Self-Cleanup Cache の提案

森眞一郎，福島直人，五島正裕，中島浩，富田眞治

京都大学 工学部

梗概

本稿では，Write-Back キャッシュの write-merge 効果を保ちつつ，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 Cache(SCC) を提案し，その評価を行った．その結果，SCC 開発の契機となったソフトウェア制御の一貫性制御と SCC を組み合わせた場合，すべての共有データを同期点で Invalidate したにも係わらず，実用に耐え得る性能が得られることが判明した．Directory 方式の一貫性制御との組み合わせでは，プロセッサの Read Stall 時間の短縮が確認できた．

Abstract

Improved Write-Back(WB) cache, which we call *Self-Cleanup Cache:SCC*, is proposed. The SCC automatically enforces write-backs to update memories on its own initiative. In this way, the SCC keeps itself as clean as possible, while maintaining the write merge effect of WB cache.In this paper, the behavior of the SCC is also examined by simulation of the two cache system models: 1)SCC with Selective Invalidation scheme which leads us to develop the SCC, and 2)SCC with full-map Directory scheme. The simulation results show that 1) Invalidation based software cache coherence scheme can work efficiently with the assist of the SCC, and 2) Read stall time is reduced as a result of eager write-back initiated by Self-Cleanup mechanism.

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．

プロセッサにとって本質的なレイテンシである Read miss 時のレイテンシについて考えてみる．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，メモリに存在した場合の 1.5 ~ 2 倍となり問題である．

我々は，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として，Self-Cleanup Cache を提案する [12]．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する．このワーキング・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，Self-Cleanup Cache 提案の契機となったコンパイラ支援キャッシュ・コヒーレンス制御と WB 型キャッシュとの親和性について述べたのち，Self-Cleanup Cache の Read レイテンシ軽減効果に関して簡単に言及する．次に 3 章で Self-Cleanup Cache について概説し，4章，5 章でその評価を行う．

A Proposal of Self-Cleanup Cache
Shin-ichiro Mori, Naoto Fukushima, Masahiro Goshima, Hiroshi Nakashima, and Shinji Tomita
Fuculty of Engineering, Kyoto University

2 研究の背景

2.1 コンパイラ支援キャッシュ・コヒーレンス制御

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)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[2]，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[5, 4, 10]，幅広く研究されている．

ここで，従来提案されてきたCACC制御について考えると，それらは主に無効化型のコヒーレンス制御[7][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[6, 11]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方式が提案されているのみで，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，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，Self-Invalidation型のCACC制御をWrite-Backキャッシュに適用可能とする．

2.2 レイテンシ・トレランス

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，レイテンシを何らかの形で補償する技術が必要不可欠と必要不可欠となりつつある．

レイテンシ隠蔽のための重要な概念の1つに，緩和されたアクセス順序付けモデル(以下，コンシステンシ・モデルと呼ぶ)[8, 1, 9]がある．これは，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る．この条件を守りさえすれば，システムがSequential Consistent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．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．

ところが，WB型キャッシュを採用するシステムでは，コンシステンシ・モデルに係わらず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．

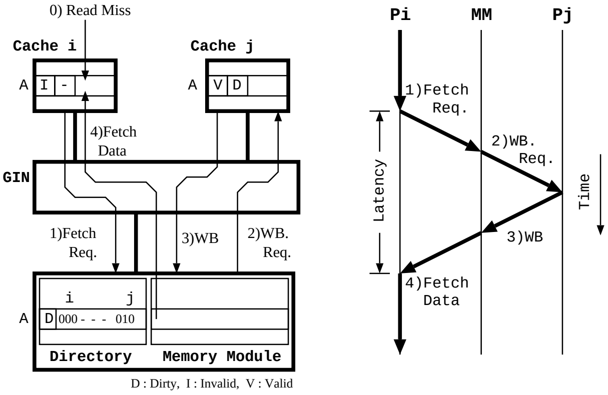
例えば，図1に示すように，あるプロセッサPjがDirtyな状態で保持しているlineに対して，別のプロセッサPiがreadを発行した場合，メモリ・アクセス・トランザクションは，コンシステンシ・モデルに関わらず同じである．したがって，このような状況が頻繁に発生すると，コンシステンシ・モデルの恩恵を受けることができなくなってしまう．そこで，

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

という制御を実現し，コンシステンシ・モデルが許すバッファリング/パイプライン処理と併用することで，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．

---

[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．



a) Overview of a Coherence Enforcement

b) Coherence Transaction Flow for any ordering model

図1: コンシステンシ・モデルがレイテンシの短縮に有効に働かない場合の例

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 ordering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

の2つを目標に開発したキャッシュである．

SCCは，図2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 Mechanism:**SCM**）を付加した構成を採る．

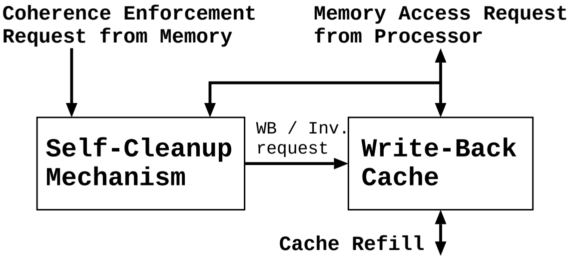


図2: SCCの構成要素

SCM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．**SCM**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[2]．

具体的な自浄制御の例としては，小容量のメモリを用いてdirty lineのアドレスをトレースし，メモリのエントリの置き換えが必要となった場合に，当該エントリに格納されたアドレスに対応するdirty lineをWrite-Backする方式などが考えられる．

3.3 ライト・バッファとの比較

自浄制御の方針として，dirty lineをLRU置換方式で管理した場合のSCC（図3参照）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[3, 15]とほぼ等価な機能を提供する．そこで本節では，LRU方式の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・バックのためのデータ・アレイ・アクセスが競合す

---

[2] cacheのspeed低下を招かない程度の回路規模にすべきである

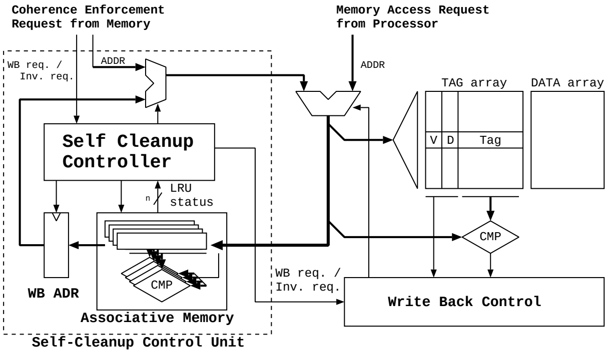


図3: SCCの実装例

る可能性がある．しかしながら，SCCのライト・バックはそれを遅らせても通常の動作には影響がないため，次のRead/Write Missでプロセッサがストールする時までSCCのライトバックを延期する方法も考えられる[16]．

また，ディレクトリ方式を採用し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 評価の方法

4.1 評価の目的

本評価の目的は以下の2点を確認することである．

1. Self-Cleanup Cacheを採用することで，Write-Back型のキャッシュ・システムにおいても，コンパイラ支援キャッシュ・コヒーレンス制御が効率的に実現可能であること．
2. Directory-baseのキャッシュ・システムにおいて，Self-Cleanup回路を付加することにより，メモリ・アクセス・レイテンシを軽減できること．

4.2 シミュレーションの対象

以下の2つのキャッシュ・システム・モデルに対して，自浄制御の方針としてLRU方式を採用したSelf-Cleanup Cacheの評価を行う．Write-Backのtimingは，LRUのリプレース時とした．

SISCモデル **S**elf **I**nvalidation型のソフトウエア・コヒーレンス制御と**S**elf-**C**leanup Cacheを組み合わせたモデル．**SISC**モデルは，同期点でcache内の全dirty lineをメモリへWrite-Backした後，privateなデータを除く全データを無効化する．この操作のために，キャッシュ・ライン毎に1bitのtagを設けており，このtagは一斉にclear可能であると仮定する[3]．

DISCモデル **Di**recotry baseのハードウェア制御キャッシュに**SC**Cを付加したモデル．**DISC**モデルでは，フル・マップのディレクトリをライン単位に保持しており，コヒーレンス・プロトコルとしてはIllinois protocol (MOdified,Exclusive Clean,Shared,Invalidの4状態)を採用した．

この2つのモデルで，3つのパラメータ：

1. LRUエントリ数(4,8,16,32,WOSCM[4])，
2. キャッシュサイズ(1MB,256KB,64KB,32KB[5])
3. ネットワーク・ディレイ(10clk, 20clk/hop)

を変化させ，実行時間とその内訳[6](プロセッサ稼働時間，Read Stall時間，Write Stall時間，同期待ち時間)，ならびに総メッセージ数(Cacheが発行するRead/Write Requst数，Memoryが発行するWrite-Back/Invalidate Request数)を計測した．

ワークロードとしては，アクセス・パターンが容易に解析可能なアプリケーションとしてFFTとSOR，ならびに，実用的なアプリケーションとしてSPLASH[14]に属するアプリケーションからWATERとMP3Dを採用した．ただし，**SISC**モデルでは，SORとFFTのみをワークロードとした．

なお，現実的な時間でシミュレーションを行うため，プロセッサ数は32とした．

4.3 シミュレータの概要

シミュレーションには実行駆動型のシミュレータを用い，実際にプロセッサ部で命令レベルのシミュレーションを行った[7]．

---

[3] 文献[5]で提案されているFast Selective Invalidation方式において，全ての共有データを*memory_read*，privateデータを*cache_read*としたものに相当する無効化方式

[4] 自浄制御なし

[5] FFTとSORの場合のみ

[6] 実行時間の内訳は，Read StallとWrite Stallの和が最も大きいプロセッサのデータである．

[7] Australia Adelaide大学のSpa package V1.0を参考に，データ分割directive，並列処理プリミティブを指定可能としている．

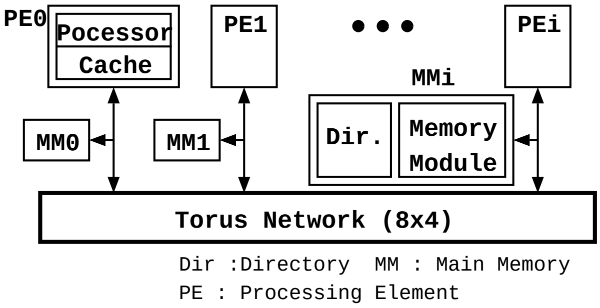


図 4: シミュレーション ・モデル

表 1: シミュレータ各部のレイテンシ

プロセッサ	1 命令の実行	1[cycle]
キャッシュ	ライン ・アクセス	4[cycle]
	ワード ・アクセス	1[cycle]
	タグ ・アクセス	1[cycle]
ネットワーク	データなし (1hop)	10[cycle]
	データ (32B) 付き (1hop)	14[cycle]
メモリ	ライン ・アクセス	10[cycle]
	ワード ・アクセス	4[cycle]
	ディレクトリ ・アクセス	4[cycle]

また，評価の対象とする計算機のモデルとしては，図 4に示すものを仮定した．以下にこの計算機モデルの構成をまとめる．また，各部でのレイテンシを表 1に示す．

- キャッシュはダイレクト ・マップ方式の Write-Back キャッシュであり，ライト ・ミス時にデータのフェッチを行う (Write Allocate 型)
- ライン ・サイズは 32 バイト
- コンシステンシ ・モデルは Weak
- 各プロセッサは最大 4 つのライト ・ミス処理（owner の獲得も含む）と 1 つのリード ・ミス処理を同時に実行可能
- キャッシュ→ネットワーク間にライトバッファは設けない．

5 シミュレーション結果

5.1 概要

シミュレーション結果を，図 5 ~ 図 14に示す．図の横軸は SCM の LRU エントリ数である．ここで，WOSCM は SCM 制御なしを，DIR は同一キャッシュ ・サイズの DISC モデルで WOSCM の場合に対応していることをそれぞれ示す．

結果の概要を簡単にまとめると以下の通りである．

1. **SISC** モデルに関して述べると，すべての共有データを同期点で Invalidate したにも係わらず，SCM を付加しない DISC モデルに対して，SOR で 50% 増，FFT で 5% 増しの実行時間が得られた．SOR の性能低下は不必要な Write-Back によるものであるが，プログラムの最適化により DISC モデルよりも高速実行が可能であることが確認できた．一方，FFT に関しては特に最適化を行わないでも十分実用に耐え得る性能が得られた．
2. **DISC** モデルに関しては，1)WriteBack Request 数の増加と，これに起因する Read Stall 時間の短縮が確認できた．但し，SOR と FFT では，Read Stall 時間の短縮を，Write Stall 時間の増加が上回ったため，最悪 3% 実行時間が長くなった．また，2)Read Request の数が不変であることから，Read Miss の増加がないことが確認できた．
3. **SCM** の LRU エントリ数は，トラッフィック数への影響を与えるものの，実行時間への影響は MP3D で若干見られる程度であった．
4. 今回測定した範囲では，Network Delay の違いは，処理時間が変化する点以外には，システムの動作特性の変化として現れなかった．なお，以下では，Network Delay = 10clock の場合の結果のみを議論する．

5.2 SOR

5.2.1 問題の性質

Even-Odd 法による差分方程式の求解プログラム．シミュレーションは 256×256 のグリッドを 32×62 のグリッドに領域分割し 2 タイム ・ステップ処理を行い，並列実行部分のデータを収集した．分割されたデータ領域は，すべて共有データとして取り扱った．SCC の欠点である「不必要な Write-Back による性能低下」が発生する例である．

5.2.2 測定結果の詳細

SISC モデル，DISC モデル 共に，**SCM** の Write Back により，トラッフィックは激増し，それにともなう Write Stall の増加のために実行時間が増加した．ただし，SISC モデルでの性能の悪さは，不必要な Write-Back よりも，不必要な無効化に起因している．

いずれの場合も，本来共有される必要のないデータを共有データとして取り扱ったために生じる性能低下であり，この種の不必要なトラッフィク増はプログラムの変更により，容易に取り除くことが可能である．

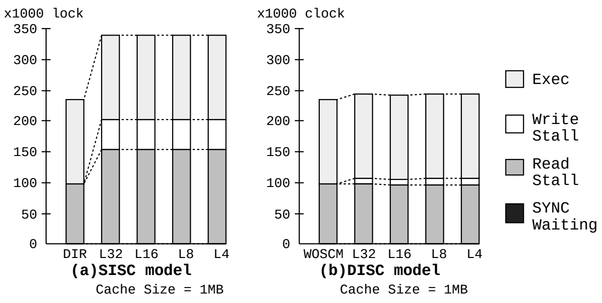


図 5: SOR の実行時間の内訳

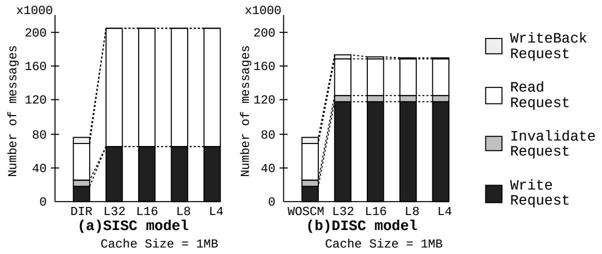


図 6: SOR のメッセージ数

図7，ならびに図8は，その例である．ここでは，各プロセッサに分割された領域の周辺部分のみを共有データとして扱っている．したがって，SISCモデルではデータの授受に必要な最小現のネットワーク・トラフィックのみが発生している．これに対してDISCモデルでは，コヒーレンス・制御のためのトラッフィク（例えば，OWNER権獲得のACK, 無効化要求，Write-Back要求）が全体のトラフィックを増やしていることがわかる．その結果，SISCモデルはDISCモデルに対し最高5%の速度向上を得ている．

また，このアプリケーションでは，アクセス・パターンが連続で，かつ，書き込みを行う共有データのワーキング・セットが３２を越えているため，LRUエントリ数の違いによる差は，SISCモデルでは無し，DISCモデルではWrite-back要求メッセージ数の減少に現れる程度で，実行時間の差は1%未満であった．

5.3 FFT

5.3.1 問題の性質

2048個の複素データをブロック分割し並列処理を行った際のデータを収集した．重み系数は全プロセッサにあらかじめ与えられているものとした．data owner computes rule に基づいており，各プロセッサが書き込みを行う共有データのワーキングセットは３２ライン（１６ラインのダブルバッファ構成）である．

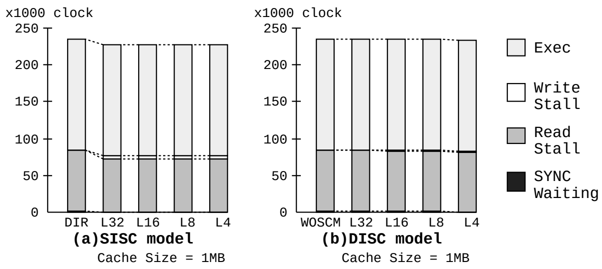


図 7: 最適化を行った SOR の実行時間の内訳

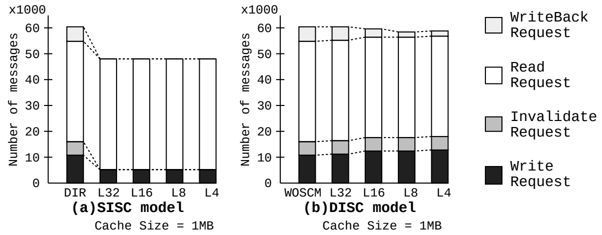


図 8: 最適化を行った SOR のメッセージ数

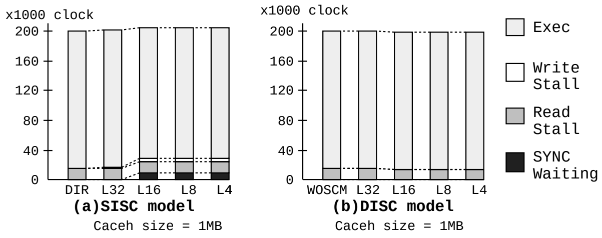


図 9: FFT の実行時間の内訳

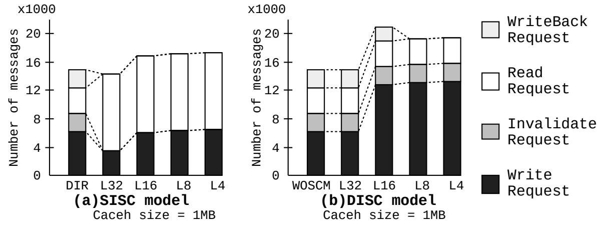
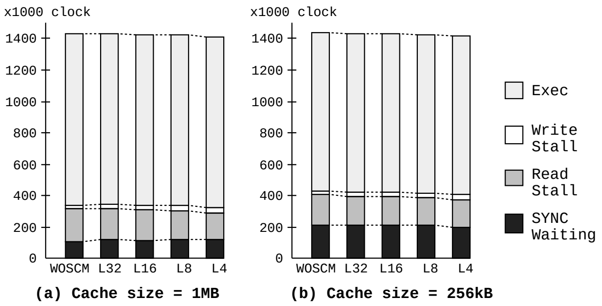


図 10: FFT のメッセージ数

5.3.2 測定結果の詳細

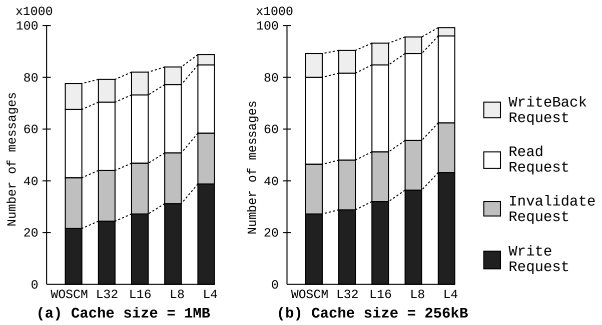
ワーキングセットが３２ラインであることから，DISCモデルではLRUエントリ数３２の場合，SCMがない場合と全く等価な結果がでている．図では有意差が見えないが，LRUエントリ数８以下では，Read Stall時間が減少しているが，逆に不必要な同期待ちのため，結果として最悪5%実行時間が延びている．

FFT計算が進んで，バタフライ演算の２要素がともに同一プロセッサ内に存在するようになると，それ以降はプロセッサ間の同期が不要となる．このため，SISCモードのLRUエントリ数３２では，Write Request数が本質的に必要な最小の値となる．これに対して，LRUエントリ３２未満では，不必要なライトバックのためWrite Request数が増加している．



(a) Cache size = 1MB　(b) Cache size = 256kB

図 11: WATER の実行時間の内訳



(a) Cache size = 1MB　(b) Cache size = 256kB

図 12: WATER のメッセージ数

5.4 water

5.4.1 問題の性質

液体状態の水分子の挙動をニュートン方程式を用いてシミュレーションするプログラム．シミュレーションは 64 分子．2 タイム・ステップ処理を行い，並列実行部分のデータを収集した．分子を静的にプロセッサに割り当て並列処理を行う．分子の運動に関するデータが 1 分子当たり 600Byte 必要である．各タイム・ステップでプロセッサは担当する分子の分子内の原子の運動を計算し，次に，自分から一定距離内にある他の分子との相互作用を求め，次状態を決定する．

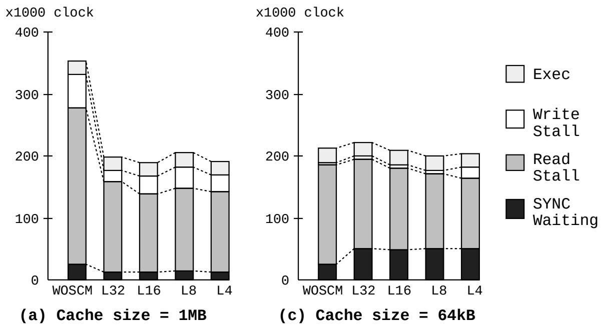
5.4.2 測定結果の詳細

Read Stall 時間は 10% から 20%程度減少しているが，Write Stall 時間や，同期待ち時間の増加により，メモリアクセス時間全体では，最高で 5% 弱の時間短縮となっている．また，メモリアクセス時間に比べ，計算時間が大きいため，全体としての処理速度の向上は最高で 2% 弱であった．

5.5 mp3d

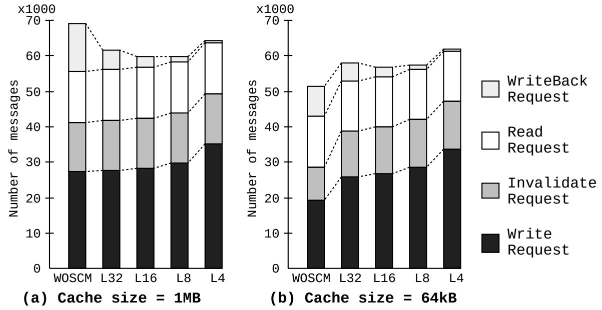
5.5.1 問題の性質

モンテ・カルロ法を用いた希薄流体力学シミュレーション・プログラム．シミュレーションは 4096 粒子，両端が開放された $16 \times 16 \times 8$ の管内に，障害物として平らな板が存在する環境で，2 タイム・ステップ処理を行い，並列実行部分のデータを収集した．各粒子は 36Byte の状態変数を，各グリッドは 40Byte の属性変数を持ち，各々静的に分散配置する．各プロセッサには一定数の粒子が割り当てられ，タイム・ステップ毎に，他の粒子の状態との関係から自分の運動をシミュレートする．メモリアクセスパターンは，ほぼランダムであると考えられる．



(a) Cache size = 1MB　(c) Cache size = 64kB

図 13: MP3D の実行時間の内訳



(a) Cache size = 1MB　(b) Cache size = 64kB

図 14: MP3D のメッセージ数

5.5.2 測定結果の詳細

キャッシュサイズ 1MB の場合，SCM を付加することで，最高で 45% 実行時間が短くなった．これは，WATER とは対象的に，メモリ・アクセスが支配的であるために現れた値である．SCC の Write-Back により，メモリが Clean である確率が高くなったことによる，Read Stall 時間の短縮であることが明確にわかる．

しかしながら，キャッシュ・サイズを 64KB とすると，SCM の効果がほとんど見られなくなった．これは，キャッシュの溢れにともないデータの replace が頻発し，これにより SCM の有無に係わらず，メモリが最新のデータを保持する確率が高くなったためである．さらに，エントリ数 32 の場合は，SCM の Write-Back が replace の妨げになり，実行時間が 4% 増える結果となっている．

6　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 Cache(SCC)を提案し，ソフトウェア制御のコヒーレンス制御との組み合わせ（SISCモデル），ならびにFull-map directory方式のコヒーレンス制御との組み合わせ（DISCモデル）の2つの場合について評価を行った．

その結果，SISCモデルでは，すべての共有データを同期点でInvalidateしたにも係わらず，SCMを付加しないDISCモデルに対して，SORで50%増，FFTで5%増しの実行時間が得られた．また，SORに関してプログラムの最適化を行った結果，SISCモデルはDISCモデルよりつねに実行時間が短いという結果が得られた．

一方，DISCモデルでは，SCMのWriteBackにより，メモリがcleanな状態にある確率が高くなることに起因するRead Stall時間の短縮が確認できた．特に，メモリ・アクセスが処理時間を支配するような例（MP3D）では，Read Stall時間の短縮により，実行時間が45%も短縮された．また，SCMは不必要なWriteBackメッセージを出すことに伴う性能低下を招くという欠点もあるが，今回測定した範囲ではその性能低下は最大で5%未満であった．

今後，SISCモデルに対しては，効率的なキャッシュ無効化方式の実装ならびに改良，DISCモデルに対しては，SCMのWriteBackがprocessorのメモリアクセスをブロックしないSCM制御方式の評価[13, 15, 16]，ならびに，プログラムからの明示的な指示によるSCMの動的制御の導入を検討していく予定である．

謝辞

本研究で使用した並列システム・シミュレータは，細見岳生氏（現NEC C & Cシステム研究所），神蔦潔氏（現Fuji Xerox）が京都大学在学中に作成したものをベースにしており，当該シミュレータなしには本研究はなし得なかった．両氏に心から感謝の意を表する．

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．

なお本研究は一部，文部省科学研究費補助金（重点領域研究(1)課題番号04235103，試験研究(A)(1)課題番号06508001，一般研究(A)(2)課題番号06402059）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] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 細見, 他：「ソフトウェアとハードウェアのコヒーレンス制御を融合したキャッシュ・コヒーレンス制御」，情処研報(ARC)，Vol.93, No.20, pp.117–124, 1993年3月

[11] 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.

[12] 森，他：「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」，情処研報(ARC), Vol.93, No.49,pp.9–16, 1993年6月

[13] S.Mori, et al., “An Proposal of Self-Cleanup Cache,” *KUIS Technical report*, No. KUIS-95-0006, Dep. of Information Science, Kyoto University, 1995.

[14] J. P. Singh, W. D. Weber and A. Gupta, “ SPLASH: Stanford Parallel Applications for Shared-Memory”, *Tech. Rep. CSL-TR-91-469,Stanford Univ*, Apr., 1991

[15] “Alpha Architecture Handbook,” *DIGITAL EQUIPMENT Co.*, 1992.

[16] “SuperSPARC User's Guide,” *TEXAS INSTRUMENTS*, Oct. 1992.