ハイパフォーマンス
コンピューティング 73−14
(1998. 10. 9)

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

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)方式を提案する。提案する方式は，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することによって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提案する方式の予備的評価を行った結果，提案する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。

A Data Cache Organization for Multithread Architecture

Shinya Yamazaki　Hiroki Honda　Toshitsugu Yuba

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

Abstract

We present 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. The paper also shows the result of the preliminary measurements on the cache simulator.

1 はじめに

現在のプロセッサ分野の状況[1]として，プロセッサの処理能力がデバイス技術の進歩やアーキテクチャの改良によって著しく向上してきたのに対し，メモリ素子のアクセスタイムはさほど改善されていない，命令レベル並列処理が理論的な性能の限界に近づいている，といった問題が挙げられる。これに対し，従来と比べて大容量のメモリをプロセッサチップ上に実装したり，マルチスレッドプロセッサやオンチップマルチプロセッサを構成する方式が現在盛んに研究されている。

マルチスレッドプロセッサでは，プロセッサ内で複数のスレッドを混在させ処理を行うので，以下のような現象が生じる。

- キャッシュ内に複数のスレッドのデータが混在するため複数スレッド間での干渉(interference)が発生する[2]
- レジスタ-キャッシュ間，キャッシュ-メモリの間，レジスタ-メモリ間の

データ転送がシングルスレッドプロセッサに比べて頻繁に行われるのでメモリやキャッシュのスループットの確保が必要である[3]

そこで，本研究では従来のセットアソシアティブ方式を変更した動的スレッドアソシアティブ(Dynamically Thread-Assoc-iative)方式を提案する。また，提案する方式に対してシミュレータによる評価を行う。

2 マルチスレッドアーキテクチャ

2.1 マルチスレッド処理

プログラムを実行する際，スケジューリングの最小単位をスレッドと呼ぶ。複数のスレッドを同時に実行することをマルチスレッド処理という。本研究ではマルチスレッド処理をハードウェアにより管理するマルチスレッドアーキテクチャを対象とする。

2.2 ハードウェア管理による マルチスレッド処理

ハードウェア管理によるマルチスレッド処理は，パイプラインストールや命令レベル並列度の不足などが原因で起こるプロセッサ利用率低下の回避を目的としており，スレッドはプロセッサの実行単位である。マルチスレッドアーキテクチャでは，処理に必要なレジスタ，プログラムカウンタをスレッド数分用意しておきスレッドの切替えをハードウェアで高速に行うことにより，プロセッサ内の機能ユニットなどの資源の稼働率を高めることができる。Fine-grained multithread processor[4]やSimultaneous multithread processor[5]等はハードウェア管理によるマルチスレッド処理を行う。

2.3 マルチスレッド化によるデータ キャッシュへの影響

マルチスレッド処理を行うとデータキャッシュ上には複数のスレッドのデータが混在する。その場合，キャッシュに対して複数のスレッド間での干渉が起こる。この干渉には次の２種類が存在する[5]。

- キャッシュにおいて，あるスレッドのデータが他のスレッドのデータを上書きして，結果としてスレッド間による競合ミスが増加する場合でこれを破壊的干渉(destructive interference)と呼ぶ。
- あるスレッドによってロードされたデータが他のスレッドにアクセスされ，結果としてキャッシュミスを減らす場合でこれを建設的干渉(constructive interference)と呼ぶ。ただし，この建設的干渉は，同一メモリ空間内でデータ共有するスレッド間では起こるが，メモリ空間を共有しない独立したスレッド間では全く起こらない。

3 提案するキャッシュ構成方式

提案する方式ではスレッド処理数に応じて，キャッシュシステムの内部処理機構を変化させる。

3.1 動的スレッドアソシアティブ方式

動的スレッドアソシアティブ方式(以下 DTA)とは，従来のセットアソシアティブ方式の置き換え動作のみを変更した方式で，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)。

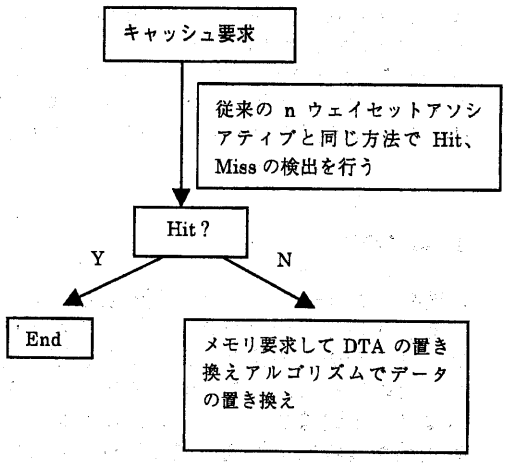
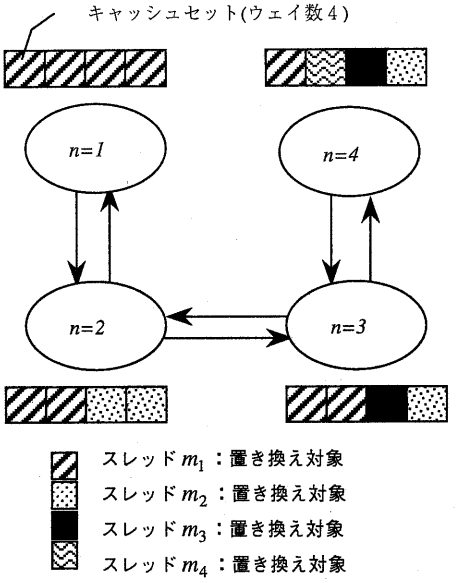


図 1：DTA の動作

n : 現在のスレッド処理数
m_i : スレッド番号
$1 \le m_i \le 4,\ m_i \ne m_j,\ i \ne j,\ 1 \le i, j \le 4$
w_i : ウェイ番号
$1 \le w_i \le 4,\ w_i \ne w_j,\ i \ne j,\ 1 \le i, j \le 4$

```
if(n=1)
    1 ≤ w1 ≤ 4;
if(n=2) {
    if(m1) w1 or w2;
    if(m2) w3 or w4;
}
if(n=3) {
    if(m1) w1 or w2;
    if(m2) w3;
    if(m3) w4;
}
if(n=4) {
    if(m1) w1;
    if(m2) w2;
    if(m3) w3;
    if(m4) w4;
}
```

(a)



(b)

図 2：DTA の置き換え動作とスレッドの置き換え対象

3.2 DTA の特徴

前章でも述べたが，複数のスレッドが同時に実行されるとき，データキャッシュ内には複数スレッドのデータが混在する。この場合，あるスレッドがキャッシュミスを起こしたときメモリから必要なデータをキャッシュにコピーするが，他のスレッドがまだ必要とするデータが存在するキャッシュラインへ上書きする場合がある(破壊的干渉)。しかし，DTA では，セット内での各スレッドの置き換え対象となる領域を限定することによってこの複数のスレッド間の破壊的干渉を減少させることができる。また，複数のスレッド間で同一メモリ空間にデータがある場合，DTA では従来のセットアソシアティブ方式と同様に，セット内のすべてのキャッシュラインに対して並列検索を行うので，他のスレッドの置き換え場所に必要なデータが存在すればキャッシュヒット(建設的干渉)となる。

3.3 DTA の置き換え動作

必要なメモリブロックのデータをキャッシュラインへコピーするとき，n ウェイセットアソシアティブ方式は，コピーできる場所がセット内に n 箇所あり，コピーする場所は LRU アルゴリズムなどを使い，その中の 1 箇所選択する。一方，DTA の置き換え動作は，現在処理中のスレッド数に応じて各スレッドの置き換え対象となる領域を限定する。また，スレッドが限定された領域内のラインを決定するためには LRU アルゴリズムを使う。図 2(a)にウェイ数 4 とした場合の DTA の置き換え動作の定義を示す。また，図 2(b)にスレッド処理数の変化に対する各スレッドの置き換え対象を示す。

4 評価

4.1 評価用アーキテクチャ

提案する 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，[5]で開発された 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 方式とはパイプラインにインターロック機構を設けることとにより，同一スレッドの命令を

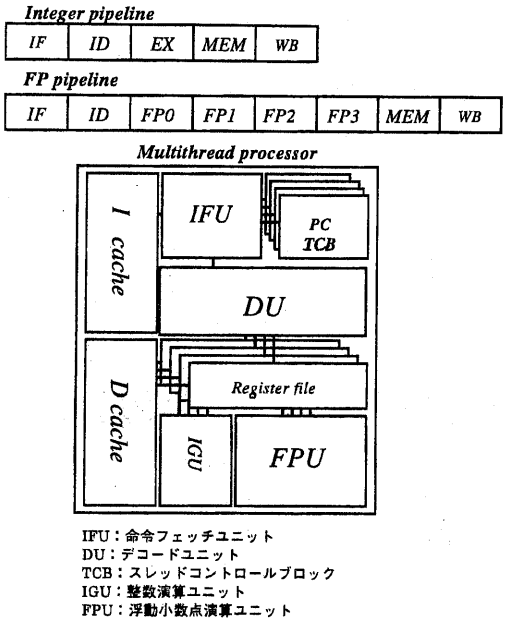


図 3：パイプライン構成とプロセッサ構成

表 1：データキャッシュモデル

データキャッシュ	
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 1MB
アソシアティビティ	4-way
ヒット・レーテンシ	1 cycle
置き換え方式	LRU
ライトミス時の動作	Fetch on write

表 2：メインメモリモデル

メインメモリ	
アクセス方式	split transaction
バンク数	1 to 16 banks
バンド幅	32 bytes
アクセスレーテンシ	40 cycles

表 3：OLTP の詳細

総命令数	CPI	ロード/ストア命令数	データ容量
98259	1.02	7828	128KB

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[3]。以下に評価に用いる基本アーキテクチャを記す。

パイプライン構成(Integer パイプライン，FP パイプライン)およびプロセッサ構成 を図 3 に示す。処理に必要なレジスタ類(Register file)，プログラムカウンタ(PC)，スレッド情報(TCB)は最大スレッド処理数分用意されている。最大スレッド処理数は 4 と設定した。処理を行うスレッドが 4 以上ある場合，処理を行っていないスレッドは待ち状態となり，優先順位の高いスレッドから順次実行を開始する。IF ステージでは，クロックサイクル毎にスレッドを切り替えて命令を発行する。切り替え方式はラウンドロビンとする。MEM ステージでキャッシュミスが発生した場合，該当スレッドはメモリアクセスが終了するまで一時停止する。本研究は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する。データキャッシュモデルを表 1 に示す。容量はシミュレーション時に指定可能である。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。アドレス空間の大きさは制限がない。

4.2 評価対象プログラム

評価対象プログラムに OLTP(オンライントランザクション処理)[6]を用いる。OLTP はメモリ上に 128KB のデータベースを持ち，乱数を発生させてこれをキーとして データの登録，検索，参照，更新および削除を行う。マルチタスクを前提として
作成されており，データベースはタスク間で共有される。評価では各タスクをスレッドとしてシミュレーションを行う。OLTP の総命令数，キャッシュをオールヒット場合の CPI 値(Cycle Per Instruction)，ロード／ストア命令数およびデータ容量を表 3 に示す。

4.3 評価結果

図 5，図 6 に OLTP のシミュレーション結果を示す。図 5 はスレッド間に共有データがある場合，図 6 は共有データがない場合で，スレッド数を 8 とした。図 6 の評価で，8 スレッドが独立したプログラムになるように設定を行ない，スレッド間の共有データをなくした理由は，複数スレッド間での破壊的干渉が多い状態とするためである。図 5, 図 6 の(a)はキャッシュサイズとミス率との関係を表し，

(b)はキャッシュサイズと CPI との関係を表している。

図5(a)では，2 方式の差がほとんどない。これは，OLTP のデータベースのサイズが小さく，ロード・ストア命令数も少ないことにより，スレッド間による干渉が少ないためと考えられる。ただし，キャッシュサイズが 128KB のとき 4 ウェイセットアソシアティブ方式のミス率が低い結果となったのは，DTA 方式では1スレッドあたりの対象となるウェイ数が1個(ダイレクトマップ等価)と 4 ウェイセットアソシアティブ方式に比べると不利な点が影響していると推測できる。

図5(b)では，DTA 方式が若干 CPI が良い結果となっている。これは，ミス率はほとんど変わらないが DTA 方式が効率のよいデータの置き換えを行っていると推測できる。128KB で逆転しているのは，図5(a) で述べた上記の説明から推測できる。

図 6(a)(b)の結果から，複数のスレッド間での破壊的干渉が多い場合には DTA の方が 4 ウェイセットアソシアティブ方式よりミス率，CPI が良いことがわかる。

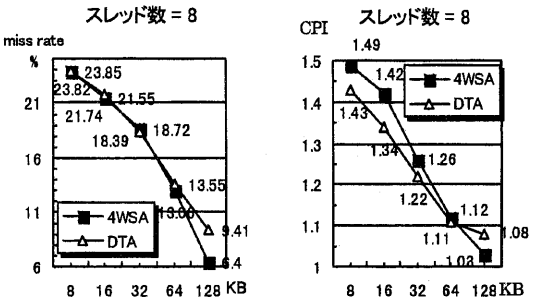


図5：シミュレーション結果(共有データ有)

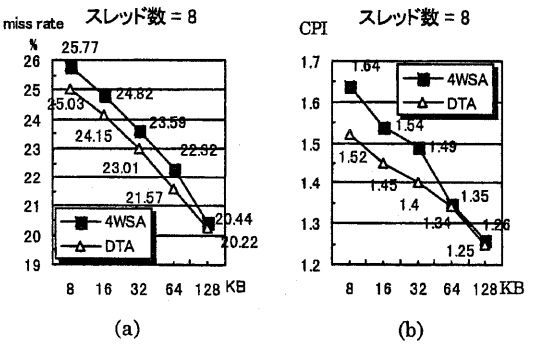


図6：シミュレーション結果(共有データ無)

5 関連研究

これまで発表されている論文では，キャッシュ方式は従来のnウェイセットアソシアティブ方式で評価を行っている研究がほとんである。[2]では，128KB の 1 次キャッシュは 2 ウェイセットアソシアティブ方式で構成されている。複数スレッドの高い干渉が存在する問題を指摘しているが 16M バイトの 2 次キャッシュでこの問題を補っている。

[7]は，スレッド毎に独立したプライベートキャッシュを設ける方式での IPC(Instruction Per Cycle)と複数スレッドでキャッシュを共有する一般的な方式での IPC とを比較評価している。プライベートキャッシュをもつ方式ではスレッド間に共有メモリ空間がある場合有効利用できない。また，スレッド数が不足するとき，スレッド専用キャッシュがまったく無駄となり，資源の有効利用がされないという問題が生じる。[8]はこの問題に関して関連研究で紹介している。

6 まとめ

今回の評価結果から以下のことが考えられる。

- DTA 方式ではスレッド処理数がウェイ数と同数で，1スレッドあたりのウェイ数が不足する場合には，性能低下が見られる
- セットアソシアティブ方式に比べて DTA 方式では複数のスレッドの破壊的干渉が多いようなスレッドに対して有効である

以上のことから，DTA のウェイ数不足に対しては，ウェイ数 4 以上のときの評価が必要である。また，実用的なスレッドでは，どのくらい複数のスレッド間での干渉があるかの分析とその評価が今後必要である。

今後評価の対象となるスレッドとして，

(1) オンライントランザクション処理のタクス
(2) 並列化したプログラムの分割部分
(3) 独立した複数のアプリケーション

を予定している。(1)では，今回使用したOLTP よりデータベースサイズ，プログラムサイズの大きい実用的なオンライントランザクション処理のタクスをスレッドとする。(2)では，科学技術計算などの並列化可能なプログラムの並列化部分をスレッドとする。(3)では行列計算，クイックソート等の独立したプログラムをスレッドとする。

また，キャッシュの比較評価モデルとして，スレッド毎に独立したプライベートキャッシュを設ける方式のモデル，キャッシュをバンク化したモデル，を追加し，比較評価を行う予定である。

参考文献

[1] Doug Burger and James R.Goodman，“Billion-transistor architectures”，Computer，pp.46-49，Sep.1997.

[2] Susan J.Eggers, Joel S.Emer, Henry M.Levy, Jack L.Lo, Rebacca L.Stamm, and Dean M.Tullsen, “Simultaneous multithreading：a platform for next-generation processor”，IEEE micro，september/october 1997，pp.12-19.

[3] 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.

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

[5] 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.

[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] 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-pp.403，June 1995.

[8] 平田 博章，奥村 晃生，柴田 幸茂，新實 治男，柴山 潔，“マルチプロセッサおよび1チップマルチプロセッサのための命令キャッシュ構成・命令フェッチ方式の性能比較”電子情報通信学会論文誌，vol.j81-D-I，no.6，pp.718-727，1998.