計算機アーキテクチャ 139－3
（2000．8．3）

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† 中島浩† 大野和彦†

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・キャッシュへ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なった結果，Radix Sortで見られた投機的実行による悪影響を60％程度削減できることが明らかになった．また2次キャッシュの導入はspecMEMの効果を高め，LU分解の性能向上率が22％から25％に増加することも明らかになった．

Improvement of the Speculative Memory Access Mechanism: specMEM

Haruyuki Matsuo,† Hiroshi Nakashima† and Kazuhiko Ohno†

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 factor of Radix Sort is reduced by 60%. It is also shown that secondary cache effectively improves the performance of specMEM. For example, the speed-up of LU decomposition is 25% with secondary cache while 22% without that.

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なうことはできない．この同期の成立が確認されるまでの共有変数へのアクセスの禁止は，操作の正当性を保証するためのものであるため，禁止対象や期間を必要最小限に留めることが困難であり，そのため不必要なオーバヘッドが生じる．

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[1),2)]．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

† 豊橋技術科学大学
Toyohashi University of Technology

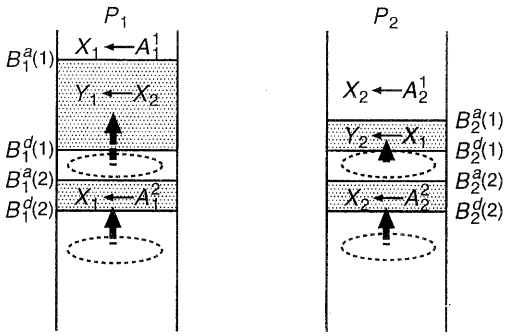

図1　投機的メモリ・アクセス

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，失敗にともな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明らかになっている．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2次キャッシュを持つシステムにspecMEMを適用する場合，機能メモリを用いた複数ラインの一括状態を行なう実装が困難であるという課題もあった．

そこで本報告では，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，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，第2章ではspecMEMの概要を，第3章で上記の二つの改良方式を述べた後，第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

specMEM では，同期操作によって充足されるデータ依存制約が，同期成立確認以前にも充足されていることを仮定し，同期操作以降の処理を投機的に行なう．例えば図1に示す二つのプロセッサ P_j $(j = 1, 2)$ は，共有変数 X_i $(i = 1, 2)$ の依存制約を充足するためのバリア同期に $B_j^a(k)$ $(k = 1, 2)$ で到達すると，その成立を確認することなく後続の操作を続行する．その結果，X_i は同期成立確認以前に参照／更新されるが，そのタイミングが図に示すように依存制約を満たすものであれば，これらの投機は成功してバリア同期の待ち時間が完全に除去される．

2.2 投機の失敗

図2に示す例では，P_1 による X_2 の投機的読出が P_2 による X_2 への書込に先行し，その結果不正な値を読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出を行な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なう必要がある．

specMEM ではこの投機的失敗の検知と無効化をコヒーレント・キャッシュを拡張して実現している．まず，バリア到達から確認までの期間においてアクセスされたキャッシュ・ラインには，全て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 の参照前の状態が **S** (Shared) であるとすると，投機的読出の結果 **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** が付いた状態にあるラインに対する他のプロセッサからの書込通知を受けとると，不正値を読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 への書込通知によって，P_1 のキャッシュが不正読出を検知する．

次に行な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おこなう．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全て無効化され，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．なお複数のプロセッサが互いに相手をロールバックさせることによるデッドロックを防止するために，ロールバック後の再実行は同期成立を確認してから行なう．

このロールバックが P_1 で生じた後，P_1 では X_2 の値を再び読出す．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番目のバリアについて X_2 への投機的書込を行なう．この書込は図に示すように P_1 による読出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が，**UM** への遷移に伴って旧値（この場合は真値）がメモリに書き戻されている．そこで P_1 による読出の際に，P_2 のキャッシュにある書込後の値 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 へ返し，逆依存制約の破綻に対処する．

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．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

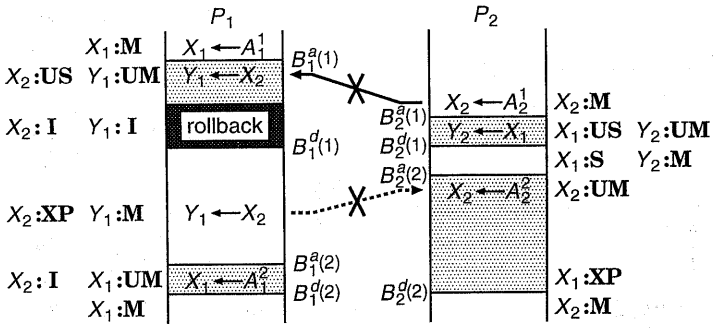

図 2　投機の失敗

結果，P_1 が X_2 を改めて読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．また図には示されていないが，投機的書込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，書込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．

2.3　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が **M**, **S**, および **I** (Invalid) とした時，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なわれる（図 3）．

(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出 ($r(s)$) を行なうと，投機的読出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．
同様に投機モードでの書込 ($w(s)$) では **UM** に遷移し，旧状態が **M** であればライトバックを行なう．またこの投機書込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なう．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 (σ^e)，**US** のラインは全て **S** に，また **UM** のラインは全て **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なう．

(3)　一方，**US** または **UM** のラインに対する他プロセッサの書込，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)．同時に全ての **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 (σ^b)，**XP** のラインの値は既に古いものとなっている可能性があるため，全ての **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．

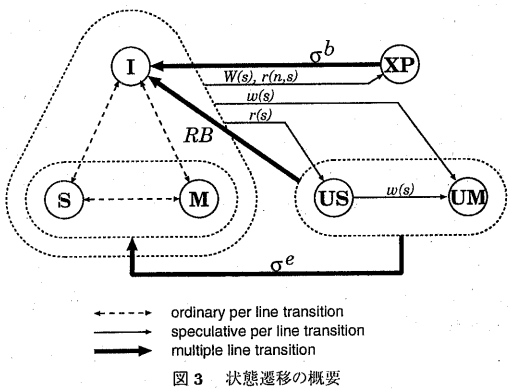

図 3　状態遷移の概要

上記のように，投機の成功 (σ^e)，失敗 (RB) および開始 (σ^b) の際に，複数のラインの状態を一括して遷移させる必要がある．この一括状態遷移はマスク付リセット機能を持つ機能メモリを用いることによって，定数時間で実現することができる[1),2)]．

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，ベースとなるキャッシュ状態と同じ数だけの投機的状態を追加すれば良いため，キャッシュ・タグに投機状態を示す 1 ビットを追加すれば実現することができる．しかしこの方式では，ロールバックが生じた時に投機状態を示すビットがオンであるラインを全て無効化する必要があり，以下のような問題点が存在する．

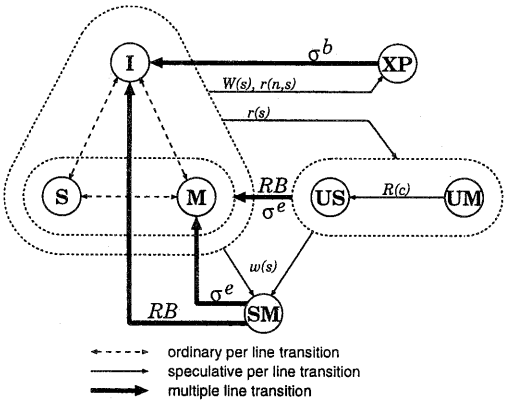

図4　改良後の状態遷移

(1)　ロールバック後の再実行時のミス率が大きな値となる．特に投機的読出だけが行なわれたライン（前章の例ではUSのライン）の無効化は不必要であるにも関わらず，投機的書込が行なわれたライン（前章の例ではUMのライン）とともに無効化されてしまう．

(2)　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出の際に，USに遷移すると同時にメモリへライトバックを行なう必要がある．

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な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[3]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出だけが行な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値を得ることができる．

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

(2)　投機的読出が依存制約を満たさないタイミングで行なわれ，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なうため，再実行開始時点では読出に先行すべき書込が完了しており，その書込によってキャッシュは正しく無効化されている．

そこで投機的書込が行な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，投機的読出だけが行な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4のようになり，基本的な実装（図3）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出 ($r(s)$) 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出要求があると ($R(c)$)，ラインの値を返してUSへ遷移する．

(2)　投機的書込 ($w(s)$) が行なわれると，全ての状態から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．

(3)　投機が成功すると (σ^e)，Uマークが全て除去され，同時にSMのラインは全てMへ遷移する．

(4)　ロールバックが生じた場合 (RB)，SMのラインは全て無効化される．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なわれたラインは，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，キャッシュ・タグにには投機状態を示すビットの他に，SMであることを示すビットが必要になる．

3.3　2次キャッシュの導入

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性能が向上することが期待できる．また投機的書込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押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な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，ベースとなる状態に以下の2状態だけを追加し，図5に示すような状態遷移を行なう．

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．投機的書込 ($w(s)$) は2

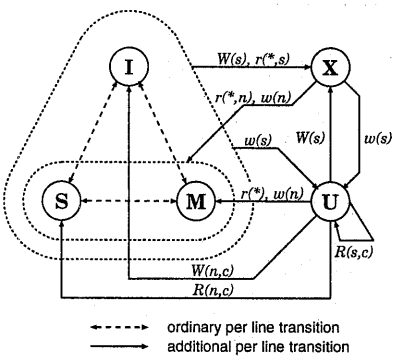

図5　2次キャッシュの状態遷移

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なわれるが，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 に留まる．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なう．

- プロセッサからの読出 ($r(*)$) あるいは非投機的書込 ($w(n)$) 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．この時 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出要求 ($R(c,n)$, $R(c,s)$) 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 ($R(c,s)$) 要求元にその旨を伝えて（要求元は XP へ遷移）U に留まり，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込要求 ($W(n,c)$) 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，読出要求と同様に 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 の一括無効化が行なわれるが，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 に留まる．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 とみなすことができる．したがって X と I の差異は，他プロセッサからの書込要求を 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出が行なわれたことを示す状態はなく，不正読出の検出は 1 次キャッシュでのみ行なわれる．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 1 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．バリア同期ついては，単純な線バリアをサポートするハードウェア機構の存在と，その同期成立がキャッシュに可視であることを想定した．また表 1 に示すバリア同期コストは，最後のプロセッサがバリアに到達してから同期成立が確認されるまでの時間とした．

評価のためのワークロードとしては，SPLASH-2[3] の中から以下の 3 つのプログラムを選択した．

- 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．
- FFT

表 1　評価に用いたマルチプロセッサモデル

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	+10
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

☆ 実際には失敗と開始を含む．

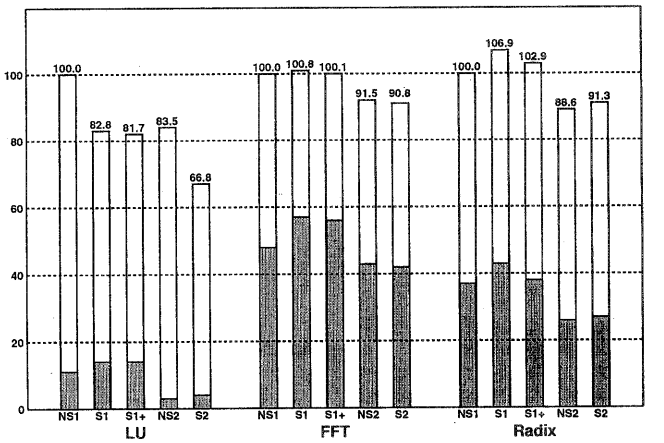

図6　SPLASH-2による評価結果

負荷の偏りが全くないため，投機的アクセスはほとんど行なわれない．

- 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なわれるが，高負荷プロセッサの実行時間は短縮されない．

4.2 評価結果

図6は，前述の3つのプログラムの実行時間（サイクル数）を，以下の5種類のシステムについて示したものである．

- 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．

なお実行時間はNS1を100として正規化してあり，網かけの部分はクリティカル・パスを実行するプロセッサでのキャッシュミス・ペナルティを表している．

図から明らかなように，**SM**の導入によってRadix Sortのキャッシュミス・ペナルティが削減され，投機的実行による悪影響を60%程度除去することができた．またLU分解とFFTについても，僅かではあるが効果が見られた．

一方2次キャッシュ導入により，specMEMの効果がより高くなることも明らかになった．たとえばLU分解でのspecMEMによる性能向上率は，1次キャッシュのみの場合が22%であるのに対し，2次キャッシュを持つシステムでは25%であり，通常のメモリを用いた2次キャッシュであっても投機実行に有効であることが明らかになった．

5. おわりに

本報告ではキャッシュミス・ペナルティを削減するためのspecMEMの改良方式として，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た．またこの二つの改良の効果をSPLASH-2ベンチマークを用いて評価した結果，Radix Sortで見られた投機的実行による悪影響を60%程度削減できることと，2次キャッシュによりLU分解の性能向上率が22%から25%に増加することが明らかになった．

しかしキャッシュミス・ペナルティの削減は十分であるとは言えず，投機によるキャッシュミス・ペナルティ増加を完全に押えることはできていない．今後，ペナルティ増加の要因を詳細に解析し，その結果に基づきさらに改良を施す予定である．

参考文献

1) Sato, T., Ohno, K. and Nakashima, H.: A Mechanism for Speculative Memory Accesses Following Synchronizing Operations, *Proc. Intl. Parallel and Distributed Processing Symp.*, pp. 145–154 (2000).
2) 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. HPS1 (2000). 掲載予定．
3) 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).