# 同期操作に対する投機的メモリ・アクセス機構 specMEMの改良

松尾治幸[†,☆] 大野和彦[†] 中島浩[†]

共有メモリ型並列計算機における同期処理オーバヘッドを削減する手法として，我々は同期操作に後続するメモリアクセスを同期成立確認以前に実行する機構specMEMを提案してきた．この機構の特徴は，投機失敗の検出やそれに伴う計算状態の復元を，機能メモリを用いたコヒーレント・キャッシュの簡単な拡張により実現することにある．これまでの評価では，負荷の変動によって同期区間が伸縮するようなプログラムに対してspecMEMが有効であることが確かめられている．しかし同時に，投機によりキャッシュ・ミスペナルティが増加し，プログラムによっては性能が低下してしまうことも明らかになっている．そこで本報告ではspecMEMの改良方式として，投機的更新を示す新たな状態の追加と，通常のメモリで構成される2次キャッシュの導入を提案する．SPLASH-2ベンチマークを用いた評価を行った結果，Radix Sortでの性能劣化を8.6%から0.7%まで削減できることや，LU分解の性能向上率が14%から16%に増加することが明らかになった．

# Improvement of the Speculative Memory Access Mechanism: specMEM

Haruyuki Matsuo,[†,☆] Kazuhiko Ohno[†] and Hiroshi Nakashima[†]

In order to reduce the overhead of synchronizing operations of shared memory multiprocessors, we have proposed a mechanism named specMEM to excute memory accesses following a synchronizing operation speculatively before the completion of the synchronization is confirmed. A unique feature of our mechanism is that the detection of speculation failure and the restoration of computational state on the failure are implemented by a small extension of coherent cache with a simple functional memory. We showed that specMEM is effective to programs in which computational loads flactuate. We also observed, however, that the speculation increases cache miss penalty not only limitting the efficiency of the specMEM but also degrading the performance of programs with load concentration. In this paper, we propose two techniques to reduce the cache miss penalty; adding one more cache state for speculation; and attaching a secondary cache using non-functional ordinary memory. The evaluation result with SPLASH-2 shows that the performance degradation of Radix Sort is reduced from 8% to 0.7%. It is also shown that the speed-up of LU decomposition is improved from 14% to 16%.

## 1. はじめに

共有メモリ型並列計算機におけるプロセッサ間通信は，共有メモリへのアクセスと同期操作の組合せによって実現される．すなわち異なるプロセッサによる共有変数のアクセスと，プログラムが要求する依存関係を充足するようにアクセスを順序付ける同期操作によって，通信が実現される．したがってある同期操作を開始すると，その同期が成立したことが確認されるまで，共有変数へのアクセスを行うことはできない．しかし依存関係が満たされているなら，同期成立以前に共有メモリへのアクセスを開始しても問題はなく，より高速にプログラムが実行できる．ただし処理の正当性を保証するためには同期操作が必要であり，また同期操作によるアクセス禁止の対象や期間を必要最小限にとどめることは困難であるため，不必要なオーバヘッドが生じてしまう．

そこで我々はこのオーバヘッドを削減する方法として，同期操作後のメモリ・アクセスをデータ依存制約が満たされていると仮定して投機的に実行する機構specMEMを提案している[15),16)]．specMEMはコ

† 豊橋技術科学大学
Toyohashi University of Technology
☆ 現在，株式会社富士通プライムソフトテクノロジ
Presently with Fujitsu Prime Software Technologies Ltd.

ヒーレント・キャッシュに簡単な拡張を施すことにより実現でき，機能メモリを用いることによって投機の開始，成功，失敗に伴う操作を定数時間で実行できるという特徴を持っている．

これまでの評価の結果，specMEMは負荷の変動によって同期区間が伸縮するようなプログラムに対して有効であることが分かっている．しかし同時に，投機によってキャッシュミス・ペナルティが増加し，プログラムによっては性能を悪化させることも判明している．また大容量の機能メモリを実装することが困難であることから，2次キャッシュに対してspecMEMを適用しにくいという問題もあった．

そこで本論文では，キャッシュミス・ペナルティを削減するために，投機的に更新されたラインに対して新たな状態を割り当てる方式を提案し[9),12)]，その効果を評価する．またこの新方式をベースとして，通常のメモリで2次キャッシュを構成する方式と，その性能についても議論する．

以下，2章でspecMEMの概要を述べた後，3章で上記の2つの改良方式を詳細に示し，4章でSPLASH-2に含まれるいくつかのベンチマークによる評価結果とそれに対する考察を述べる．

## 2. specMEMの概要

### 2.1 投機による高速化

共有メモリ型並列計算機における同期操作では，いったん操作が開始されると同期が成立するまで同期に関連する共有変数へのアクセスを行うことはできない．また実装の簡便さを保つために，多くの場合はより広い範囲の操作，たとえばあらゆるメモリ・アクセスが同期成立まで禁止される．

この同期操作は一般に通常のメモリ・アクセスよりも時間を要するため，同期操作の遅延隠蔽や削減は共有メモリのアーキテクチャとプログラミングの双方にとって重要な技術的課題である．このための自然なアプローチとして考えられるのは，同期操作の実行頻度を削減するために通信の粒度を大きくすることである．この方法は，分散メモリマシンにおけるメッセージの集合化などの技術と共通する発想に基づいており，同様の効果をもたらすように見える．

しかし通信の粗粒度化は，細粒度通信という共有メモリの特質とは整合しないため，有効な方法であるとは限らない．すなわち同期成立確認までの操作禁止によって，不必要なオーバヘッドが生じることがしばしばある．たとえば**図1**はバリア同期を用いたプロセッサ $P_1$ と $P_2$ の間の通信を示したものである

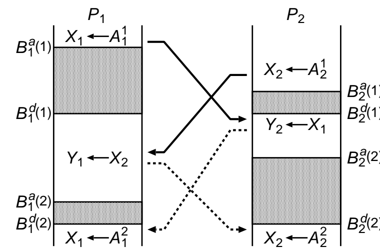



**図1** バリア同期によるデータ依存制約の充足
Fig. 1 Satisfying data dependency constraint by barrier synchronization.

が，$X_1$ のフロー依存制約（図中の実線矢印）は $P_2$ よりも $P_1$ が最初のバリアに先に到達しているので（$B_1^a(1) < B_2^a(1)$）で，明らかに充足されている．したがって $P_2$ がバリア同期成立を確認するために要する時間 $B_2^d(1) - B_2^a(1)$ は，無駄なアイドル時間であるといえる（図中の暗い影付の部分）．また $X_2$ のフロー依存制約は，$P_1$ がバリアへ到達した時点 $B_1^a(1)$ では満たされていないが，同期成立確認時点 $B_1^d(1)$ よりも以前に満たされているので，やはり無駄なアイドル時間が生じている．さらに同様の現象は，$X_1$ と $X_2$ の逆依存制約を満たすための2番目のバリア同期についても見ることができる．

このような無駄なアイドル時間は，フロー依存や逆依存などのデータ依存制約を，同期という一種の制御依存制約に置き換えて充足しようとするために生じたものであると考えることができる．そこで我々は，制御依存による遅延を除去する方法として一般的に用いられている投機的実行[11),19)]を応用することにより，無駄なアイドル時間を除去あるいは削減する機構**specMEM**を提案している．

specMEMでは，同期操作を実行してもプロセッサは停止せず，その完了が確認されるまでの間は投機モードに移行して処理を続行する．したがって，共有変数へのアクセスを含むすべてのメモリ・アクセスは，同期操作により充足されるべきデータ依存制約がすでに満たされているものと仮定して，投機的に実行される．

たとえば**図2**に示す例では，共有変数 $X_i$ $(i = 1, 2)$ のフロー依存制約を充足するためのバリア同期にプロセッサ $P_1$ が到達すると（$B_1^a(1)$），$P_1$ は投機モードに移行して処理を続行する（図中の明るい影付部分）．その結果，同期成立確認が $B_1^d(1)$ でなされる以前に $P_1$ は $X_2$ を読み出す．しかし $X_2$ の書き込み/読み出しのタイミングが図に示すようにフロー依存制約を満たす場合，この読み出しによって得られる値は正し

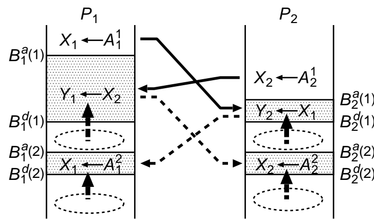



**図 2** 投機的メモリ・アクセス
Fig. 2 Speculation of shared memory accesses.

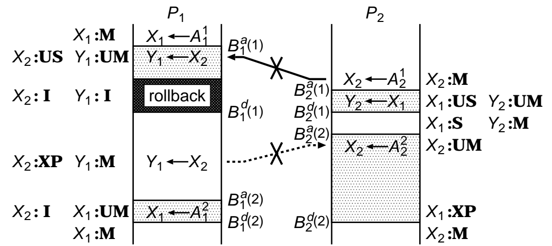



**図 3** 投機の失敗
Fig. 3 Speculation failure.

く，投機は成功する．また $B_2^a(1)$ と $B_2^d(1)$ の間に行われる $P_2$ による $X_1$ の読み出しについても同じように投機は成功し，この結果どちらのプロセッサについてもアイドル時間が完全に除去される．

また同様に，$X_1$ と $X_2$ の逆依存制約を充足するためのバリア同期に $P_1$ と $P_2$ が $B_1^a(2)$ と $B_2^a(2)$ で到達後，これらへの書き込みが $B_1^d(2)$ と $B_2^d(2)$ での完了確認以前に投機的に行われる．この場合も読み出し / 書き込みの順序が逆依存制約を満たしているので，やはり投機は成功する．

この例に示すように，投機的アクセスは同期成立確認に要する時間を削減するものであるので，バリア到達時刻にずれが生じる場合，特に負荷変動などによって到達順序が変化する場合に有効である．また負荷が均衡している場合も，多数のプロセッサがバリアに参加することにより確認のための遅延が大きくなれば，遅延隠蔽の効果が顕著に現れる．

### 2.2 投機の失敗

前節の例では充足すべきデータ依存制約が，投機的に行われたすべてのアクセスについて満たされたため，プログラムの意味を保存しつつオーバヘッドを除去することができた．しかし投機である以上，依存制約を満たさないアクセスが行われる可能性は常にあり，その場合にもプログラムの意味が保存されるような措置が必要である．

たとえば**図 3** に示す例では，$P_1$ による $X_2$ の投機的読み出しが $P_2$ による $X_2$ への書き込みに先行し，その結果不正な値を読み出してしまっている．またこの不正な値は $Y_1$ に格納されるので，$Y_1$ を参照する操作があれば不正値が次々に伝搬する．このような場合，まず不正な投機的読み出しを行ってしまったことを検知し，続いて不正値によって生じたあらゆる計算状態変化を無効化し，正しい値による計算を再度行う必要がある．

specMEM ではこの投機失敗の検知と無効化をライトバック型のコヒーレント・キャッシュを拡張して実現している．まず，投機モードでアクセスされたキャッシュ・ラインすべてに，潜在的に危険であることを示すマークが付けられる．上記の例では，$X_2$ の読み出し前の状態が **S**（Shared）であるとすると，投機的読み出しの結果 **US**（Unsafe Shared）という特別な状態に遷移する．また $Y_1$ の状態も，**M**（Modified）に対応する状態 **UM** となる．この **U** が付された状態にあるラインに対する他のプロセッサからの書き込み通知を受け取ると，不正値を読み出していた可能性があることが判明する．すなわち上記の例では，$P_2$ による $X_2$ への書き込み通知によって，$P_1$ のキャッシュが不正読み出しを検知する．

次に行う計算状態変化の無効化（ロールバック）は，キャッシュのライトバック機構を利用して行う．すなわち，ラインの状態が **M** から **UM** または **US** に変化する際に，ラインの値をメモリに書き戻すことにより，投機開始時点の値がメモリに保存されるようにする．投機失敗が検知されると **U** が付されたラインはすべて無効化され，以後の操作では保存された真値が参照されるようにする．

この結果，メモリに関する計算状態変化はすべて無効化されるので，他の計算状態をシャドーレジスタ[17)]などの機構を用いて投機開始時点で保存しておけば，その復元によりロールバックを行うことができる．ロールバック後の実行再開は，複数のプロセッサが互いを投機的にロールバックさせてデッドロックに陥ることを防止するために，図 3 に示すように同期成立が確認されるまで抑止される．

この計算の無効化と再実行を，動的命令スケジューリングを行うプロセッサに用いられるロード / ストア・バッファに類似した機構を用いて実現することは可能である[3),4),18),20)]．すなわち，潜在的に不正なロード命令のアドレスを連想バッファに記憶して他のプロセッサからの更新通知と比較し，かつ計算状態を保存するためにストア命令のアドレスとデータ（あるいはそのアドレスの旧値）を別の連想バッファに記憶すればよい．しかしこの方法では連想バッファに要するハード

ウェア・コストの制約から，バッファ容量が必然的に小さい値となり，同期成立確認までの大きな遅延を隠蔽するには不十分であることが予想される．また投機が成功したことが確認されたとき（あるいは失敗が検出されたとき），ストア・バッファに記憶しておいたすべての書き込み操作をバースト的に実行する必要があり，投機成功に関する手間が投機的書き込みの数に比例することとなる．

一方 specMEM ではキャッシュ容量分だけ投機的アクセスを許容できるので，遅延を隠蔽するには十分である．また specMEM にはキャッシュラインを一括遷移する機能が必要になるが，後述のマスク付の一括リセットができるような簡単な機能メモリを用いることによって，投機失敗時の一括無効化を定数時間で実現できる．さらにこの機能により投機の成功時，すなわち同期成立が確認された時点で，**U** が付されたラインをすべて普通の状態に戻す操作も定数時間で実現できる．

### 2.3 共有変数の投機的書き込み

図 3 の例では，$P_1$ でのロールバックの後，$X_2$ の値が再び読み出される．一方 $P_2$ では最初のバリアに関する投機は成功し，続いて 2 番目のバリアに関する投機的アクセス，すなわち $X_2$ への書き込みを行う．この書き込みは図に示すように $P_1$ による $X_1$ の読み出しに先行してしまっているので，逆依存制約を満たしていない．

この不正な書き込みを行ってしまったことは，無効化型のプロトコルを用いれば，前節と同様に **UM** 状態のラインへの他プロセッサからの読み出し要求によって検知できる．したがって前節と同様にロールバックすることもできるが，**UM** 状態のラインについては書き込み前の値がメモリに保存されていることを利用して，正しい値を参照元プロセッサに返すことができる．すなわち図の例では，$P_2$ のキャッシュにある書き込み後の値 $A_2^2$ ではなく，$B_2^a(2)$ の時点での値である $A_2^1$ をメモリから $P_1$ へ返すことが可能である．

しかしこの値は，2 番目のバリア同期成立確認までに読み出される限りは正しいが，それ以降は逆に不正に古い値となってしまう．すなわち（図には示されていない）3 番目のバリア以降では，$X_2$ の値は $A_2^2$ でなければならない．しかし $P_2$ は $X_2$ への書き込みの時点で書き込み通知を送ってしまっているので，$P_1$ はキャッシュした値 $A_2^1$ を無効化（あるいは更新）する機会を失っている．

そこで，他のプロセッサのキャッシュに **UM** 状態のラインが存在するためにメモリから値を得たラインについては，**XP**（eXPiring）という特別な状態を割り当て，次のバリア同期に到達した際に（図では $B_1^a(2)$）一括して無効化する．この結果，$P_1$ が $X_2$ を改めて読み出す際にはキャッシュミスとなり，$P_2$ のキャッシュから正しい値である $A_2^2$ を得ることができる．なおこの一括無効化は，前述の投機成功/失敗時の一括状態変更と同様に，簡単な機能メモリを用いて定数時間で行うことができる．

また図の例では，$P_2$ が $X_2$ に書き込む時点で $P_1$ は $X_2$ をキャッシュしていないが☆，投機的書き込みの対象が他のキャッシュに存在することもある．その場合には，書き込み通知に投機的であることを示す情報を付加し，他のキャッシュに存在するコピーの状態を直接 **XP** に遷移させる．これにより，投機的書き込みによる無駄な無効化に伴うキャッシュ・ミスを防止できる☆☆．

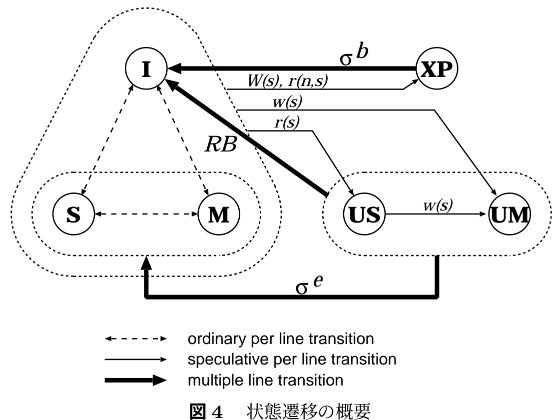



**図 4**　状態遷移の概要

Fig. 4　Overview of state transition.

### 2.4 キャッシュの状態遷移

前節で述べたように，specMEM はライトバック型のコヒーレント・キャッシュを拡張する形で実装される．すなわちキャッシュのベースとなる状態を **M**，**S**，および **I**（Invalid）としたとき，投機的アクセスに関する特別な状態 **UM**，**US**，**XP** が加えられ，以下のような状態遷移が行われる（**図 4**）．

(1) 通常の状態 {**M**,**S**,**I**} にあるラインに対して投機モードでの読み出し（$r(s)$）を行うと，投機的読み出しを記憶するためにラインは **US** へ遷移する．また元の状態が **M** であれば，ライトバックにより投機失敗に備えてラインの状態をメモリに保存する．

☆ ロールバックの際に無効化されている．

☆☆ write-invalidate の場合．write-update であれば不正な更新が防止される．

同様に投機モードでの書き込み（$w(s)$）では **UM** に遷移し，旧状態が **M** であればライトバックを行う．またこの投機書き込みを通知された他のキャッシュでは（$W(s)$），保持している値がいずれ無効になることを示す状態 **XP** に遷移する．その際，旧状態が **M** であればやはりライトバックを行う．この **XP** への遷移は，キャッシュミスにより **UM** 状態のラインを得ようとした場合にも生じ（$r(n, s)$），その際にはメモリに保存された投機開始前の値が返される．

（2）　同期の成立確認によって投機が成功すると（$\sigma^e$），**US** のラインはすべて **S** に，また **UM** のラインはすべて **M** に遷移する．この結果投機的状態は解消し，次の投機開始までは通常の状態遷移を行う．

（3）　一方，**US** または **UM** のラインに対する他プロセッサの書き込み，これらのラインのリプレース，あるいは **XP** のラインへの自プロセッサからの投機的アクセスが生じると，プロセッサは投機開始時点までロールバックする（$RB$）．同時にすべての **US** と **UM** のラインは **I** に遷移し，メモリに保存した値を参照できるようにする．

（4）　次の同期点に達すると（$\sigma^b$），**XP** のラインの値はすでに古いものとなっている可能性があるため，すべての **XP** のラインが **I** に遷移し最新の値を参照できるようにする．

### 2.5　機能メモリによる実現

以上述べた状態遷移のうち，$\sigma^b$，$\sigma^e$，$RB$ により引き起こされるものは，複数のキャッシュラインに対して行われる．この一括状態遷移は，以下の機能を備えた簡単な機能メモリを用いることにより実現できる．

（1）　すべてのワードについて，あるビット $b_r$ を 0 にする機能：$reset(b_r)$

（2）　すべてのワードについて，あるビット $b_m$ が 1 であれば別のビット $b_r$ を 0 にする機能：$masked_reset(b_m, b_r)$

すなわちこれらの機能を用い，**表 1** に示すように状態をエンコードすることにより，以下の操作によって一括状態遷移を行うことができる．

$$\begin{aligned} \sigma^b &: reset(b_2); \\ \sigma^e &: reset(b_2); \\ RB &: masked_reset(b_2, b_1); \\ &\quad masked_reset(b_2, b_0);\ reset(b_2); \end{aligned}$$

このような機能を実現するためにメモリセルに必要なトランジスタ数はワードあたり 1 ビット分の追加ですみ[15),16)]，ハードウェアの実装コストは許容範囲であるといえる．またリセット操作に要する時間を通常アクセスよりも長くしても性能に大きなダメージは与えないため，許容される電力消費量に見合った設計も可能である．

**表 1**　状態のエンコード
Table 1　Encoding of cache states.

| state | $b_2b_1b_0$ | $\sigma^b$ | $\sigma^e$ | $RB$ |
|---|---|---|---|---|
| **I** | 000 | **I** (000) | **I** (000) | **I** (000) |
| **S** | 001 | **S** (001) | **S** (001) | **S** (001) |
| **M** | 010 | **M** (010) | **M** (010) | **M** (010) |
| **XP** | 100 | **I** (000) | **I** (000) | **I** (000) |
| **US** | 101 | — | **S** (001) | **I** (000) |
| **UM** | 110 | — | **M** (010) | **I** (000) |

## 3.　投機的実行の改良

### 3.1　投機的実行の問題点

前章で述べた基本的な実装方式では，ロールバックが生じたときに投機状態を示すビットがオンであるラインをすべて無効化しなければならない．そのためロールバック後の再実行時のミス率が大きな値となるが，3.2 節で説明するように実際には投機的読み出しだけが行われたライン（前章の例では **US** のライン）の無効化は不必要であるにもかかわらず，投機的書き込みが行われたライン（前章の例では **UM** のライン）とともに無効化されてしまう．

また，ライトバックを伴うような一括無効化は困難であるため，dirty なラインを投機的に読み出した際に clean にする必要がある．前章の例では **M** のラインの投機的読み出しの際に，**US** に遷移すると同時に状態保存のためにメモリへライトバックを行う必要があり，そのためバストラフィックが増加してしまう．

これらの問題点は，ロールバックあるいは投機的アクセスを行うプロセッサの実行を妨げるだけではなく，バスやメモリのトラフィックを増やすことによって間接的に他のプロセッサの実行も妨げてしまう．たとえば SPLASH-2[21)] 中の Radix Sort では，クリティカル・パスを実行するプロセッサのキャッシュミス・ペナルティが，他のプロセッサの投機およびその失敗により顕著に増加し，かえって性能が低下してしまうことが明らかになっている．

### 3.2　キャッシュの状態遷移の改良

前述のように投機的読み出しだけが行われたラインの値は，ロールバック時に無効化する必要はない．すなわち，このようなラインは以下のいずれかの状態にあるため，ロールバック後の再実行時には必ず正しい

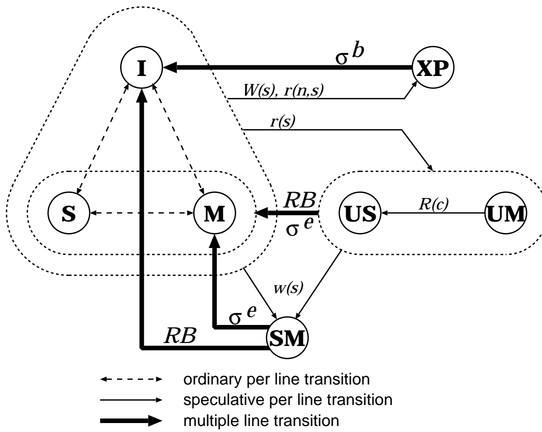



**図 5** 改良後の状態遷移
Fig. 5 Modified state transition.

値を得ることができる．

（1） 投機的読み出しが依存制約を満たすようなタイミングで行われた場合，キャッシュは当然正しい値を保持している．

（2） 投機的読み出しが依存制約を満たさないタイミングで行われ，他の投機的アクセスが原因でロールバックした場合，ロールバック時点ではキャッシュの値は不正である．しかし再実行は同期成立確認後に行うため，再実行開始時点では読み出しに先行すべき書き込みが完了しており，その書き込みによってキャッシュは正しく無効化されている．

そこで投機的書き込みが行われたことを意味する新たな投機的状態として **SM**（Speculatively Modified）を導入し，投機的読み出しだけが行われたラインと明確に区別できるようにする．この結果，キャッシュの状態遷移は**図 5** のようになり，基本的な実装（図 4）から以下のように改善される．

（1） 状態 **M** のラインは投機的読み出し（$r(s)$）により（**US** ではなく）**UM** へ遷移する．この状態は dirty で排他的，かつ潜在的に危険ではあるが値自身は正しいことを意味する．したがって旧値をライトバックする必要はなく，バスやメモリのトラフィックを増加させることもない．なお **UM** のラインに対して他のプロセッサからの読み出し要求があると（$R(c)$），ラインの値を返して **US** へ遷移する．

（2） 投機的書き込み（$w(s)$）が行われると，すべての状態から **SM** へ遷移する．元の状態が dirty であれば（すなわち **M** または **UM**），状態保存のために旧値をメモリへライトバックする．

**表 2** 改良後の状態のエンコード
Table 2 Encoding of modified cache states.

| state | $b_3b_2b_1b_0$ | $\sigma^b$ | $\sigma^e$ | $RB$ |
|---|---|---|---|---|
| **I** | 0000 | **I** (0000) | **I** (0000) | **I** (0000) |
| **S** | 0001 | **S** (0001) | **S** (0001) | **S** (0001) |
| **M** | 0010 | **M** (0010) | **M** (0010) | **M** (0010) |
| **XP** | 0100 | **I** (0000) | **I** (0000) | **I** (0000) |
| **US** | 0101 | – | **S** (0001) | **S** (0001) |
| **UM** | 0110 | – | **M** (0010) | **M** (0010) |
| **SM** | 1110 | – | **M** (0010) | **I** (0000) |

（3） 投機が成功すると（$\sigma^e$），**U** マークがすべて除去され，同時に **SM** のラインはすべて **M** へ遷移する．

（4） ロールバックが生じた場合（$RB$），**SM** のラインはすべて無効化される．しかし **UM** や **US** のラインは無効化されず，単に **U** マークが除去されて元の **M** または **S** に遷移する．したがって正しいタイミングで投機的参照が行われたラインは，再実行時にヒットする．

なおロールバック時に **SM** のラインだけを一括無効化するために，2.5 節で述べた機能メモリの状態遷移は**表 2** のように拡張され，それに伴い一括状態遷移の操作は以下のようになる．

$$\sigma^b : reset(b_2);$$
$$\sigma^e : reset(b_3);\ reset(b_2);$$
$$RB : masked_reset(b_3, b_1);\ reset(b_3);\ reset(b_2);$$

### 3.3 2 次キャッシュの導入

投機的アクセスを行うことにより，メモリ・アクセスのタイミングや回数が変化し，その結果としてキャッシュミス・ペナルティが増加する．増加要因の 1 つはキャッシュミス回数自体の増加であり，もう 1 つは投機的書き込み時の状態保存のためのライトバックである．

そこで specMEM においても一般の共有メモリ・システムと同様に，2 次キャッシュを導入すればキャッシュミス・ペナルティの影響が少なくなり，性能が向上することが期待できる．また投機的書き込みの際の状態保存先をメモリではなく 2 次キャッシュとすることにより，バスやメモリのトラフィック増加を抑える効果も期待できる．

しかし 2 次キャッシュは容量が大きいため，複数ラインの一括状態遷移を行うための機能メモリの使用は困難であり，特にオフチップの 2 次キャッシュでは現実的ではない．したがって specMEM の 2 次キャッシュは，一括状態遷移機能を用いずに実装できなければならない．

そこで 2 次キャッシュについては，ベースとなる状

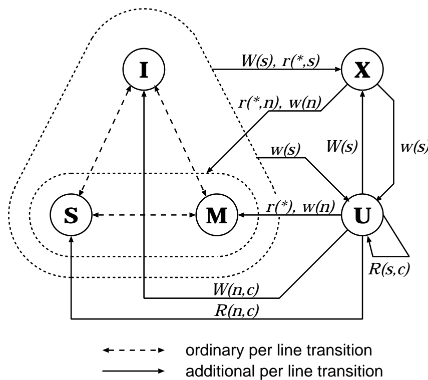



**図 6**　2 次キャッシュの状態遷移
Fig. 6　State transition of secondary cache.

態に以下の 2 状態だけを追加し，**図 6** に示すような状態遷移を行う．

**U**　対応する 1 次キャッシュのラインが **SM** である可能性があることを示す．投機的書き込み（$w(s)$）は 2 次キャッシュに必ず伝達され，対応するラインは **U** へ遷移し，必要に応じてラインの旧値が 2 次キャッシュへライトバックされる．

一方投機の成功/失敗時には，1 次キャッシュでは **SM** からの一括状態遷移が行われるが，2 次キャッシュは一括状態遷移ができないため対応するラインは **U** にとどまる．したがって **U** のラインに対応する 1 次キャッシュのラインは，**SM**（投機中），**M**（投機成功）または **I**（投機失敗）のいずれかとなる．そこで **U** のラインに関しては以下のような状態遷移を行う．

- プロセッサからの読み出し（$r(*)$）あるいは非投機的書き込み（$w(n)$）が 2 次キャッシュに伝達された場合，1 次キャッシュの対応するラインは必ず **I** である．このとき 2 次キャッシュは正しい値を保持しており，かつ他のキャッシュの状態は **I** または **XP** である．そこで **U** のラインは **M** へ遷移し，（**XP** のラインを除いて）排他的かつ dirty な状態とする．
- 他のプロセッサからの読み出し要求（$R(n, c)$，$R(s, c)$）を受理すると，1 次キャッシュの対応するラインの状態を調べ，**M** であれば 1 次キャッシュの値を，そうでなければ 2 次キャッシュの値を返す．また 1 次キャッシュの状態が **SM** であれば（$R(s, c)$）要求元にその旨を伝えて（要求元は **XP** へ遷移）**U** にとどまり，そうでなければ **S** へ遷移する．
- 他のプロセッサからの非投機的書き込み要求（$W(n, c)$）は 1 次キャッシュに伝達され，対応するラインが **SM** であればロールバックを生じる．また **U** のラインは（**XP** のラインを除いて）排他的であるので，読み出し要求と同様に 1 次キャッシュの状態に応じて 1 次または 2 次キャッシュの値を返し，**I** に遷移する．

**X**　対応する 1 次キャッシュのラインが **XP** である可能性があることを示す．1 次キャッシュが **XP** に遷移する際に（$W(s)$，$r(*, s)$），2 次キャッシュでは **X** へ遷移する．

一方投機の成功時には☆，1 次キャッシュでは **XP** の一括無効化が行われるが，2 次キャッシュは一括状態遷移ができないため対応するラインは **X** にとどまる．したがって **X** のラインに対応する 1 次キャッシュのラインは，**XP**（投機開始前）または **I**（投機開始後）のいずれかとなるが，自プロセッサおよび他プロセッサのどちらからアクセスされても **I** と見なすことができる．したがって **X** と **I** の差異は，他プロセッサからの書き込み要求を 1 次キャッシュに伝達することだけである．

なお 2 次キャッシュには投機的読み出しが行われたことを示す状態はなく，不正読み出しの検出は 1 次キャッシュでのみ行われる．したがって **U** マークが付されたラインが 1 次キャッシュでリプレースされると，2 次キャッシュがない場合と同様にロールバックが生じる．

## 4. 評　　価

### 4.1　評価の方法

評価のために，**表 3** に示す仕様に基づく集中共有メモリ型マルチプロセッサのシミュレータを構築した．なお前章で述べたキャッシュ状態は MSI をベースとしたものであるが，評価に用いたキャッシュは MESI をベースとしたものである．ただし，状態 **E**（Exclusive）とそれに対応する **UE** の導入に伴う投機的状態の追加はほぼ自明であり，キャッシュタグに要するハードウェア・コストはまったく増加しない[12]．

またシミュレータの単純化と結果の解析を容易にするために，プロセッサはサイクルあたり 1 命令発行かつ in-order 実行とし，メモリ・モデルは weak consistency[1]とした．バリア同期については単純な線バリア[13]をサポートするハードウェア機構の存在を想定し，バリア同期命令によってこのバリア機構とキャッシュにバリア到達が通知されるとともに，レジスタな

☆ 実際には失敗と開始を含む．

**表 3** 評価に用いたマルチプロセッサモデル
Table 3 Architectural parameters for evaluation.

| プロセッサ数 | 4 |
|---|---|
| 1 次キャッシュ | |
| 容量 | 64 KB |
| ラインサイズ | 32 B |
| 連想度 | ダイレクトマップ |
| キャッシュコヒーレンス | MESI |
| 2 次キャッシュ | |
| 容量 | 512 KB |
| ラインサイズ | 32 B |
| 連想度 | ダイレクトマップ |
| キャッシュコヒーレンス | MESI |
| 命令実行コスト（サイクル数） | |
| 一般命令 | 1 |
| バリア同期 | 1 + 0 |
| 2 次キャッシュアクセス | +2 |
| バスアクセス | +5 |
| メモリアクセス | +10 |

どの状態保存が行われるものとした．また最後のプロセッサがバリアに到達すると，表 3 に示すバリア同期コストを経過した後にすべてのプロセッサとキャッシュに同期成立が伝達されるものとした．

なおこのように小規模の集中共有メモリのシステムを評価対象としたのは，単に性能評価を迅速に行うためであり，specMEM の適用対象が小規模なシステムに限定されることを意味するものではない．これまで説明したとおり specMEM はキャッシュを使って投機処理を行っているので，旧値のライトバックを除き☆，ローカルな処理だけで実行される．また状態 SM の導入によりライトバック処理が削減され，さらに 2 次キャッシュを利用するシステムでは投機的書き込み時の旧値の保存は 2 次キャッシュへ行われるのでスケーラビリティが向上し，より大規模なシステムでも効果が期待できる．

また本論文での評価結果では，投機状態であるキャッシュラインの追い出しによるロールバックは発生しておらず，投機処理に関する限り容量，ラインサイズ，連想度などのパラメータは妥当なものであると考えられる．

評価のためのワークロードとしては，SPLASH-2 [21] の中から以下の 3 つのプログラムを選択し，投機的アクセスの有無による実行時間（サイクル数）を計測した．

- LU 分解
  負荷の偏りがあり，かつ高負荷のプロセッサが変動するため，投機的アクセスの効果が高い．

☆ ロールバックに備えて投機前の値はメモリに退避しておく．

**表 4** キャッシュミスペナルティ
Table 4 Cache miss penalty.

| program | NS1 | S1 | S1+ | NS2 | S2 |
|---|---|---|---|---|---|
| LU 分解 | 100.0 | 102.8 | 101.6 | 28.9 | 34.9 |
| FFT | 100.0 | 100.0 | 83.0 | 73.5 | 54.0 |
| Radix sort | 100.0 | 133.6 | 114.9 | 91.6 | 89.1 |

- FFT
  負荷の偏りがまったくないため，投機的アクセスはほとんど行われない．したがって specMEM が性能に対して「中立的」であることの試験となる．
- Radix Sort
  負荷が特定のプロセッサに偏るため，他のプロセッサによる投機的アクセスが行われるが，高負荷プロセッサの実行時間は短縮されない．したがって投機的アクセスの悪影響が現れる．specMEM の中立性に関するより厳しい試験となる．

### 4.2 評価結果

**図 7** は前述の 3 つのプログラムの実行時間（サイクル数）を，また**表 4** は各プロセッサのキャッシュミスペナルティの平均を，それぞれ以下の 5 種類のシステムについて示したものである．ただしいずれも NS1 の値を 100 として正規化している．

- NS1：非投機的かつ 1 次キャッシュのみ．
- S1：改良前の specMEM．1 次キャッシュのみ．
- S1+：**SM** を導入した specMEM．1 次キャッシュのみ．
- NS2：非投機的かつ 2 次キャッシュあり．
- S2：**SM** を導入した specMEM．2 次キャッシュあり．

図 7 の内訳の中で rollback は命令再実行を含むロールバックのコストであり，cache miss には共有ライン書き込みや状態保存ライトバックのペナルティも含む．また図には，投機的アクセスを行った場合のロールバック率も示されている．以下，各々の結果に関する考察を述べる．

#### 4.2.1 LU 分解

改良前の specMEM でも投機的アクセスによってアイドル時間が短縮され，14%程度の速度向上が得られていたが，$P1$ と $P2$ においてロールバックのコスト（命令再実行コストを含む）とキャッシュミス・ペナルティが無視できない値となっていた．これらのうちキャッシュミス（共有ラインの無効化を含む）の回数は **SM** の導入によって 37%削減されたが，ミスペナルティの削減は 1%にとどまり，実行時間もほとんど改善されなかった．一方 2 次キャッシュの導入により，S2 では S1 に比べてバスアクセス回数が 78%，ミ

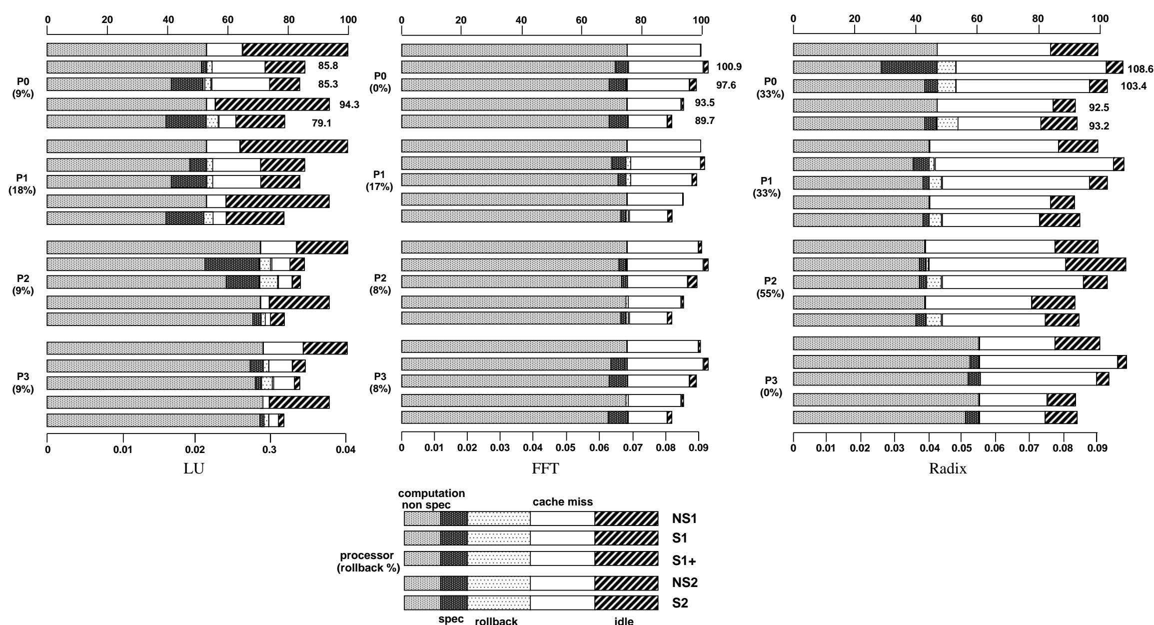



**図 7**　SPLASH-2 による評価結果
Fig. 7　Performance results of SPLASH-2 benchmarks.

スペナルティが 66%と，それぞれ大幅に削減された．その結果，投機的実行の効果が相対的に大きくなり，NS1 に対する S1 や S1+の性能向上率が 14%であるのに対し，NS2 に対する S2 の性能向上率はそれを上回る 16%となった．

### 4.2.2　FFT

負荷がほぼ完全に均衡しているため，投機的アクセスの効果が小さいことが予想される．すなわちこのような場合，投機的アクセスにより隠蔽できるのはバリア同期自体に要するコストであるが，1 回あたりのコストが 10 サイクルという小さい値としており，またバリア同期の実行回数もわずか 12 回であるので，隠蔽効果は 1%未満にすぎない．そのため，改良前の specMEM ではロールバックのコストの影響で，1%の性能劣化となっていた．しかし **SM** や 2 次キャッシュの導入によって，NS1 に対して S1+は 2.4%，NS2 に対して S2 は 4.1%，それぞれ性能が向上することが明らかになった．

このように負荷が均衡しているプログラムに対しても投機実行の効果が得られるのは，以下の 2 つの理由からである．第 1 の理由は，specMEM ではバリア同期の際に先行するメモリアクセスの完了を待つ必要がなく[12]，いわゆるメモリバリア操作の遅延が隠蔽されることにある．実際，NS1 のミスペナルティの 8%を占めるメモリバリアの遅延は S1+ではほとんどない．第 2 の理由は，バリア同期の直後にすべてのプロセッサがほぼ同時にキャッシュミスを起こすという，バスやメモリの一時的な輻輳が投機により回避されることである．すなわち specMEM ではバリアの通過タイミングがプロセッサごとに異なるため，キャッシュミスが集中的に生じる確率が低くなっている．

これらの理由により，S1+のミスペナルティは NS1 よりも 17%小さく，また S2 では NS2 よりも 27%小さくなり，この効果がロールバックの悪影響を上回って性能を向上させている．

### 4.2.3　Radix Sort

二分木による一種のリダクション操作を行うため，そのルートである $P3$ が常に性能上問題となるバリアの最終到達者となる．したがって，FFT と同様に投機的アクセスによって隠蔽できるコストは小さい．さらに，バリア同期がデータだけではなくアドレスに関する依存関係の保証に用いられるため，投機的実行により本来アクセスされないアドレスへの書き込みが生じ，不必要な無効化が行われる．その結果，キャッシュミス・ペナルティの増加，特にクリティカル・パスを実行する $P3$ での増加が顕著となり，S1 は NS1 に対して 8.6%の速度低下が生じている．

この問題は，SM の導入により全体的なミスペナルティを 14%削減することによりある程度改善され，S1+の NS1 に対する速度低下は 3.4%にとどまっている．またミスペナルティは 2 次キャッシュの導入によりさらに削減され，4.2.2 項で述べた輻輳回避の効果

が投機によるミスの増加を上回り，S2 のミスペナルティは NS2 よりも 2.7%小さくなった．この結果，S2 では投機の悪影響がほとんど生じず，NS2 に対する速度低下は 0.7%ときわめて小さな値となった．

## 5. 関連研究

コヒーレント・キャッシュと投機的実行を関係付けた研究は，適切なコヒーレンス維持操作の選択に関するものと，specMEM のように潜在的には危険なアクセスを投機的に行うものに大別される．前者はさまざまなコヒーレンス維持操作のバリエーションの中から，命令ごとの[6]，あるいはメモリ・ライン（ブロック）ごとの[7),10)]履歴に基づいて最適な操作を予測し，それを投機的に行うことによって遠隔アクセスの遅延を削減しようというものである．

一方後者の例として sequential consistency（SC）[8)]を要求するプログラムにおけるメモリ・アクセス順序を投機的に入れ替える，Gniady らの SC++ [4)]が挙げられる．SC++には，SC に基づくプログラムに潜在する同期操作の成立を仮定すること，投機の失敗を他のプロセッサからの書き込み要求によって検出することなど，specMEM との共通点がいくつか存在する．しかし投機的アクセスの履歴を連想バッファに記憶する点が大きく異なり，2.2 節で述べたように大きな同期遅延を隠蔽することができない．

別の例としては，分岐予測に基づく逐次プログラムの投機的マルチスレッド実行を，集中共有メモリ型マルチプロセッサをベースに実現する Gopal らによる Speculative Versioning Cache（SVC）[5)]がある．SVC は我々の初期の報告[14)]と同時期に提案されたもので，他の研究[3),20)]が SC++と同様の連想バッファを用いているのに対し，キャッシュを利用した投機の成否の検出や投機的書き込みの実現法など，specMEM との共通点が多い．

しかし投機的マルチスレッド実行では多数の細粒度スレッドが並列実行されるため，1 つのメモリ・アドレスに対して各々のキャッシュがそれぞれ異なった値を同時に持ちうることが必要となる．そのため，スレッドに応じて適切なキャッシュを選択する集中的な制御論理が必要であり，プロセッサの数は自ずから限定される．一方 specMEM は粗粒度並列タスクを対象としているので，1 つのアドレスに対する値はたかだか 2 種類であり，先に述べたように大規模な分散共有メモリにも簡単に適用できる．

また，投機的マルチスレッド実行をオンチップ共有メモリマルチプロセッサで実現した Cintra らの MDT-based CMP [2)]では，specMEM と同様に投機的アクセスを行ったラインに対して投機ビットを立てることで依存関係の乱れを検知している．MDT-based CMP でも投機ビットの集中管理が必要であるが，これをオンチッププロセッサの内部用と外部用の 2 種類のテーブルで管理し，階層化によるスケーラビリティの維持を図っている．

一方 specMEM では各プロセッサのキャッシュごとに投機ビットを管理しているので，4.1 節で述べたように投機アクセスをローカルに処理でき，非常にスケーラビリティに優れているといえる．

## 6. おわりに

バリア同期に対する投機的なメモリ・アクセスを行う機構 specMEM はコヒーレント・キャッシュに簡単な拡張を施すことにより実現でき，また投機の開始，成功，失敗に伴う操作を定数時間で実行できるという特徴を持っている．本論文ではキャッシュミス・ペナルティの増加という specMEM の問題点を解決するために，新たな投機的キャッシュ状態の追加と，通常のメモリを用いた 2 次キャッシュの導入を提案し，この 2 つの改良の効果を SPLASH-2 ベンチマークを用いて評価した．その結果，Radix Sort で見られた性能劣化を 8.6%から 0.7%に抑えることができ，また LU 分解や FFT の性能も 2〜5%改善することができた．なかでも Radix Sort のようなプログラムに対しても specMEM が中立的に動作するようになったことは重要であり，投機的アクセスを広い範囲のプログラムに対して適用可能であることを示すことができた．

## 参考文献


1) Adve, S.V. and Hill, M.D.: Weak Ordering—A New Definition, *Proc. 17th Intl. Symp. Computer Architecture*, pp.2–14 (1990).
2) Cintra, M., Martínez, J.F. and Torrellas, J.: Architectural Support for Scalable Speculative Parallelization in Shared-Memory Multiprocessors, *Proc. 27th Intl. Symp. Computer Architecture*, pp.13–24 (2000).
3) Franklin, M. and Sohi, G.S.: ARB: A Hardware Mechanism for Dynamic Reordering Memory References, *IEEE Trans. Comput.*, Vol.45, No.5, pp.552–571 (1996).
4) Gniady, C., Falsafi, B. and Vijaykumar, T.N.: Is SC + ILP = RC?, *Proc. 26th Intl. Symp. Computer Architecture*, pp.162–171 (1999).
5) Gopal, S., Vijaykumar, T., Smith, J.E. and Sohi, G.S.: Speculative Versioning Cache, *Proc.*

*4th Intl. Symp. High-Performance Computer Architecutre*, pp.195–206 (1998).
6) Kaxiras, S. and Goodman, J.R.: Improving CC-NUMA Performance Using Instruction-Based Prediction, *Proc. 5th Intl. Symp. High-Performance Computer Architecture*, pp.161–171 (1999).
7) Lai, A.C. and Falsafi, B.: Memory Sharing Predictor: The Key to a Speculative Coherent DSM, *Proc. 26th Intl. Symp. Computer Architecture*, pp.172–183 (1999).
8) Lamport, L.: How to Make a Multiprocessor Computer that Correctly Execute Multiprocessor Programs, *IEEE Trans. Comput.*, Vol.28, No.9, pp.690–691 (1979).
9) 松尾治幸，大野和彦，中島　浩：同期操作に対する投機的メモリ・アクセス機構 specMEM の改良，並列処理シンポジウム JSPP'01，pp.181–188 (2001).
10) Mukherjee, S.S. and Hill, M.D.: Using Prediction to Accelerate Coherence Protocols, *Proc. 25th Intl. Symp. Computer Architecture*, pp.179–190 (1998).
11) 中島　浩：投機に投資しよう，情報処理，Vol.40, No.2, pp.195–201 (1999).
12) Nakashima, H., Sato, T., Matsuo, H. and Ohno, K.: Improved Implementation of the Speculative Memory Access Mechanism specMEM, *Innovative Architecture for Future Generation High-Performance Processors and Systems*, pp.97–104, IEEE Computer Society (2000).
13) O'Keefe, M.T. and Dietz, H.G.: Hardware Barrier Syhchronization: Static Barrier MIMD (SBM), *Proc. Intl. Conf. Parallel Processing*, Vol.1, pp.35–42 (1990).
14) 佐藤貴之，中島　浩：同期操作に対するメモリ・アクセスの投機的実行の提案，情報処理学会研究報告，98-ARC-129, pp.19–24 (1998).
15) 佐藤貴之，松尾治幸，大野和彦，中島　浩：specMEM：同期操作に対するメモリ・アクセスの投機的実行機構，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.41, No.SIG5 (HPS1), pp.1–14 (2000).
16) Sato, T., Ohno, K. and Nakashima, H.: A Mechanism for Speculative Memory Accesses Following Synchronizing Operations, *Proc. Parallel and Distributed Processing Symp.*, pp.145–154 (2000).
17) Seo, K. and Yokota, T.: Pegasus : A RISC Processor for High-Performance Execution of Prolog Programs, *Proc. Intl. Conf. on Very Large Scale Integration*, pp.261–274 (1987).
18) Smith, J.E.: Dynamic Instruction Scheduling and the Astronautics ZS-1, *Computer*, Vol.22, No.7, pp.21–35 (1989).
19) Smith, M.D., Johnson, M. and Horowitz, M.A.: Limits on Multiple Instruction Issue, *Proc. Intl. Conf. Architectural Support for Programming Languages and Operating Systems*, pp.290–302 (1989).
20) 玉造潤史，松本　尚，平木　敬：Loop を並列実行するアーキテクチャ，情報処理学会研究報告，96-ARC-119, pp.61–66 (1996).
21) Woo, S.C., Ohara, M., Torrie, E., Singh, J.P. and Gupta, A.: The SPLASH-2 Programs: Characterization and Methdological Considerations, *Proc. 22nd Intl. Symp. Computer Architecture*, pp.24–36 (1995).

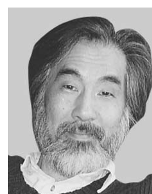
(平成 13 年 8 月 31 日受付)
(平成 14 年 2 月 13 日採録)

**松尾　治幸**

1977 年生．2002 年豊橋技術大学大学院工学研究科情報工学専攻修士課程修了．同年富士通プライムソフトテクノロジ入社．在学中は共有メモリ型並列計算機のアーキテクチャとシミュレーションに関する研究に従事．

**大野　和彦**（正会員）

1970 年生．1998 年京都大学大学院工学研究科情報工学専攻博士後期課程修了．同年豊橋技術科学大学助手．並列プログラミング言語の設計と最適化に関する研究に従事．博士（工学）．



**中島　　浩**（正会員）

1956 年生．1981 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 年京都大学工学部助教授．1997 年豊橋技術科学大学教授．並列計算機のアーキテクチャ等並列処理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．IEEE-CS，ACM，ALP，TUG 各会員．