計算機アーキテクチャ　100－2
（1993．6．11）

セルフ・クリーンアップ型ライトバック・キャッシュの提案

森 眞一郎，大森洋一†，中島 浩，富田眞治
京都大学 工学部
（†: 現在 奈良先端科学技術大学院大学）

並列計算機システムの大規模化と，要素プロセッサの高速化にともない，メモリ・アクセスのレイテンシ隠蔽／軽減の技術が必須となりつつある．

そのような技術の1つに緩いメモリ・オーダリング・モデルが提案されており，ライト・アクセスに関するレイテンシ軽減に対しては非常に有効である．しかしながら，リードのレイテンシ軽減，特にライト・バック・キャッシュにおけるレイテンシの軽減には，あまり有効に働かない．これは最新のデータがメモリに存在しないことに伴う，ライトバック・キャッシュ固有のオーバヘッド（ライトバック・オーバヘッド）に起因する．

この問題を解決する1手法として，Write-Back 型の特徴である Write アクセスの merge 機能を保ちつつ，可能な限りキャッシュおよびメモリを clean な状態に保つ新しいキャッシュ・システムを提案する．

Proposal on Self-Cleanup Write-Back Cache

Shin-ichiro MORI, Yoichi OMORI, Hiroshi NAKASHIMA, Shinji TOMITA

Department of Information Science
Faculty of Engineering, Kyoto University
Yoshida-hon-machi, Sakyo-ku, Kyoto 606–01 Japan

E-mail: {moris, paro, nakasima, tomita}@kuis.kyoto-u.ac.jp

The latency tolerance and reduction techniques becomes more and more important as the size of the multiprocessor system grows increasing the speed of microprocessors at the same time.

Some weaker forms of memory access ordering model that have been proposed resently hopefully help to decrease the write access latencies, however, they help not so much to decrease the read latencies. In deed, they can't help for write-back cache in particular because of the write-back overhead inherent in the write-back cache.

In order to decrease the latency which comes from this write back overhead, we propose an new cache system which has an ability of write-merging while keeping the memory clean as possible.

1 はじめに

マイクロプロセッサの動作速度は，メモリのアクセス速度の向上に比べ，著しく向上している．このようなマイクロプロセッサの性能を最大限発揮させるためには，キャッシュ・メモリによるメモリ・アクセス・レイテンシの隠蔽が必須である．さらに，システムの大規模化に伴いアクセス・レイテンシの増加が無視できなくなっている現在，レイテンシの隠蔽だけではなくレイテンシの軽減のための技法も必要不可欠となりつつある．

従来のキャッシュの研究は，キャッシュの構成論的な立場でのヒット率向上によるレイテンシ隠蔽技術が主であり [20]，レイテンシ自体の軽減に関する研究は少なかった．今までに提案されているレイテンシの軽減手法としては，Write-Through(WT) 型のキャッシュに対する，Write 時のレイテンシ軽減のためのライトバッファが知られている．ライト・バッファの技術は適切なメモリ・オーダリング・モデルと組合せることで Write-Back(WB) 型のキャッシュにもその効果を発揮できる．

しかしながら，プロセッサにとって本質的なレイテンシは Write のレイテンシではなく，Read miss 時のレイテンシである．この Read レイテンシ軽減の手法としては，主にプリフェッチによる手法が研究されてきたが，レイテンシが大きいとプリフェッチ自体も有効に働かない．特に，WB 型キャッシュにおいて，メモリに最新のデータが存在しなかった場合のレイテンシは，最低でも通常の 1.5 ～ 2 倍になってしまう．

本稿では，このような WB 型キャッシュにおける Read miss 時のレイテンシを軽減するための手法を提案する．本手法ではキャッシュ内のデータに関して一種のワーキング・セット的な概念を導入し，ワーキング・セットから外れたデータのうち当該キャッシュで更新されたデータをメモリに書き戻しておくことで，当該データへの他プロセッサの Read miss に伴うレイテンシの軽減を図る．ある意味で WB 型キャッシュと WT 型キャッシュの中間的なキャッシュのメモリ更新アルゴリズムの提案であり，両者の長所を兼ね備えることで read レイテンシの軽減を図っている．

以下では，まず 2章で，本提案を行う契機となったいくつかの関連研究について述べる．次に 3章でセルフ・クリーンアップ型ライトバックキャッシュ（以下では SCC: Self-Cleanup Cache と略す.）について概説し，4章でその評価を行う．

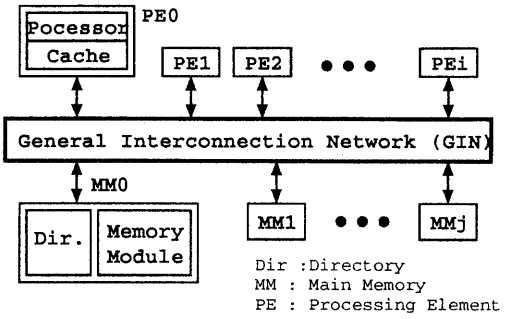

図 1: 検討の対象とする計算機モデル

2 研究の背景

2.1 前提となるシステム・モデル

ここでは，今回提案するキャッシュ・システムがもっとも有効に働く並列計算機のモデルと，いくつかの前提を示す．図 1に並列計算機モデルを示す．システムの前提としては以下のものを考える．

1. 100 ～ 10,000 規模の並列計算機
2. 相互結合網は任意の相互結合網.
3. Directory ベースのコヒーレンス制御
4. WB 型のキャッシュを採用，Write Allocate 型
5. 改良 Synapse Protocol[3][19]
6. メモリ・アクセス・レイテンシは 10 ～ 100 クロック

以上が，以降の議論の前提であるが，本提案はここで揚げたモデルに対してのみ有効であるのではなく，シングル・プロセッサ・システムや，ソフトウェアによるコヒーレンス制御を行うシステムに対しても有効である．

2.2 コンパイラ支援キャッシュ・コヒーレンス制御

本稿では，directory base のキャッシュ・コヒーレンス制御を前提として議論を進めるが，本来，今回提案するキャッシュは無効化型のコンパイラ支援キャッシュ・コヒーレンス制御（以下 CACC 制御と呼ぶ）[10] を WB 型のキャッシュで実現するために考案したものである．以下では，簡単に CACC 制御について述べ，これを WB 型のキャッシュに適用するための要件を示す．

CACC 制御は，大規模システムにおける実行時のコヒーレンス制御オーバヘッドを軽減する目

的で提案されたもので，書き込み可能なデータのキャッシングを許さないという保守的なものから[4]，コンパイル時の解析結果を利用して，Directory base のコヒーレンス制御と協調処理を行うものまで[8][7][14][19][16]，幅広く研究されている．

ここで，従来提案されてきたCACC制御につて考えてと，それらは主に無効化型のコヒーレンス制御[1]であり，コヒーレンスを維持すべきプログラムのある時点（以下，同期点と呼ぶ．）でメモリには最新のデータが存在することが前提とされている．そのため，WT型のメモリ更新アルゴリズムとの親和性が良かった．

いま，Write-Back 型のキャッシュに CACC 制御を適用することを考えると，同期点ではまず自キャッシュ内にある最新データ (dirty line) のメモリへの書き戻しを行い（これによりメモリに最新のデータがあることを保証する），その後で無効化処理を行わなければならない．無効化処理に関しては従来の WT 型キャッシュでの種々の技術 [9][17] が応用できるが，メモリへの書き戻しに関しては何等かの高速化手法を講じなければならない．その際，以下の2つ解決すべき問題が発生する．

- どの line を書き戻すかの選択
  現在はキャッシュの全検査による書き戻ししか提案されておらず，より効率のよい手法の提案が望まれている．プログラム中に明示的に Write-Back 命令を挿入するなどの手法も考えられるがあまり効率がよいとはいえない．

- Write-Back トラフィック集中の問題
  同期点に達すると，各プロセッサが Write-Back を開始するため，ネットワークの負荷が激増する．特に，DOALL 型の並列実行では複数のプロセッサがほぼ同時に同期点に達する可能性が高く，ネットワーク飽和の危険がある．

この問題の1つの解決策が，本稿で提案する Self-Cleanup Cache (SCC) である．SCC では，

要件1 Processor が行った Write アクセスのトレース，かつ

要件2 時間的に分散された（フェーズ内の適切な時点での）Write-Back 処理

を実現することにより上記の問題を解決し，Invalidation 型の CACC 制御を Write-Back キャッシュに適用可能とする．

[1]プログラムをいくつかの実行フェーズに分割し（フェーズ間ではプロセッサ間データ依存がない．），フェーズの境界において各プロセッサが自分のキャッシュ内にあるデータのうち，最新のものでなくなった可能性のあるデータを自主的に無効化することでキャッシュのコヒーレンスを保つ手法．

2.3 オーダリング・モデル

オーダリング・モデル [12][13][15][22] とは，簡単にいうと，1) 本質的に逐次化する必要のないメモリ・アクセス群の並列実行と，2) これらのメモリ・アクセスとプロセッサの命令実行のパイプライニング，を行うことで，システム全体のスループット向上を図ることを目的としたマルチプロセッサ・システムにおいて，メモリ・アクセスに関してプログラマが従うべき条件を定めたものであり，この条件を守りさえすれば，システムが Sequential Consistency であった場合と等価な実行結果が得られることを保証するものである．この条件の厳しさが，メモリ・アクセスの並列実行可能性に大きく影響し，システムの性能を左右する重要な要素の1つになる．

ハードウェアの立場でいうと，このようなモデルを設定することで，プロセッサは先行するメモリ・アクセスの完了[2]を待たずに次の命令実行が開始可能となる．またシステムレベルでは，out-of-order なメモリ・アクセスや，一時的にメモリが incoherent な状態になることが許される．これにより，逐次化のためのオーバヘッドが軽減され（逐次化の頻度自体も減少する）全体としてのスループット向上が期待できる．

いま，WB 型キャッシュの性能向上という観点からオーダリング・モデルを考えると，その効果は主に次の2点に集約できる．

- コヒーレンス処理のバックグランド化によるレイテンシの隠蔽

- ライト・アクセスのパイプライニングによるスループットの向上[3]

図2は，オーダリング・モデルがメモリ・アクセス・レイテンシに与える影響の例を示している．図は，プロセッサ Pj に clean な 状態で キャッシングされている line に，プロセッサ Pi が Write を行った場合のメモリ・トランザクション（コヒーレンス処理も含む）の例である．

この場合，ハードウェアに対して逐次化の要求が高いオーダリング・モデル (以後 Strict な Ordering と呼ぶ.) では，まずコヒーレンス処理を行った後（Pi が書き込み権 (Ownership) を獲得した後）に当該 write を行っている．これに対し，逐次化の要求が緩い場合（以後 loose な Ordering と呼ぶ.）は，メモリからのデータ・フェッチとコヒーレンス制御のための無効化要求が同時に行われており，Pi

[2]ここでは，当該メモリ・アクセスに関するコヒーレンス処理が終った時点を完了と定義しておく．厳密な定義は文献 [12] 参照．

[3]プロセッサ自体が out-of-order なメモリ・アクセスを許す場合，リード・アクセスもパイプライニング可能である．

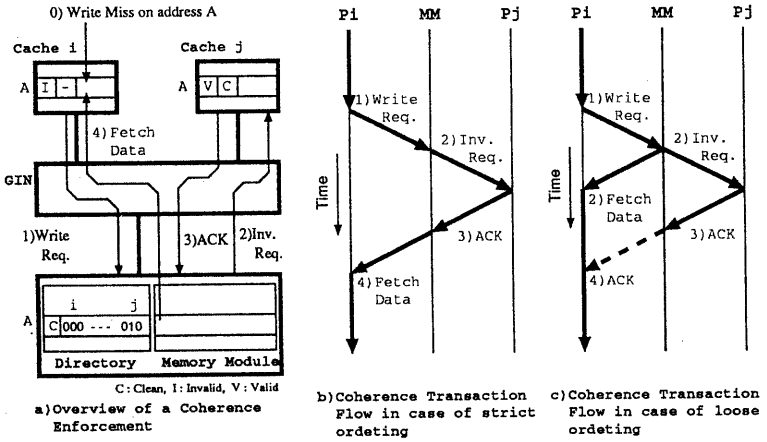

図 2: オーダリング・モデルがレイテンシの短縮に有効に働く場合の例

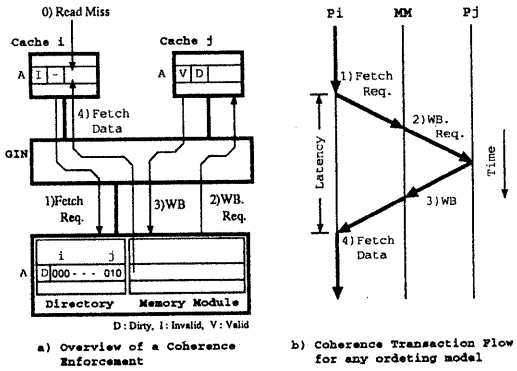

図 3: オーダリング・モデルがレイテンシの短縮に有効に働かない場合の例

はデータ・フェッチが完了した時点で直ちに write を実行している．その結果としてレイテンシが半分に短縮されている．

また，仮に Pi が当該 line を clean な状態で保持していた場合，loose な ordering では Pi はいかなるトランザクションをも待つことなく当該 Write を実行し，当該 line に対する Ownership 獲得を待たずに次のメモリ・アクセスを開始することもできる．

ところが，WB 型キャッシュを採用するシステムでは，オーダリング・モデルがレイテンシ隠蔽効果を発揮できない状況が発生する．それは，メモリに最新のデータが存在しないことに起因する．例えば，図 3に示すように，あるプロセッサ Pj が Dirty な状態で保持している line に対して，別のプロセッサ Pi が read を発行した場合，メモリ・アクセス・トランザクションは，オーダリング・モデルに関わらず同じである[4]．したがって，このような状況が頻繁に発生すると，オーダリング・モデルの恩恵を受けることができなくなってしまう．そこで，

要件 3 アクセスされなくなった diry line はできる限りすみやかにメモリへ書き戻す．

という制御を実現することで，オーダリング・モデルのパイプライン効果との相乗効果で，ライトバックのレイテンシを隠蔽するとともに，データ・フェッチのレイテンシを軽減することが可能となる．

これを実現するのが，Self-Cleanup Cache の第二の目的である．

3 Self-Cleanup Cache

3.1 Self-Cleanup Cache の概要

Self-Cleanup Cache(SCC) は，

- WB キャッシュを持つ並列計算機システムにおける，コンパイラ支援キャッシュ・コヒーレンス制御の高速化
- 任意の loose ording モデルを採用したシステムにおける，WB 型キャッシュ固有のオーバヘッド（Write-Back レイテンシ）の軽減

[4]このことは，コヒーレンス制御の隠蔽に関するオーダリング・モデルの能力の限界をも示唆している．つまり，ライトバックを伴わない無効化処理に関しては効果を発揮できるが，ライトバックを伴う場合はライトバックのオーバヘッドを隠蔽することができない．

の2つを目標に開発したキャッシュである．

SCCは，図4に示すように，通常のWBキャッシュ部に自浄制御回路（Self-Cleanup Control）を付加した構成を採る．

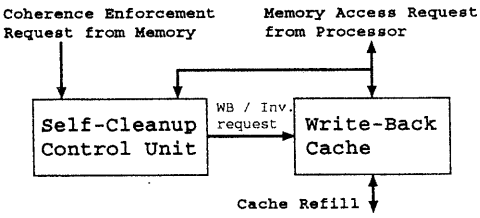

図4: SCCの構成要素

自浄制御回路は，キャッシュ内のdirtyなラインの管理と，それらのdirty lineのうち，どのラインを，いつ書き戻すかを制御する回路である．自浄制御回路が適切な時期に適切なlineに対するWrite-Back要求をWB型キャッシュ部に送ることで，2章で示した3つの要件を充足可能としている．

これによりSCCは，WB型キャッシュの特徴であるWriteアクセスのmerge機能を保ちつつ，可能な限りキャッシュおよびメモリをcleanな状態に保つことで，システム全体としてのメモリ・アクセス・レイテンシの軽減を図ることができる．ある意味で，WB型とWT型の中間のメモリ更新アルゴリズムを採用するキャッシュとみなすこともできる．

3.2 自浄制御の方針

SCCの自浄制御には，プロセッサのアクセス・パターンが時間ならびに空間の局所性をもっており，あるアドレスへのアクセスが一旦始まると，時間的にも，空間的にも連続してアクセスが行われることが多い[20]という性質を利用する．具体的には，キャッシュ内のデータに対しある種のワーキング・セットを定義し，そのワーキングセットから外れたdirty lineを順次メモリへ書き戻す制御を行う．このワーキング・セットの定義と，その管理の方法により，目的に応じたSCCを実現することが可能である．

例えば，当該プロセッサのプライベート・データ以外のdirty line N個をワーキング・セットとし，FIFOあるいはLRUでその管理を行う方法等が考えられる． FIFO管理のSCCに関しては，次節でより具体的な実現例を示す．

3.3 FIFO管理によるSCCの実現例

本節では，自浄制御回路のワーキング・セットをFIFO管理するSCCの実現例を示す．簡単のために，ワーキングセットの属性は，任意のdirty lineとする．ワーキング・セット・サイズ（すなわち，キャッシュ内のdirty lineの数の上限）はFIFOの深さで定まる．なお，以下の説明はSCCの一実現例であり，ここで述べる以外にも多種多様な実現方法が考えられる．

3.3.1 回路構成

図5に構成例を示す．前述のとおり，通常のWB型キャッシュに自浄制御回路を付加した構成をとる．自浄制御回路は，以下の5つの構成要素からなる．

- FIFOメモリ：キャッシュ内のdirtyなlineのアドレス・トレースを行う．
- アドレスレジスタ（WB ADR)：自浄制御のために，次にWBすべきdirty lineのアドレスを保持する．
- アドレス・マルチプレクサ：自浄制御に伴うライトバック要求（以後WB要求）と，メモリからのコヒーレンス制御要求（無効化要求，WB要求）の選択を行う．
- 自浄制御部：自浄制御の方針を決定する回路である．FIFOメモリにアドレスを格納すべき条件（Enqueue条件）や，自浄制御にともなうWB要求を発行する条件（Dequeue条件）を定める．

3.3.2 動作原理

プロセッサやメモリから見たキャッシュとしての動作は基本的には通常のWB型キャッシュと同じである．違いは，通常のWB型キャッシュに比べ，WBの頻度が高い点である．これは，SCCでは1) WBキャッシュ部でのline replaceに伴うWB, 2) メモリからのWB要求によるWBの他に，3) 自浄制御回路からの要求によるWBが発生することに起因する．

FIFO管理によるSCCの場合，自浄制御回路からWB要求が発生するのは，FIFOメモリがFULLの状態で，WBキャッシュ部でのWrite miss（Tag mismatch，あるいはCleanなlineへのWrite）が発生した場合である．以下に，Write miss時のSCCの動作を示す．なお，WBキャッシュ部ではプロセッサからのメモリ・アクセス要求よりも，メモリあるいは自浄制御回路からのWB要求が処理

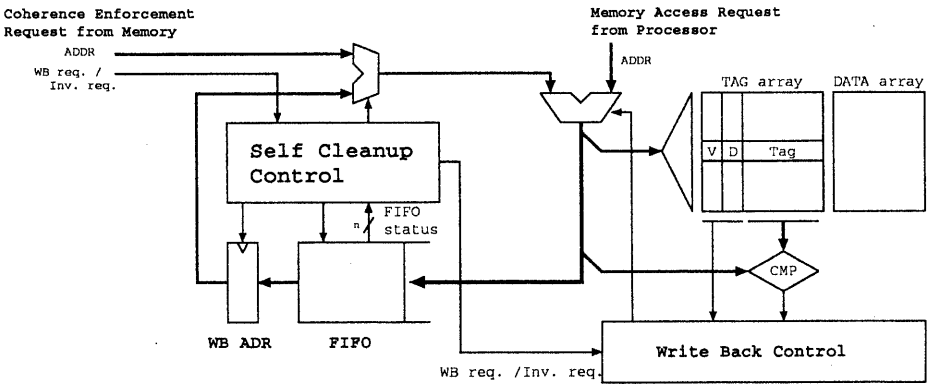

図 5: FIFO 管理による SCC の実現例

の優先順位が高いものとし，処理中の横取りはないものとする．また，自浄制御回路はそれ自身からあるいはメモリからの WB 要求を連続して WB キャッシュに出すことを許す．また（）内は WB キャッシュ部の動作である．

- FIFO が full でない場合
  miss を起こした line のアドレスをＦＩＦＯにエンキューし，(通常の miss hit 処理を行う)．
- FIFO が full の場合
  1. FIFO を dequeue し得られたアドレスを WB ADR に格納するとともに，WB キャッシュ部に対して WB 要求をアサートする．
  2. miss を起こした line のアドレスをＦＩＦＯにエンキューする．
  3. （miss を起こした line に対して通常の miss hit 処理を行う）
  4. （自浄制御回路からの WB 要求を処理する．この際当該 WB の対象となった line の無効化を行う必要はない．）
  5. メモリからのコヒーレンス制御要求がなければ，WB 要求をネゲートする．

自浄制御の実現に際しては，FIFO 管理の場合においてもさらに種々のオプションが存在する．以下にその例を示す．

- 自浄制御回路からの WB 要求の無視：ハードウェアによるコヒーレンス制御を行うシステムにおいて，高速化の目的で SCC を利用する場合は，自浄制御回路からの WB 要求を WB キャッシュ部で無視することで，ネットワークの負荷が大きい場合に，無理に WB を行うことによるスループット低下を抑えることができる．
- Half Full の状態での WB 開始：FIFO が FULL になる前に WB を開始することで，ネットワークの負荷に応じた柔軟な WB 処理（例えば，負荷が高い間は WB を見送るなど．）を可能とし，メモリ・アクセスのパイプライニング [12] を併用する場合のプロセッサ・ストールを最小限に抑える．
- dirty line flush 機能：前述の無効化型コンパイラ支援キャッシュ・コヒーレンス制御（CACC 制御）を実現するうえで必要な機能である．同期点において，この flush 機能を利用することで，FIFO に登録されている dirty line 全ての WB を行う．これにより，メモリに最新のデータが存在することを保証する．

3.4 ライト・バッファとの比較

FIFO 管理の SCC は，WT 型キャッシュにおけるライト・マージ機構付きライト・バッファ [7][24] とほぼ等価な機能を提供する．そこで本節では，SCC と WT 型キャッシュにおけるライト・バッファの簡単な比較を行う．

まず第一に，SCC ではデータ用のバッファをキャッシュ本体と別に設ける必要がない（アドレスだけでよい）ためハードウェア量の軽減が可能である．さらに，SCC と等価なライト・マージ機構付きのライト・バッファを実現するためには，小容量の WB 型キャッシュと同程度のハードウェア量が必要であり，これを考慮すると，SCC がコスト・パフォーマンス的に優れていることがわかる．

反面，データ・バッファがないことで，プロセッサのデータ・アレイ・アクセスと SCC のライト・

表 1: SCC の WS size と WB 回数の関係

Cache Size	FIFO Depth							
[Byte]	1	2	4	8	16	32	64	128
64K	30,156	20,411	14,494	10,525	9,659	2,900	590	130
128K	30,186	20,429	14,506	10,526	7,194	1,256	88	85
256K	30,206	20,438	14,506	10,506	5,039	11	8	8
512K	30,206	20,438	14,506	10,506	5,039	11	8	8

Line size = 32 bytes, Direct Map Cache

バックのためのデータ・アレイ・アクセスが競合する可能性がある．しかしながら，SCC のライト・バックは原則として，プロセッサの Write Miss によって起動されるため，いずれにせよプロセッサはストールしており，miss 処理（Owner 権の獲得を含む．）のバックグランドで SCC のライトバック処理を行うことが可能である．ただし，Clean なラインへの Write Miss 時にはプロセッサをストールしないオーダリング・モデルも考えられるが，この場合は次の Read/Write Miss 時まで SCC のライトバックを延期する方法も考えられる [25].

また，並列計算機における WB 型キャッシュの高速化という用途で SCC を利用する場合などは，必ずしも完全な形のアドレス・トレースを採る必要がないばかりか，SCC のライト・バック処理を一時的に無視する（SCC に対してはライトバックを行ったことにして，実際には何もしない．）ことも可能である．これは，SCC が最新のデータをキャッシュ本体内に保持しているために可能となったことであり，ライト・バッファでは実現不可能である．この特徴は，ネットワークの負荷が高い場合の，トラフィック増加の抑制に利用することができる．

4 シミュレーションによる SCC の評価

4.1 SCC の 基本特性

SCC はその原理上キャッシュのヒット率（Clean な line への Write も hit に含める．）とは何等相関関係を持たない．ここでは，シングル・プロセッサでの SCC の特性をワーキング・セット・サイズ（WS Size）と WB 回数の関係から評価する．

表 1に，ドライストーン・プログラムのアドレス・トレース（命令をフェッチを含む 750,000 トレース，write アクセスは 64,507）を用いたトレース駆動シミュレーション[5]の結果を示す．

表より，当該アプリケーションに対しては，WS Size 1 ～ 1 6 の範囲では，SCC は WT 型キャッシュ的な振舞いを，WS Size 3 2 以上では WB 型キャッシュ的な振舞いをしていることがわかる．

[5] シミュレータの概要は文献 [16] を参照．

表 2: SCC のレイテンシ軽減効果

#PE	WS	h	Pd	SCWB	MMWB	T_{100}
1	1	99.74	—	1024	0	—
	2	99.74	—	1023	0	—
	No	99.74	—	0	0	—
2	1	99.35	14.12	513	510	1.734
	2	99.35	28.24	1	1020	1.826
	No	99.35	28.24	0	1020	1.826
3	1	99.19	7.066	640	383	1.860
	2	98.90	14.10	255	764	2.247
	No	98.90	14.10	0	764	2.247

問題のサイズ N = 4096,NR=32, 処理の粒度 G = 2 line
h:hit ratio, Pd:Dirty ratio at Memory
SCWB: SCC からの WB 要求, MMWB: メモリからの WB 要求
Miss on Dirty Line =0, Write Hit on Clean Line =1024

アクセス集中に伴う，ライトバッファのストール等を考慮すると，当該アプリケーションに対しては，WS Size 3 2 ないし 6 4 と設定した場合に SCC の効果がもっとも期待できると考えられる．

4.2 マルチ・プロセッサ・システムでの SCC のレイテンシ軽減効果

ここでは，あらかじめアプリケーション自体のワーキング・セットが判っている問題に対して，SCC のレイテンシ軽減効果を調べる．アプリケーション・プログラムとしては NR 次のディジタル・フィルタ・プログラム（リバモア・ループ 1 4 番の変形）をソフトウェア・パイプライニングにより並列化した細粒度並列プログラムを用いる．なお，同期機構としてハードウェア・バリアを想定する．

表 reftable:filter にシミュレーション結果を示す．T_{100} はレイテンシを 1 0 0 とした場合の，平均アクセス時間である．通常の WB 型キャッシュに比べて PE 台数 2 台の場合は 5 %，4 台の場合は 1 7 %，のレイテンシ軽減効果が得られた．

今回のシミュレーションでは，ライト・バッファの溢れに伴うプロセッサ・ストールは考慮していないが，このようなバッファ溢れに対しては，SCC の負荷分散機能により，さらなる性能向上が期待できる．

5 まとめ

本稿ではWrite-Back型の特徴であるWriteアクセスのmerge機能を保ちつつ，可能な限りキャッシュおよびメモリをcleanな状態に保つことで，メモリ・アクセス・レイテンシの軽減を図るSelf-Cleanup型Write-Back Cache(SCC)を提案した．今後，さらに詳しい評価を行い，SCCの特性の解析を行う予定である．

謝辞

汎用キャッシュ・シミュレータを御提供頂いた京都大学工学部 情報工学教室 富田研究室 細見岳生氏に厚く感謝致します．また，日頃より有益な御意見を頂く，富田研究室の諸氏に感謝致します．

なお本研究の一部は文部省科学研究費補助金（重点領域研究(1) 課題番号 04235103 および奨励研究(A) 課題番号 05780243）による．

参考文献

[1] Sarita V.Adve and Mark D.Hill, "WEAK ORDERING - A NEW DEFIFINITION AND SOME IMPLICATIONS," *Comuter Sciences Technical Report*, #902, Computer Sciences Department, University of Wisconsin-Madison, December 1989.

[2] Archibald, J. and Bear, J. -L, "An Economical Solution to the Cache Coherency Problem," *Proc. 11th Ann. Int'l. Symp. Comput. Architect.*, pp.355–362, June 1984.

[3] Archibald, J. and Bear, J. -L, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," *ACM Trans. on Comuter Systems*, Vol.4, No. 4, pp.273–298, Nov. 1986.

[4] Brantley, W.C., McAuliffe, K. P., and Weiss, J., "RP3 Processor-Memory Element," *Proc. 1985 Int'l. Conf. on Parallel Processing*, pp.782–789, Aug. 1985.

[5] Censier. M. Lucie, and Feutrier. Paul, "A New Solution to Cache Coherence Problems in Multicache System," *IEEE Trans. on Computers*, Vol. C-27, No. 12, pp.1112 – 1118, Dec. 1978.

[6] Yung-Chin Chen and Alexander V. Veidenbaum, "An Improved Write Buffer Design for A Write-Through Cache," *CSRD Report No. 1105*, pp.1–17, May 1991.

[7] Yung-Chin Chen and Alexander V. Veidenbaum, "A Software Coherence Scheme with the Assistance of Directories," *CSRD Report No. 1106*, pp.1–22, June 1991.

[8] Cheong, H. and Veidenbaum, A. V., "A Cache Coherence Scheme with Fast-Selective-Invalidation," *Proc. 15th Int'l. Symp. Comput. Architect.*, pp. 299-307, June 1988.

[9] Cheong, H. and Veidenbaum, A. V., "A Version Control Approach to Cacge Coherence," *Proc. Int'l Conf. on Supercomputing '89*, pp. 322-330, June 1989.

[10] Cheong, H. and Veidenbaum, A. V., "Compiler-Directed Cache Management in Multiprocessors, " *IEEE Computer*, vol.23, no.6, pp. 39-47, June 1990.

[11] David R.Cheriton, Hendrik A.Goosen, and Patrick D.Boyle, "Pradigm:A Highly Scalable Shared-Memory Multicomputer Architecture," *IEEE computer*, pp.33–46, February 1991.

[12] Michel Dubois, Christoph Scheurich, and Faye A. Briggs, "Memory Access Buffering in Multiprocessors," *Proc. 13th Ann. Int'l. Symp. Comput. Architect.*, pp.434–442, 1986.

[13] Michel Dubois, Christoph Scheurich, and Faye A. Briggs, "Synchronization, Coherence, and Event Ordering in Multiprocessors," *IEEE computer*, pp.8–21, February 1988.

[14] Michel Dubois, "Delayed Consistency," *SCALABLE SHARED MEMORY MULTIPROCESSORS*, ed. by Michel Dubois and Shreekant Thakkar, pp.207–218, Kluwer Academic Publisheers, 1991.

[15] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy, "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors," *Proc. 17th Ann. Int'l. Symp. Comput. Architect.*, pp.15–26, May 1990.

[16] T. Hosomi, "A Hardware Cache Coherence Scheme with the assistance of software," *IPSJ technical report of SIG. Arc.*, Vol.93, No.20, pp.117–124, Mar. 1993 (in Japanease).

[17] Min.,S.L. and Baer.,J.-L., "A Timestamp-Based Cache Coherence Scheme," *Proc. 1989 Int'l. Conf. on Parallel Processing*, pp.I-23 – I-32, June 1989.

[18] S.Mori, K. Murakami, E. Iwata, A. Fukuda, and S. Tomita, "The Kyushu University Reconfigurable Parallel Processor – Cache Architecture and Cache Coherence Schemes –," *Proc. Int'l Symp. on Shared Memory Multiprocessing*, pp.218–229, April 1991.

[19] S.Mori, et al., "Overview of the ASURA: A Distributed Shared Memory Multiprocessor," *IPSJ technical report of SIG. Arc.*, Vol.92, No.48, pp.41–48, June 1992 (in Japanese).

[20] David A. Pattersion and John L. Hennessy, *Computer architecture : A Quantitaitive Approach*, Morgan Kaufmann Publishers, Inc., 1990.

[21] Stenström, P., "A Survey of Cache Coherence Schemes for Multiprocessor," *IEEE Computer*, vol.23, no.6, pp. 12-24, June 1990.

[22] Naoya Tokunaga, "Memory Consistency Models: – Frameworking and Performance Evaluation –," *Master Thesis submitted to Dep. of Information System, Kyushu University* Feb. 1993 (in Japanease).

[23] Josep Torrellas, et al., "Shared Date Placement Optimization to Reduce Muliprocessor Cache Miss Rates," *Proc. 1991 Int'l. Conf. on Parallel Processing*, pp.II-266 – II-270, 1990.

[24] "Alpha Architecture Handbook," *DIGITAL EQUIPMENT Co.*, 1992.

[25] "SuperSPARC User's Guide," *TEXAS INSTRUMENTS*, Oct. 1992.