# 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†

本稿では，ライトバック型キャッシュのライト・マージ効果を保ちつつ，可能な限りキャッシュおよびメモリをクリーン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，その有効性をシミュレーションにより評価した結果を示す．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，SCCの動的制御によりライトバックのオーバヘッドが隠蔽できることを確認した．また，すべての共有データを同期点で無効化したにもかかわらず，プログラムの最適化によりディレクトリ方式に匹敵する性能が得られることも判明した．ディレクトリ方式の一貫性制御との組合せでは，プロセッサのリード待ち時間の短縮が確認できた．

# A Proposal of Self-Cleanup Cache

SHIN-ICHIRO MORI,† NAOTO FUKUSHIMA,† MASAHIRO GOSHIMA,†
HIROSHI NAKASHIMA† and SHINJI TOMITA†

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ともないアクセス・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リード・ミス時のレイテンシについて考えてみる．このリード・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，ディレクトリ・ベースで無効化型のコヒーレンス制御を行うシステムでライトバック型（以下，WB型と略す）のキャッシュを採用した場合，メモリに最新のデータが存在しない場合のリード・レイテンシは，メモリに存在した場合の1.5～2倍（2.2節参照）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リード・ミスヒット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[18]．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リード・ミスヒットにともなうレイテンシの軽減を図る．ある意味でライトバック型キャッシュ（以下，WB型キャッシュと呼ぶ）とライトスルー型キャッシュ（以下，WT型キャッシュと呼ぶ）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リード・レ

† 京都大学大学院工学研究科
Faculty of Engineering, Kyoto University

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のリード・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．

## 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，ディレクトリ・ベースのコヒーレンス制御と協調処理を行うものまで[4),5),12)]，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)] ☆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),15),25)]が応用できるが，メモリへの書戻しに関しては何ら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つの解決すべき問題が発生する．

- どのlineを書き戻すかの選択
  現在までに，1) キャッシュの全検査による書戻し方式や，2) コンパイル時の静的解析やプログラマの明示的な指示によりプログラム中にライトバック命令を挿入する手法等[8),20),22),23)]が提案されており，粗粒度並列処理を行う場合[8),22)]や静的解析が完全に可能な場合[23)]の有効性が示されている．しかしながら，前者はキャッシュの大容量化にともなう検索コストが問題となり，後者ではどのラインを書き戻すかの情報管理のオーバヘッドや，実行前に完全な解析ができない場合の安全策にともなう効率の悪さが問題となる．
- ライトバック・トラヒック集中の問題
  同期点に達すると，各プロセッサがdirty lineのメモリへの書戻しを開始するため，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．

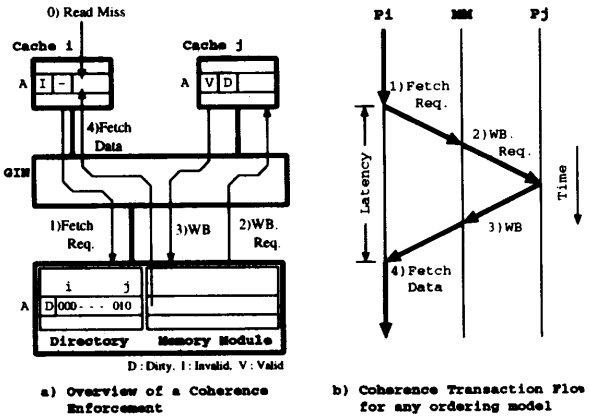


図1 ライトバック・キャッシュにおけるリード・レイテンシ
Fig. 1 Read latency on write back cache.

この問題の1つの解決策が，Self-Cleanup Cache（SCC）である．SCCでは，

**「Processorが行ったWriteアクセスのトレース」，かつ**

**「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，自己無効化型のCACC制御をWB型キャッシュに適用可能とする．

### 2.2 レイテンシの軽減

システムの大規模化にともないアクセス・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なりつつある．このようなシステムの例として，ディレクトリ・ベースの無効化型コヒーレンス制御を行うシステムでWB型キャッシュを採用した場合を考えてみる．

このシステムでは，**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ラインに対して，別のプロセッサPiが読出し（図1の0)）を行った場合のメモリ・アクセス・トランザクションの流れ

☆ 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．

は以下のようになる．

(1) Piのキャッシュが主記憶MMに対してライン・フェッチ要求を送出（図1の1)）

(2) 要求を受け取ったMMは，ディレクトリ検索の結果，そこに最新のデータが存在していないことを確認し，最新のデータを保持するPjに対してライトバック要求を送出（図1の2)）

(3) ライトバック要求を受け取ったPjは当該ラインのMMへの書戻し（図1の3)）を行う．このとき，Pjは当該ラインを無効化する必要はない．

(4) 最新のラインを受け取ったMMは，そのラインを要求元のPiへ転送（図1の4)）

今，ネットワーク等での競合がなく，かつ図1の1)～4)のトランザクションに同じ時間を要した場合のリードのレイテンシは，主記憶が最新のデータを保持していた場合の約2倍となる．また，PjからPiへ直接ラインを転送するプロトコル[14]を採用した場合でも，リード・レイテンシは約1.5倍となってしまう．

ここで，

**「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」**

という制御が実現できれば，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）[1),9),10)]が許すバッファリング/パイプライン処理との併用で，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
- 緩和されたアクセス順序付けモデル[1),9),10)]を採用するシステムにおける，WB型キャッシュ固有のオーバヘッド（ライトバック・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．この目標を達成するためには，SCCは2章で述べた以下の3つの要件を満足する必要がある．

**要件1：Processorが行ったWriteアクセスのトレース**

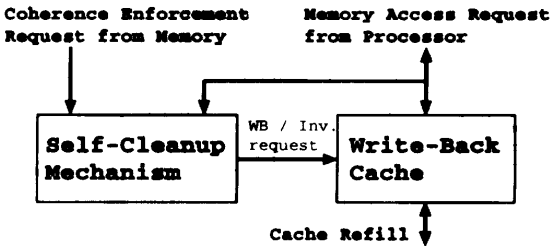


図2 SCCの構成要素
Fig. 2 Organization of the SCC.

**要件2：時間的に分散されたWrite-Back処理**

**要件3：アクセスされなくなったdirty lineの速やかなメモリへの書戻し**

そのために設けられた回路が自浄制御回路（Self-Cleanup Mechanism: **SCM**）であり，通常のWBキャッシュ部に自浄制御回路（**SCM**）を付加することで，SCCを構成する（**図2参照**）．

**SCM**は，キャッシュ内のdirtyなラインの管理と，それらのdirtyなラインのうち，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ラインに対するWB要求をWB型キャッシュ部に送ることで，上記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ライト・アクセスのマージ機能を保ちつつ，可能な限りキャッシュおよびメモリをクリーン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見なすこともできる．

### 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持っており，あるアドレスへのアクセスがいったん始まると，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なライン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．ただし，キャッシュのスピード低下を招くような複雑なキャッシュ管理/制御は行うべきではない．

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なラインのアドレスをトレースし，メモリのエントリの置換えが必要となった場合に，当該エ

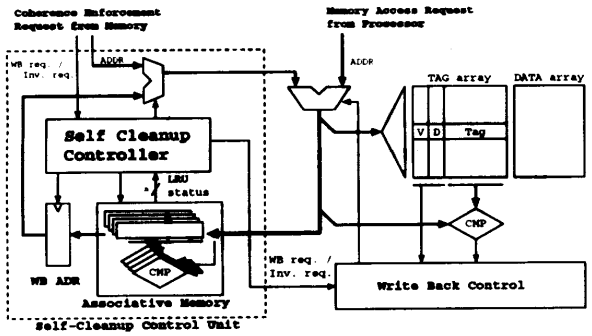


図3 SCCの実装例
Fig. 3 An implementation of the SCC with LRU replacement policy.

ントリに格納されたアドレスに対応するdirtyなラインを書き戻す方式などが考えられる.

### 3.3 SCCの構成例

自浄制御の方針として,dirtyなラインをLRU置換方式で管理した場合のSCCの構成例を図3に示す.この場合の,自浄制御回路(**SCM**)のハードウェア構成要素を以下に示す.

**連想メモリ:** キャッシュ内のdirtyなラインのアドレス・トレースを行うための小容量連想メモリ.

**アドレス・レジスタ:** 自浄制御の目的で,次にWBすべきdirtyなラインのアドレスを保持する.

**アドレス・マルチプレクサ:** 自浄制御にともなうWB要求と,メモリからのコヒーレンス制御要求(無効化要求ならびにWB要求)の選択を行う.

**自浄制御部:** 連想メモリにアドレスと格納する条件(トレース条件)や,自浄制御のためのWB要求を発行する条件を定める.

図に示すとおり,**SCM**はWBキャッシュ部に併設されており,キャッシュ・アクセス時のクリティカル・パス上には何らハードウェアの追加が行われていない.したがって,**SCM**がキャッシュ・アクセス時間の律速条件となることはない.

### 3.4 ライト・バッファとの比較

自浄制御の方針として,dirtyなラインを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),26)]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がコスト的に優れていることが分かる.

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,次の読出し/書込みのキャッシュ・ミスヒットでプロセッサがストールするときまでSCCのライトバックを延期する方法も考えられる[27)].

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・トレースをと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,トラヒ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,WB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) ディレクトリ・ベースのキャッシュ・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.書戻しのタイミングは,LRUエントリのリプレース時とした.

#### 4.2.1 SISCモデル

Self Invalidation型のソフトウエア・コヒーレンス制御とSelf-Cleanup Cacheを組み合わせたモデル.**SISC**モデルは,同期点でキャッシュ内のすべてのdirtyなラインをメモリへ書き戻した後,プライベート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・ラインごとに1 bitのタグを設けており,このタグはいっせいにクリア可能であると仮

定する☆.

#### 4.2.2 DISCモデル

Directory baseのハードウェア制御キャッシュにSCC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[24]（Dirty, Valid Exclusive, 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：

- LRUエントリ数（4, 8, 16, 32, 自浄制御なし），
- キャッシュサイズ（1 MB, 256 KB, 64 KB）
- ネットワーク・ディレイ（10クロック，20クロック（1ホップあたり））

を変化させ，実行時間とその内訳☆☆（プロセッサ稼働時間，リード待ち時間，ライト待ち時間，同期待ち時間），ならびに総メッセージ数（キャッシュが発行する読出し要求/書込み要求数，メモリが発行するWB要求/無効化要求数）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[21]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**モデルのシミュレータが，同期メカニズムとしてバリア同期のみしかサポートしておらず，かつFalse Sharingが発生するアプリケーションを実行できないため，SISCモデルではSORとFFTのみをワークロードとした．

なおプロセッサ数は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[13]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☆☆☆．コンシステンシ・モデルはWeak[1]を仮定している．

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は，複数の並列計算機[11),16),17)]の設計データを参考として，**表1**に示す値とした．

**プロセッサ** 全命令1クロックで動作．命令はすべて命令キャッシュにヒットするものとし，命令フェッチがデータアクセスの妨げになることはない．

**キャッシュ** 1階層キャッシュであり，連想度1のダイレクト・マップ方式のライトバック・キャッシュとする．また，ライト・ミス時にデータのフェッチを行うライト・アロケート型を採用する．ライン・サイズは32バイト固定である．各キャッシュでは最大4つのライト・ミス処理（書込み権利の獲得も含む）と1つのリード・ミス処理を同時に実行可能である．キャッシュとネットワークの間にライトバッファは設けない．

**メモリ** 幅64ビットのメモリで，アクセスタイムは4クロック，連続アクセス時のサイクルタイムは2クロックとする．DISCモデルでは，各ライン対応のフルマップのディレクトリを設ける．

**ネットワーク** 双方向（リンク共有）のトーラスネットワークで，フロー制御はストア・アンド・フォワード方式．データ幅は64ビット．

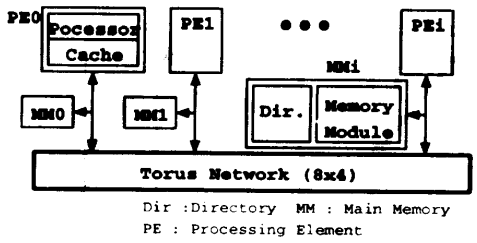


**図4** シミュレーション・モデル
Fig. 4 Simulation model.

**表1** シミュレータ各部のレイテンシ
Table 1 Systen configuration parameters.

| | | |
|---|---|---|
| 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（1 hop） | 10 [cycle] |
| | データ（32 B）付き（1 hop） | 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] |

**図5**にDISCモデルのシミュレータにおけるリード・ミス時の処理の流れとレイテンシの積算を示す．

## 5. シミュレーション結果

シミュレーション結果を，**図6～15**に示す．図の横軸はSCMのLRUエントリ数である．ここで，DIはSCM制御なしのDISCモデルを，SIはSCM制御

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

☆☆ 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．

☆☆☆ 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．

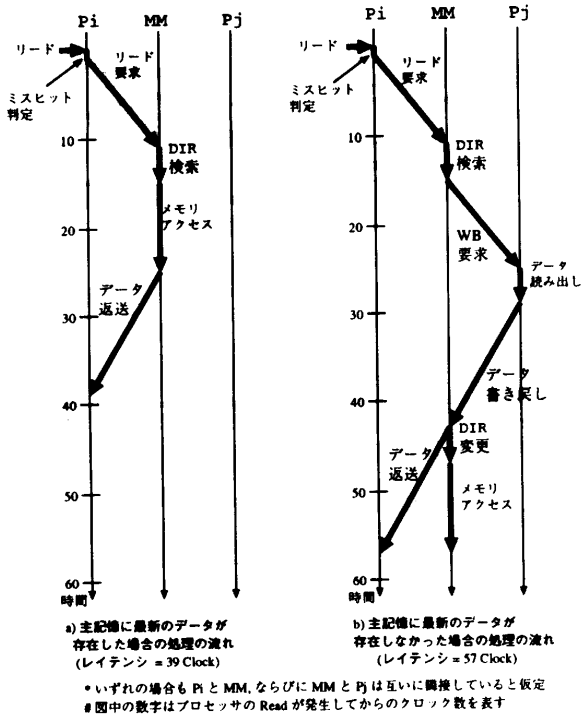


図5 リード・ミス時の処理の流れ
Fig. 5 Sequences of read miss transaction.

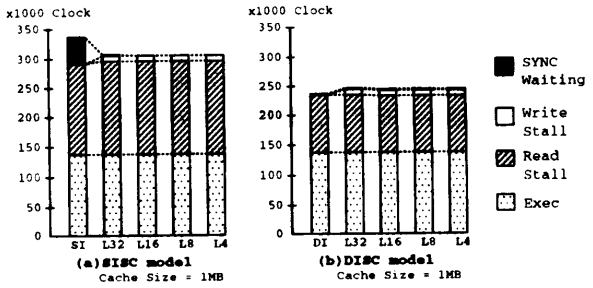


図6 SOR の実行時間の内訳
Fig. 6 SOR: execution time.

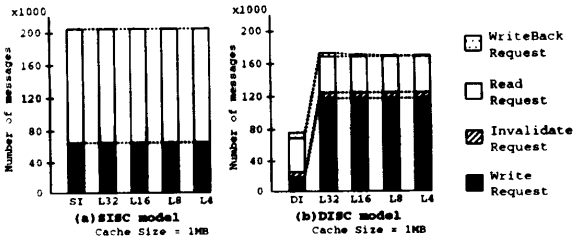


図7 SOR のメッセージ数
Fig. 7 SOR: number of messages.

なしの SISC モデルで，同期点でコンパイラが明示的にキャッシュ内の全 dirty line のライトバックを行うモデル[22]である．なお，SI では，ソフトウェアによる情報管理オーバヘッドを 0 とした理想状態のシミュレーションを行った．

今回測定した範囲では，ネットワーク遅延の違いは，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．よって，以下では，ネットワーク遅延が 1 ホップあたり 10 クロックの場合について議論する．

### 5.1 SOR

#### 5.1.1 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 32×64 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．

#### 5.1.2 測定結果の詳細

SISC モデルでは，SCM の有無にかかわらずネットワーク・トラヒック量は一定となった．しかしながら，実行時間は SCM の有無で 9%の差が生じている．これは SCM がない場合（SI）は，同期点でデータの書戻しが集中するために非常に大きな同期待ちが発生するためである．SCM を設けた場合は，同じ量の書戻しをしたにもかかわらず，その書戻しが時間的に分散しているために計算と通信のオーバラップが可能となり結果として実行時間が SI より短くなっている．

DISC モデルでは SCC の欠点が顕著に現れている．すなわち，SCM のライトバックにより，トラヒックは激増し，それにともなうライト待ち時間の増加のために SCM なしの場合（DI）に比べて実行時間が増加している．この際，SOR ではネットワーク・トラヒックが倍近く増えているが，実行時間は 4%増で収まっている．これは SISC モデルの場合と同様大半の WB 処理が他の処理とオーバラップできているためである．

次に SISC モデルと DISC モデルを比較してみる．SCM の LRU エントリ数が同じ場合，実行時間は SISC モデルが 25%長くなっている．この実行時間の差は，リード待ち時間の差からきており，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の不必要な無効化による性能低下である．この種の不必要なトラヒック増はプログラムの変更あるいは無効化方式の改良により，取り除くことが可能である．

図 8 ならびに図 9 は，プログラムの改良を行った例である．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限のネットワーク・トラヒックのみが発生している．

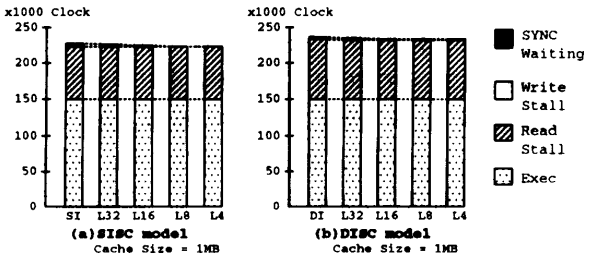


図8 最適化を行ったSORの実行時間の内訳
Fig. 8 SOR: execution time (after optimization).

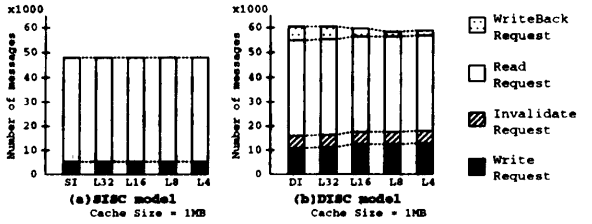


図9 最適化を行ったSORのメッセージ数
Fig. 9 SOR: number of messages (after optimization).

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ヒック（たとえば，書込み権獲得のアクノリッジ，無効化要求，WB要求）が全体のトラヒックを増やしていることが分かる．その結果，SISCモデルはDISCモデルに対し最高4%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，書込みを行う共有データのワーキング・セットが32を超えているため，LRUエントリ数の違いによる差は，SISCモデルではなし，DISCモデルではWB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．

### 5.2 FFT

#### 5.2.1 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．

#### 5.2.2 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，SISCモデルではLRUエントリ数32の場合，SCMがない場合とまったく等価な結果がでている．LRUエントリ数16以下ではWBのトラヒック数が増加しているが，通信と計算のオーバラップにより，実行時間の差は±0.1%未満となり有意な差は見られなかった．

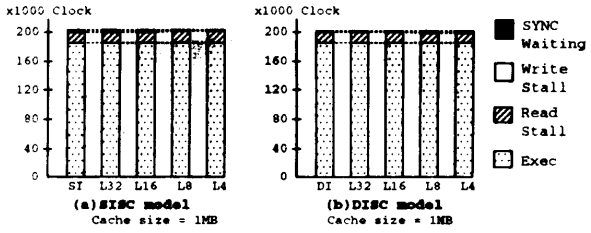


図10 FFTの実行時間の内訳
Fig. 10 FFT: execution time.

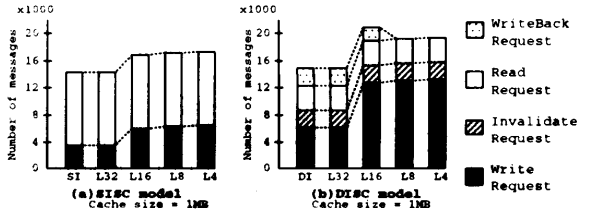


図11 FFTのメッセージ数
Fig. 11 FFT: number of messages.

DISCモデルでもほぼ同様にSCMを設けると実行時間が0.3%増加しているが有意差とは考えられない．LRUエントリ数8以下では，メモリからのライトバック要求が0となりそれに対応してリード待ち時間が短くなっている．しかしライト待ち時間が増加しているため全体での待ち時間が増えてしまった．これは以下の理由による．

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，LRUエントリ数32では，ライト要求の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ライト要求数が増加する．

### 5.3 WATER

#### 5.3.1 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分子あたり600 Byte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．

#### 5.3.2 測定結果の詳細

リード待ち時間は10～20%程度減少しているが，ライト待ち時間や，同期待ち時間の増加により，メモリ

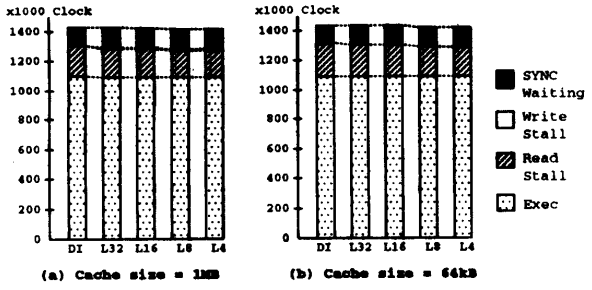


図12 WATERの実行時間の内訳
Fig. 12 WATER: execution time.

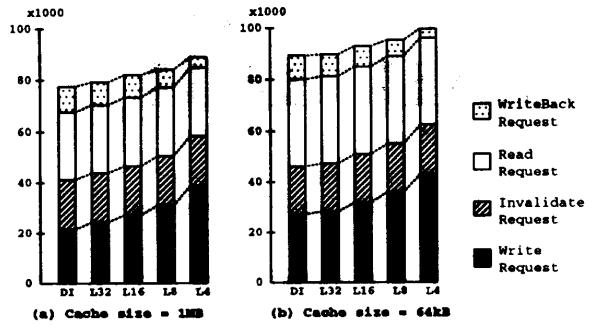


図13 WATERのメッセージ数
Fig. 13 WATER: number of messages.

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．

### 5.4 MP3D

#### 5.4.1 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36 Byteの状態変数を，各グリッドは40 Byte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・ステップごとに，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．

#### 5.4.2 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30%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のライトバックにより，メモリがクリーンである確率が高くなったことによる，リード待ち時間の短縮であることが明確に判断できる．

また，キャッシュ・サイズが64 KB，256 KB，1 MBと増加するに従ってSCMの効果が顕著に現れてくる．これは，キャッシュ・サイズが小さい間は，キャッシュのキャパシティ・ミスによるデータのリプレースのため，SCMの有無にかかわらず，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．

次に，LURエントリ数に関して，mp3dではエントリ数2および1でも評価を行った．その結果，キャッシュ・サイズ1MB，256 KBでは，エントリ数の減少とともに実行時間が短縮され，エントリ数8で最短になった後，再び増加していることが分かる．これは，エントリ数4以下ではSCCのWrite-Backにともなうオーバヘッドが Read Stall 時間の短縮を上回ったためである．ただし，キャッシュ・サイズ64 KBでは，エントリ数4までは実行時間が単調減少であるが，エントリ数2で実行時間が増加する．

なお，SCCの制御を簡単化するため，Write-Back対象のキャッシュ・エントリが非定常状態☆の場合，SCMからのWrite-Back要求を無効化（無視）する方針をとっている．そのため，エントリ数1ではエントリ数2の場合に比べ，ライト要求が減少し全体のメッセージ数が減少するという現象が現れている．

エントリ数1ないし2でのシステムの挙動は，ライトマージ機能付きライトバッファを設けたWT型キャッシュの挙動[3]とほぼ等価であると考えられる．

### 5.5 シミュレーション結果のまとめと考察

#### 5.5.1 SISCモデルに関して

SCMを設けない場合（SI），同期点でいっせいにdirtyなラインのWBを行うことによる，同期待ち時間の増加が見られた．これに対しSCMを設けた場合にはWBが時間的に分散して行われるため，WB処理と他の処理とのオーバラップ可能となりWBにともなうライト待ち時間の増加はSI時の同期待ち時間の増加より少ないことが確認できた．また，ソフトウェアの静的解析が不十分で完全な最適化ができなかった場合（最適化なしのSOR）においても，SCMの動的制御により速度向上が得られることが分かった．

今回の評価ではSCMなしの場合，キャッシュ内のdirtyなラインの情報を管理するための実行時オーバヘッドを無視しているが，実際にはこのオーバヘッドが存在するため，SCMを設けることによる効果は今

☆ メモリ・モデルをWeakとしたことにともなって発生する中間状態．たとえば，ライトミスヒットにともなうデータ・フェッチが完了するまでの状態（InvalidateからDirtyへ遷移する途中の状態等）．

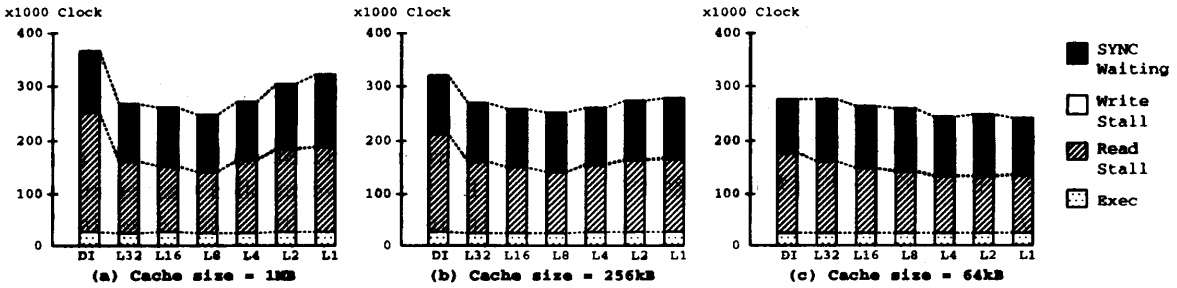


**図14** MP3Dの実行時間の内訳
Fig. 14 MP3D: execution time.

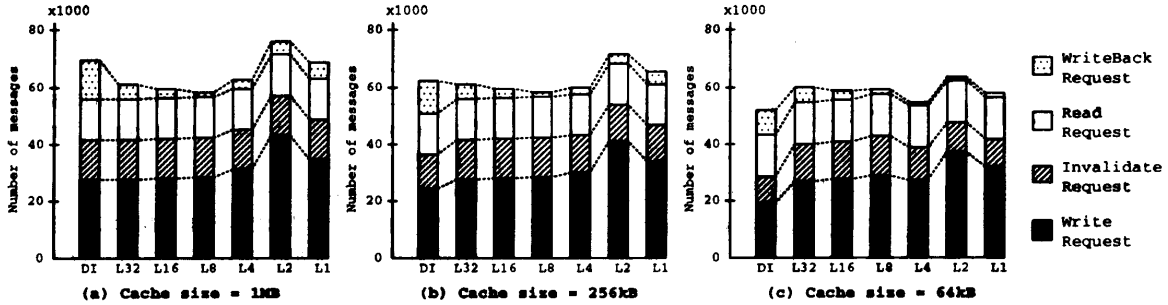


**図15** MP3Dのメッセージ数
Fig. 15 MP3D: number of messages.

回の評価結果よりも大きいものと考えられる．なお，キャッシュの全検査を行う場合を考えると，この場合の実行時間はSISCモデルでSCMなしの場合の同期待ち時間にキャッシュ容量に比例する時間が加わったものとなり，SISCモデルがさらに有利であることが分かる．

### 5.5.2 DISCモデルに関して

**SCM**のライトバックによりWB要求数が増加するが，リード待ち時間の有意な短縮を確認した．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，リード待ち時間の短縮により最大30%実行時間が短縮された．

また**SCM**には，不必要なライトバックを行うことによるライト待ち時間の増加がリード待ち時間の短縮を上回ると性能低下を招くという欠点もあるが，今回の測定結果では，最適化なしのSORで最大4%，それ以外では1%以下の性能低下であった．

さらに，SORではネットワーク・トラヒックが倍近く増えているにもかかわらず，実行時間が4%増で収まっていることから，大半のWB処理が他の処理とオーバラップできていることが分かる．また，リード要求の数が不変であることから，リード・ミスの増加がないことが確認できた．

今回評価は行っていないがDISCモデルにおいては，3.4節で述べたように，SCCのライト・バック処理を一時的に無視しトラヒックを抑制することができるため，不必要なライトバックにともなう性能低下は阻止可能であると考えられる．

### 5.5.3 SISCモデルとDISCモデルの比較

一般にSISCモデルでは，不必要な無効化のためにDISCモデルより性能が低下すると考えられる．今回評価した**SISC**モデルでも，すべての共有データを同期点で無効化したために，SCMを付加しないDISCモデルに対して，SORで25%，FFTで2%の実行時間増となった．しかしながら，プログラムを最適化し，不必要な無効化を削減することでSISCモデルがDISCモデルよりも高速になり得ることが確認できた．

## 6. まとめ

本稿ではWrite-Back型の特徴であるライト・アクセスのマージ機能を保ちつつ，可能な限りキャッシュおよびメモリをクリーン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合せ（**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合せ（**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，SCCの動的制御によ

りライトバックのオーバヘッドが隠蔽できることを確認した．また，すべての共有データを同期点で無効化したにもかかわらず，プログラムの最適化によりディレクトリ方式に匹敵する性能が得られることも判明した．**DISC** モデルにおいては，**SCM** の早期 Write-Back により，プロセッサのリード待ち時間が短縮することを確認した．

今後の課題としては，**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 がプロセッサのメモリアクセスをブロックしない **SCM** 制御方式の評価[19),26),27)]，ならびに，プログラムからの明示的な指示による SCM の動的制御の導入などがあり，順次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），神嶌潔氏（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いる．両氏に心から感謝の意を表する．また，日頃より有益なご意見をいただく，富田研究室の諸氏に感謝いたします．

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) Adve, S.V. and Hill, M.D.: Weak Ordering - A New Defifinition and Some Implications, Comuter Sciences Technical Report, #902, Computer Sciences Department, University of Wisconsin-Madison, (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 (1985).
3) Chen, Y.-C. and Veidenbaum, A.V.: An Improved Write Buffer Design for a Write-through Cache, CSRD Report No.1105, pp.1–17 (1991).
4) Chen, Y.-C. and Veidenbaum, A.V.: A Software Coherence Scheme with the Assistance of Directories, CSRD Report No.1106, pp.1–22 (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 (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 (1989).
7) Cheong, H. and Veidenbaum, A.V.: Compiler Directed Cache Management in Multiprocessors, *IEEE Computer*, Vol.23, No.6, pp.39–47 (1990).
8) Cheriton, D.R., Slavenburg, G.A. and Boyle, P.D.: Software-controlled Caches in the VMP Multiprocessor, *Proc. 13th Ann. Int'l. Symp. Comput. Architect.*, pp.366–374 (1986).
9) Dubois, M., Scheurich, C. and Briggs, F.A.: Memory Access Buffering in Multiprocessors, *Proc. 13th Ann. Int'l. Symp. Comput. Architect.*, pp.434–442 (1986).
10) Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P., Gupta, A. and Hennessy, J.: Memory Consistency and Event Ordering in Scalable Shared-memory Multiprocessors, *Proc. 17th Ann. Int'l. Symp. Comput. Architect.*, pp.15–26 (1990).
11) 富田ほか：超並列ハードウェア・アーキテクチャの研究，文部省重点領域研究「超並列原理に基づく情報処理基本体型」第6回シンポジウム予稿集，第4章 (1995).
12) 細見ほか：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).
13) 細見ほか：ディレクトリ型キャッシュコヒーレンスプロトコルの性能評価，情報処理学会論文誌，Vol.37, No.2, pp.290–299 (1996).
14) Lenoski, D., Laudon, J., Gharachorloo, K., Weber, W.-D., Gupta, A., Hennessy, J., Horowitz, M. and Lam, M.S.: The Stanford Dash Multiprocessor, *IEEE Computer*, pp.63–79 (1992).
15) 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 (1989).
16) Mori, S., Murakami, K., Iwata, E., Fukuda, A. and Tomita, S.: The Kyushu University Reconfigurable Parallel Processor Cache Architecture and Cache Coherence Schemes, *Proc. Int'l Symp. on Shared Memory Multiprocessing*, pp.218–229 (1991).
17) Mori, S., et al.: A Distributed Shared Memory Multiprocessor: ASURA - Cache and Memory Architectures, *Proc. Supercomputing '93*, pp.740–749 (1993).
18) 森ほか：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).
19) Mori, S., et al.: An Proposal of Self-Cleanup Cache, KUIS Technical report, No.KUIS-95-

0006, Dep. of Information Science, Kyoto University (1995).
20) Protić, J., Tmašević, M. and Miltinović, V.: Distributed Shared Memory: Concepts and Systems, *IEEE J. of Parallel and Distributed Technology*, pp.63–79, Vol.4, No.2 (1996).
21) Singh, J.P., Weber, W.D. and Gupta, A.: SPLASH: Stanford Parallel Applications for Shared-memory, Tech. Rep. CSL-TR-91-469, Stanford Univ (1991).
22) Sandhu, H.S., Algothitms for Dynamic Software Cache Coherence, *J. of Parallel and Distributed Computing*, pp.142–157, Academic Press (1995).
23) Skeppstedt, J. and Stenström, P.: A Compiler Algothitm that Reduces Read Latency in Ownership-based Cache Coherence Protocols, *Proc. Parallel Architectures and Compilation Techniques*, pp.69–78 (1995).
24) 富田眞治：キャッシュ・コヒーレンス，並列コンピュータ工学 5.3 節，pp.137–159, 昭晃堂 (1996).
25) Lebeck, A.R. and Wood, D.A.: Dynamic Self-invalidation: Reducing Coherence Overhead in Shared-memory Multiprocessors, *Proc. Int'l Symp. on Computer Architecture*, pp.48–59 (1995).
26) Alpha Architecture Handbook, Digital Equipment Co. (1992).
27) SuperSPARC User's Guide, Texas Instruments (1992).

(平成 7 年 9 月 5 日受付)
(平成 8 年 12 月 5 日採録)

**森 眞一郎**（正会員）

1963 年生．1987 年熊本大学工学部電子工学科卒業．1989 年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．1992 年同大学大学院総合理工学研究科情報システム学専攻博士課程単位取得退学．同年京都大学工学部助手．1995 年同助教授．工学博士．並列/分散処理．計算機アーキテクチャの研究に従事．IEEE-CS, ACM 各会員．

**福島 直人**（学生会員）

1972 年生．1995 年京都大学工学部情報工学科卒業．同年，同大学大学院工学研究科修士課程情報工学専攻に進学．現在に至る．並列計算機アーキテクチャの研究に従事．

**五島 正裕**（正会員）

1968 年生．1994 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年より日本学術振興会特別研究員．並列計算機アーキテクチャの研究に従事．1996 年同大学大学院工学研究科情報工学専攻博士後期課程退学．同年より同大学工学部助手．

**中島 浩**（正会員）

1956 年生．1971 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年三菱電機（株）入社．推論マシンの研究開発に従事．1992 年より京都大学工学部助教授．並列計算機のアーキテクチャ，プログラミング言語の実装方式に関する研究に従事．工学博士．1988 年元岡賞，1993 年 坂井記念特別賞受賞．

**富田 眞治**（正会員）

1945 年生．1973 年京都大学大学院博士課程修了，工学博士．同年同大学工学部情報工学教室助手．1978 年同助教授．1986 年九州大学大学院総合理工学研究科教授．1991 年京都大学工学部情報工学科教授．計算機アーキテクチャ，並列計算機システムに興味を持つ．著書「並列コンピュータ工学」「並列処理マシン」「コンピュータアーキテクチャI」など．電子情報通信学会，IEEE, ACM 各会員．本会理事．