリブロックの内容を必要としミスが起きる
建設的干渉の消滅(2)：
ミス前後でスレッド番号がnからmと変わり，その後スレッドn以外のスレッドが上書きされたメモリブロックの内容を必要としミスが起きる

3．2　評価用プログラム

各評価用プログラムの総命令数，キャッシュをオールヒットとした場合の CPI 値，ロード／ストア命令数およびデータ容量を表5に示す。

4 評価結果

4．1　オンライントランザクション処理

OLTP はスレッド間でメモリ領域を共有しているが，メモリアクセスパターンの画一化を防ぐため，8 個の異なるシード(seed)を用いて発生させた乱数をデータベースアクセスキーとして実行し，トレースデータを収集したものである。得られた 8 個のトレースファイルは，メモリアクセスパターンは異なるが同数の命令を含む。図3から図9の結果は，スレッド数8で評価を行ったものです(ただし，一度に処理可能なスレッド数は4スレッド)。また，図の比較モデルにおけるキャッシュ方式の名前および記号の nWSA，nWDTA，Single は，それぞれ n ウェイセットアソシアティブ方式，n ウェイ DTA 方式，OLTP をシングルスレッドで評価した場合，を意味する。

表 3：メインメモリモデル

アクセス方式	Split transaction
バンク数	1 to 8 banks
バンド幅	32 bytes
アクセスレイテンシ	40 cycles

表 4：ミス前後でのスレッド番号

キャッシュミス要因	スレッド番号の変化
スレッド内での競合	m→ミス→m→ミス→m
破壊的干渉	n→ミス→m→ミス→n
建設的干渉の消滅(1)	m→ミス→m→ミス→m以外
建設的干渉の消滅(2)	n→ミス→m→ミス→n 以外

表 5：評価用プログラムの詳細

プログラム	総命令数	CPI	ロード/ストア命令数	データ量
OLTP	98259	1.02	7828	128KB
qsort	19418	2.37	5544	8KB
qsort_64k	1814957	4.41	435318	64KB
Matrix	203023	2.45	99329	24KB

メインメモリのバンク数とスレッド数との比較

CPI 値とメインメモリのバンク数との関係を図3に示す。最大同時スレッド処理数の4スレッドとバンク数の関係は，図3のグラフから，スレッド処理数より大きくするのが好ましいことがわかる。キャッシュサイズが大きくなるにつれて，差がなくなっているのは，キャッシュミス率が低くなっているからである。

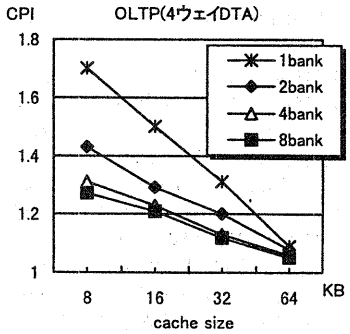

図3：バンク数とスレッド処理数4との関係

セットアソシアティブ方式との比較評価

バンク数1での評価

メモリバンク数が1の場合の評価結果を図4，図5に示す。キャッシュミス率と CPI 値共にほとんど DTA 方式とセットアソシアティブ方式での差はない。

32KB，64KB でキャッシュミス率がシングルスレッドと比較して，小さくなっているのは，キャッシュ容量が増え，建設的干渉の消滅の回数がなくなり，建設的干渉の機会が増えたのが原因である。図8でも同様ことが言える。

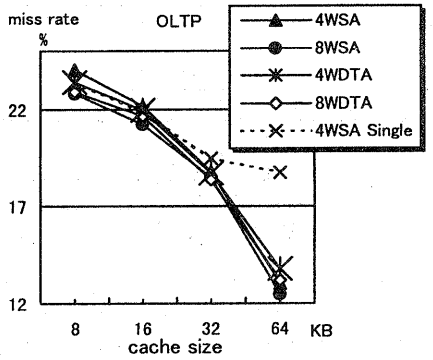

図4：キャッシュミス率
（バンク数1，メモリ空間共有）

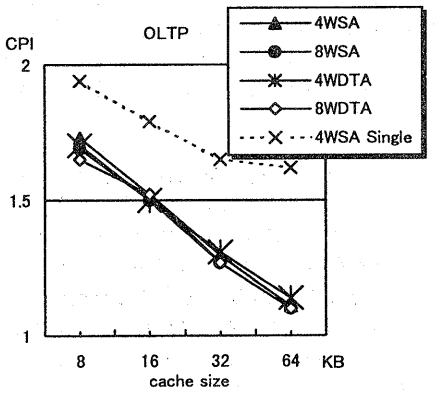

図5：CPI 値(バンク数1，メモリ空間共有)

メモリ空間共有なしでの評価

スレッド間でメモリ空間を共有しないように設定を行い評価を行った(図6，図7)。これは，メモリ空間を共有しない場合，キャッシュミス率，CPI 値にどんな影響があるか調べるためである。DTA 方式とセットアソシアティブ方式とを比較すると，8KB，16KB，32KB でのキャッシュミス率，CPI 値を比較すると DTA 方式の方が良い結果となっている。これは，スレッド間でメモリ空間を共有しないので，破壊的干渉の影響が現れた結果である。

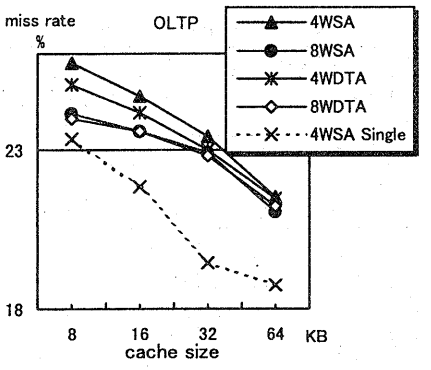

図6：キャッシュミス率
（バンク数1，メモリ空間共有なし）

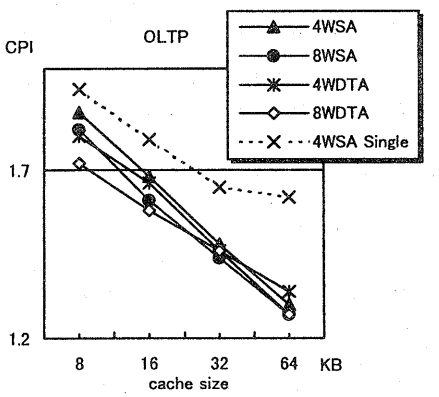

図7：CPI 値(バンク数1，メモリ空間共有なし)

キャッシュミス率はシングルスレッドよりキャッシュミス率が高くなっているが CPI 値はシングルスレッドより低くなっている。これは，マルチスレッド化によってキャッシュミスによるレイテンシが隠蔽されているからである。

オンライントランザクション処理のおいて，スレッド間でメモリ空間を共有する場合としない場合を比較すると，メモリ空間を共有した場合の方がよい結果となるのは，建設的干渉が起こるか起こらないかの違いであると考えられる。このことから，各スレッド毎にキャッシュを設けるプライベートキャッシュ方式では，建設的干渉の恩恵を受けられず，キャッシュミス率は，シングルスレッドのキャッシュミス率より常に高くなると推測できる。

バンク数8での評価

図8にキャッシュミスが起こる原因の内訳を示す。これから，図9でのセットアソシアティブ方式と DTA 方式を比べるとキャッシュミス率はほとんど変わらないが，両2方式を比べるとミスの内容が全く違うものとなっている。セットアソシアティブ方式では，スレッド内での競合と破壊的干渉の2種類

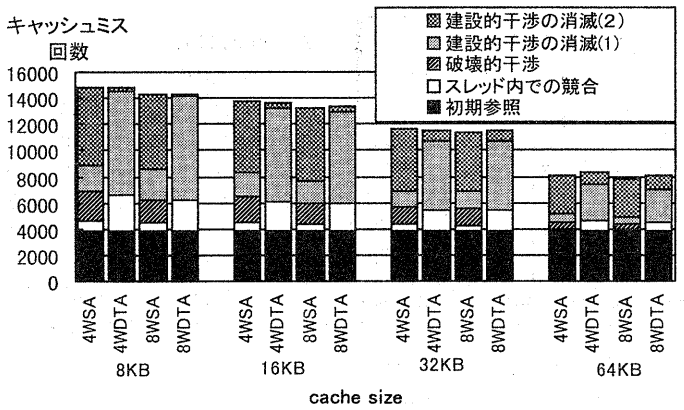

図8：キャッシュミスが起こる原因の内訳

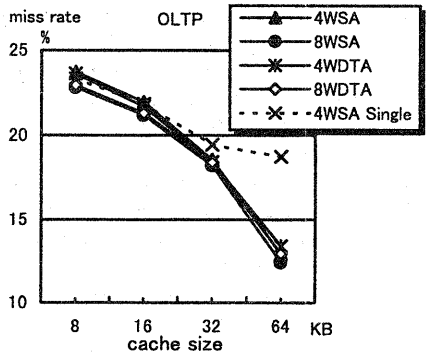

図9：キャッシュミス率
(バンク数8，メモリ空間共有)

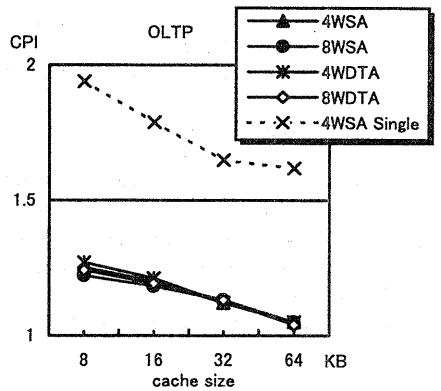

図10：CPI 値(バンク数8，メモリ空間共有)

のキャッシュミスが存在している。一方 DTA 方式では，破壊的干渉はほとんど起こらないが，スレッド内での競合がかなり存在する。DTA 方式では，破壊的干渉と建設的干渉の消滅(2)はほとんど無いが，その代わりにスレッド内での競合と建設的干渉の消滅(1)が減った回数とほぼ同じ回数増えている。図 10 でのセットアソシアティブ方式と DTA 方式の CPI 値の差は，ほとんどない。

4．2　メモリ空間を共有しない独立プログラムの場合での評価

OLTP，qsort，qsort_64，matrix の 4 スレッドで実行した場合のキャッシュミス率，CPI をそれぞれ図 11，図 12 示す。各スレッドの総命令数が異なるため，常に 4 スレッド実行されないので，命令実行数が 181495 を下回ったスレッドは，スレッド最初に戻り最初から実行されるものとして評価を行った。図 11 から 4 ウェイ DTA 方式のキャッシュミス率は，キャッシュサイズが 8KB，16KB で一番高くなっている。その後セットアソシアティブ方式のキャッシュミス率に近づいている。これは，qsort_64k のキャッシュミス率が他のスレッド比べ高いのでその影響が出ていると考えられる。4 ウェイ DTA 方式では，1 スレッドあたりウェイ数が 1 (ダイレクトマップと同等)

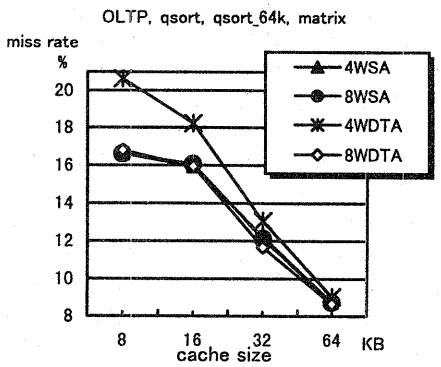

図11：キャッシュミス率(バンク数8)

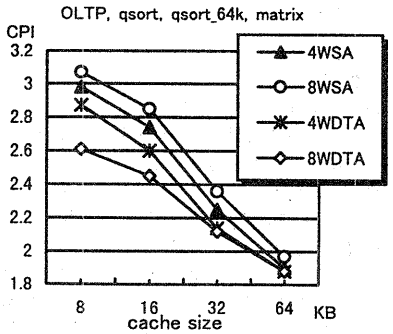

図12：CPI 値(バンク数8)

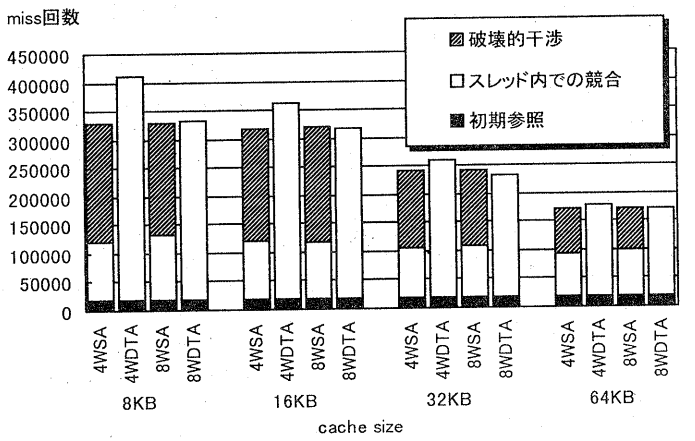

図13：キャッシュミスが起こる原因の内訳

なので，あるスレッド(この評価では qsort_64k)のキャッシュミスが高いと総キャッシュミス率にも影響する。4 ウェイ DTA と比較して 8 ウェイ DTA 方式のキャッシュミス率が 8 ウェイセットアソシアティブ方式のキャッシュミス率まで下がっているのは，1 スレッドあたりウェイ数が 2 となって，スレッド内での競合が減ったからである(図 13 参照)。

図 11，図 12 の評価結果を比較すると，DTA 方式とセットアソシアティブ方式の性能が逆転している。これは，DTA 方式の場合，キャッシュミス率の高いスレッドが他のスレッドに干渉しないで，他のスレッドの処理効率が低下しないので，平均の CPI 値は，セットアソシアティブ方式と比らべて上がらないと考えられる。さらにマルチスレッド化によって，キャッシュミスの高いスレッド(この評価では qsort_64k)のキャッシュミスを隠蔽していると考えられる。セットアソシアティブ方式の性能が悪いのは，キャッシュミスの高いスレッド(この評価では qsort_64k)が他のスレッド(この評価では OLTP，qsort，matrix)に干渉しているのが原因であると考えられる。

5 まとめ

今回の評価では，メモリ空間を共有するプログラムとメモリ空間を共有しない独立プログラムで評価を行い以下のことわかった。

- OLTP では，メインメモリのバンク数は，最大スレッド処理数以上の確保が必要
- OLTP では，キャッシュ容量が十分あり，スレッド間で建設的干渉が多く存在する場合，キャッシュミス率はシングルスレッドより低くなる
- セットアソシアティブ方式と DTA 方式を比較すると破壊的干渉が多い場合，DTA 方式の方が CPI 値がよく，破壊的干渉が少ない場合は，両方式の差は見られない。

6 おわりに

今回の評価で，スレッド間での破壊的干渉が多い場合，DTA 方式の有効性がわかったっが，他のプログラムでマルチスレッド処理する場合，スレッド間での破壊的干渉の影響が問題となることがあるのかこれから検証する必要がある。

今回の発表で，研究環境を提供してくださった方々に深く感謝いたします。

参考文献

[1] 山崎真矢，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”，情報処理学会研究報告 HPC-93,Vol.98,No.93,pp.79-84,Oct.1998.

[2] 中澤喜三郎，“計算機アーキテクチャとその構成方式”，朝倉書店，pp.361-363.

[3] Jack L.Lo，Luiz Andre Barroso，Susan J.Eggers，Kourosh Gharachorloo，Henry M.Levy，and Sujay S.Parekh“An analysis of database workload performance on simultaneous multithreaded processors”，Proc. of the 25th Ann.Int'l. Symp. on Computer Architecture，pp.39-50，June 1998.

[4] Dean M.tullsen，Susan J.Eggers，and Henry M.Levy，“Simultaneous multithreading： maximizing on-chip parallelism”，Proc. of the 22th Ann.Int'l. Symp. on Computer Architecture，pp.392-403，June 1995.

[5] 伊藤英治，相原 孝一，丹 康雄，日比野 靖，“関数型プログラムの実行に適したマルチスレッド型プロセッサ・アーキテクチャの提案”，情報処理学会研究報告 ARC-121,Vol.97,No.121,pp.81-88,Dec.1996.

[6] 吉村 隆，中澤喜三郎“マルチスレッドアーキテクチャのオンライントランザクション処理への応用”，電気通信大学電気通信学部情報工学科卒業論文，1996

[7] James Laudon，Anoop Gupta，and Mark Horowitz，“Interleaving：a multithread technique targeting multiprocessors and workstations”，ASPLOS-VI proceedings，pp.308-318，october 1994.