チップマルチプロセッサの同期付きキャッシュメモリに対するミスペナルティ隠蔽機構

山脇 彰[†] 岩根雅彦[†]

キャッシュミスペナルティの隠蔽はプロセッサの性能向上にとって重要であり，多くの研究がなされてきた．一方，TSVM キャッシュはチップマルチプロセッサにおいて，プロセッサコア間の同期通信を共有変数の一貫性制御と同時に実現し，より効率的な並列処理の実現を目指す．TSVM キャッシュは，タスク，および，スレッド間でのエントリの有効利用を考慮し，アドレスではなくタスク ID とスレッド ID からなるタグでラインを指定する．さらに，外部に特殊なメモリを必要としないように，TSVM キャッシュがキャッシングする同期付き共有変数は同期通信情報をフィールドに持つ構造体として扱われる．よって，TSVM キャッシュは，従来型キャッシュとは異なり，キャッシュミス時にタグをアドレスに変換してキャッシュ–メモリ間で構造体を転送する．本論文では，TSVM キャッシュに 3 つの機構からなるミスペナルティ隠蔽機構の導入を試み，同期通信の融合とともにさらなる性能向上を図る．それらの機構を持たない場合，従来型キャッシュに対する速度向上は平均で 1.11 倍（最大で 1.58 倍），バス使用率が平均で 44%（最大で 74%）だったが，導入機構は速度向上を平均で 1.91 倍（最大で 2.77 倍），バス使用率を平均で 11%（最大で 17%）に改善させた．さらに，並列度の増加にともなった同期通信オーバヘッドの増加は，性能に悪影響を与える一方で，プリフェッチの遅れに対する耐性を高めることが定量的に確認された．

Mechanisms Hiding Miss Penalty for Cache Memory to Shared Variables with Synchronization on a Chip-multiprocessor

Akira Yamawaki[†] and Masahiko Iwane[†]

To hide cache miss penalty is important for improving a performance of processors. On a chip–multiprocessor, the TSVM cache performs inter–processor communication and synchronization simultaneously with coherence maintenance to make parallel processing more efficiently. To utilize entries more efficiently among tasks and threads, the TSVM cache specifies a shared variable with synchronization by the tag including the task ID and thread ID. To not use a structured physical memory with synchronization mechanism, the TSVM cache treats shared variables with synchronization as just structures on a conventional main memory. Thus, when a cache miss occurs, the TSVM cache translates a tag to the address and transfers the structured line to the memory. This paper attempts to improve the performance more by introducing the mechanisms to hide the miss penalty for the TSVM cache. The result shows that the introduced mechanisms improve a speedup up to 2.77 times (1.91 times on average) compared with a conventional data cache, and suppress a bus utilization up to 17% (11% on average). It is also confirmed that overheads of communication and synchronization make a capability of prefetching delay strong.

1. まえがき

近年，単一プロセッサに対する ILP や動作クロック周波数の向上に限界が見えてきた[1]．そのような中で，チップマルチプロセッサ（CMP）[2)~4)] は有望なプロセッサアーキテクチャの 1 つである．各プロセッサコアが L1 キャッシュを持った共有メモリ型マルチプロセッサでは，プロセッサ間通信は，キャッシュ間で共有変数の一貫性を保つことにより実現され，一般的にプロセッサ間で同期と通信は別に行われる．

我々はそのような CMP において，より効率的な並列処理の実現を目指したキャッシュアーキテクチャを提案している[5)~8)]．そのキャッシュアーキテクチャは，各プロセッサコアが持つ従来型データキャッシュの一部に同期共有変数（通信に同期を必要とする変数）をキャッシングする TSVM（Tagged Shared Variable Memory）キャッシュを持つ．TSVM キャッシュは，キャッ

† 九州工業大学工学部
Faculty of Engineering, Kyushu Institute of Technology

シュ間で同期共有変数の一貫性を保つ（通信）と同時に，同期も実行する．つまり，プロセッサコア間の同期と通信が，チップ内部の高速な結合網上での 1 回の一貫性制御により実現される．

キャッシュメモリのミスペナルティ隠蔽はプロセッサの性能向上に重要であり，プリフェッチを中心に多くの研究がある[9]．特に，キャッシュミス時のラインフィルや置換えによるバス転送に一貫性制御のバス転送も加わるマルチプロセッサでは，ミスペナルティの隠蔽がシングルプロセッサよりも重要である[10)~12)]．

TSVM キャッシュは，従来のキャッシュメモリと異なり，同期共有変数を同期通信情報のフィールドを持った構造体として扱う．また，タスク，および，スレッド間でエントリの有効利用を考慮し，タスク ID とスレッド ID からなるタグでラインを指定する．よって，TSVM キャッシュはキャッシュミス時にタグをアドレスに変換してキャッシュ–メモリ間で構造体を転送する．

本論文では，TSVM キャッシュに対してミスペナルティ隠蔽機構の導入を試み，同期通信の融合とともにさらなる性能向上を目指す．導入機構は，タグ・アドレス変換レイテンシを削減する機構と，無駄なライン転送を削減する機構，および，ライン転送のオーバヘッドを隠蔽するソフトウェア・プリフェッチ機構である．

以降，2 章で TSVM キャッシュの概要について述べ，3 章で導入する 3 つのミスペナルティ隠蔽機構について説明する．4 章の実験では，導入した機構の効果を定量的に明らかにする．そして，5 章で TSVM キャッシュとミスペナルティ隠蔽機構のハードウェア量について簡単に言及する．次いで，6 章で他のミスペナルティ低減手法と同期付きキャッシュメモリとの関連について議論し，最後に 7 章でむすぶ．

2. TSVM キャッシュの概要

2.1 TSVM キャッシュを搭載した CMP

TSVM キャッシュを搭載した CMP の概念を**図 1** に示す．タスクはプログラムの実行環境であり，1 つ以上のスレッドから構成される．スレッドは動的にマイクロスレッド数分のプロセッサコアに割り当てられ，各々のマイクロスレッドは異なる 1 台のプロセッサコアに割り当てられる．スレッドはユーザによって指定された一連の命令実行であり，粗粒度（関数やジョブ）の並列性を持つ．マイクロスレッドはスレッドが従来の静的並列化手法によって並列化されたコード列であり，近細粒度（文列程度）[3)]～中粒度（ループ程度）の並列性を持つ．プロセッサコアはスレッドの消滅まで割り当てられた同一のマイクロスレッドを実行し続ける．タスク，スレッド，および，マイクロスレッドは共有変数を介して協調動作する．共有変数のうち，通信に同期を必要とする変数を同期共有変数，その他の変数を一般変数とよぶ．

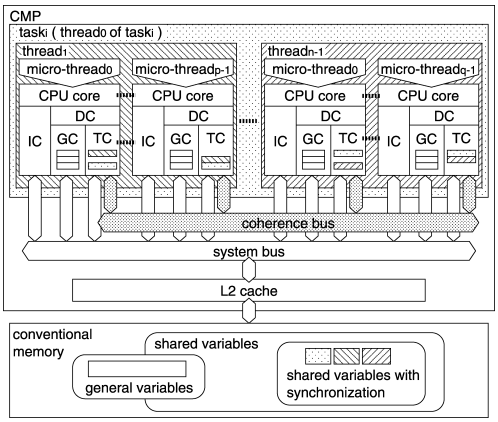

図 1 TSVM キャッシュを搭載した CMP の概念
Fig. 1 Concept of CMP with TSVM cache.

同期共有変数はデータキャッシュ（DC）に導入された TSVM キャッシュ（TC）によってキャッシングされる．データキャッシュの残りの部分が一般変数キャッシュ（GC）となり，一般変数をキャッシングする．TSVM キャッシュは同期機能を有する更新型の一貫性制御プロトコルを持つ（一貫性制御プロトコルの詳細は文献 6) を参照されたい）．同期共有変数を介した同期通信は TSVM キャッシュの一貫性制御と同時に暗黙的に実行される．TSVM キャッシュの一貫性制御は，専用に設けられたバス（一貫性バス）上で行われる．TSVM キャッシュミス時は，ラインが，システムバスを介して，メモリと TSVM キャッシュ間で転送される☆．

2.2 データキャッシュの構成

図 2 に TSVM キャッシュと一般変数キャッシュからなるデータキャッシュの構成を示す．

同期共有変数はタグ（tag）によって識別され，そのタグはタスク ID，スレッド ID，および，同期共有変数 ID からなる．これは，TSVM キャッシュの各エントリをタスク，スレッドごとに動的に保護し，エントリの有効利用を図るためである[13)]．TSVM キャッシュは同期通信の状態をカウンタ（sc）によって表現する．カウンタが 0 ならば empty であり，非 0 ならば full である．異なる TSVM キャッシュ間でカウン

☆ システムバスを介したライン転送時に同一ラインに対する一貫性制御があった場合，一貫性バス上のラインが最新値なので，システムバス上のライン転送は中断される．

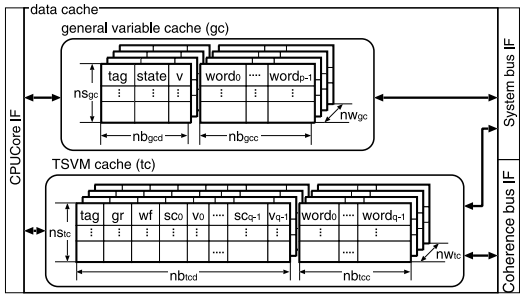

図 2 TSVM キャッシュを持ったデータキャッシュの構成
Fig. 2 Organization of data cache with TSVM cache.

タの内容を一貫させることによりプロセッサ間の同期通信が実現される．待合せフラグ（wf）は，1 対多のマルチキャストが完了するまですべての読み出しを待たせるために使用されるフラグであり，wf を活用することによって効率的なバリア同期を実現できる[6]．グループフラグ（gr）は，TSVM キャッシュ群をタスク，および，スレッドごとにグループ化し，グループ間で無関係な同期共有変数を取り込まないようにするためのフラグである[5]．

TSVM キャッシュのラインあたりのワード数，連想度，および，ラインの置換アルゴリズムは実装依存である（導入対象となるデータキャッシュの機構をそのまま使用してもよい）．同様に，タグの各フィールド，カウンタの幅，および，TSVM キャッシュと一般変数キャッシュの割合も実装依存である．ただし，false sharing の削減を図り，一貫性制御を 1 ワード単位で行うには，各ワードに 1 つのカウンタとラインの存在を表す v ビットも付加する．

TSVM キャッシュの導入にあたって，ハードウェア量を総ビット数によって一次近似的に見積もることができる．図 2 の ns_{gc} は一般変数キャッシュのセット数であり，nw_{gc} はウェイ数である．ns_{tc} は TSVM キャッシュのセット数であり，nw_{tc} はウェイ数である．また，nb_{gcd}，nb_{gcc} は一般変数キャッシュのディレクトリ部とコンテンツ部のビット数を，nb_{tcd}，nb_{tcc} は TSVM キャッシュのそれらを表す．したがって，既存データキャッシュの総ビット数 tnb_{org} は以下のようになる．

$$tnb_{org} = (nb_{gcd} + nb_{gcc}) \cdot ns_{org} \cdot nw_{gc} \quad (1)$$

ただし，ns_{org} は既存データキャッシュのセット数である．また，TSVM キャッシュを導入した提案データキャッシュの総ビット数 tnb_{pro} は，次式で表される．

$$tnb_{pro} = (nb_{gcd} + nb_{gcc}) \cdot ns_{gc} \cdot nw_{gc} + (nb_{tcd} + nb_{tcc}) \cdot ns_{tc} \cdot nw_{tc} \quad (2)$$

したがって，総ビット数の増加率 gr_{tnb} は次式となる．

$$gr_{tnb} = (tnb_{pro} - tnb_{org})/tnb_{org} \quad (3)$$

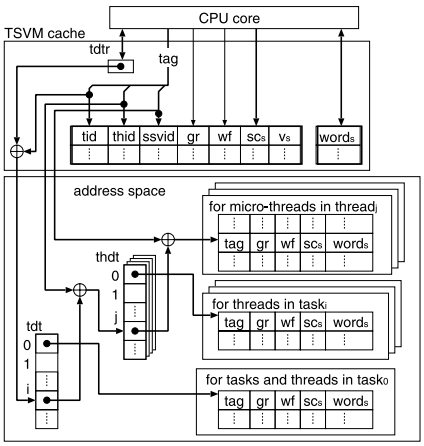

図 3 タグ・アドレス変換機構
Fig. 3 Tag to address translation mechanism.

2.3 タグ・アドレス変換

同期共有変数を識別するタグは，**図 3** に示すメモリ上の 2 つのテーブル（tdt: Task Description Table, thdt: THread Description Table）によって，物理，または，論理アドレスへ変換される．

TSVM キャッシュ内の tdtr（tdt Register）は，tdt のベースアドレスを保持するレジスタである．tdt の 1 番目以上のエントリは各タスクごとに存在する thdt のベースアドレスを保持し，tdt の 0 番目はすべてのタスク，スレッド間で使用される同期共有変数領域のベースアドレスを保持する．thdt の 1 番目以上のエントリはスレッド内のマイクロスレッド間で使用される同期共有変数領域のベースアドレスを保持し，thdt の 0 番目はタスク内の全スレッド間で使用される同期共有変数領域のベースアドレスを保持する．

アドレス変換時には，tdtr の指し示す tdt から，tid を添え字として，thdt のベースアドレスが読み出される．次いで，当該 thdt から，thid を添え字として，同期共有変数領域のベースアドレスが読み出される．最終的に，thdt から読み出されたベースアドレスに同期共有変数 ID（ssvid）を加算して，対象の同期共有変数がアクセスされる．

2.4 同期共有変数に対する命令と動作

TSVM に関する命令に関して，本論文に関係する命令のみを**図 4** に示す（他の命令に関する詳細は文献 5)〜7) を参照されたい）．図 4 中の RXLT，RXST は命令のニーモニックである．タグやカウンタ，および，wf，gr は従来のロード/ストア命令（LW/SW）におけるアドレス部で指定される[5]．

```
Conventional Load / Store      Load / Store Instructions for TSVM
LW   Register, Addr            RXLT  Register, tag_WordAddress
SW   Register, Addr            RXST  Register, gr_wf_sc_tag_WordAddr
```

図 4 同期共有変数に対する命令
Fig. 4 Load/store instructions for TSVM.

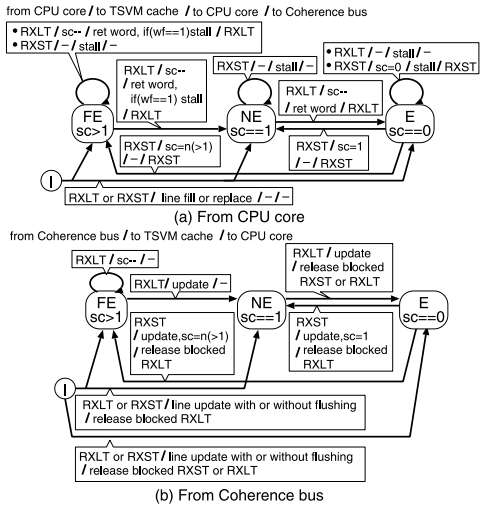

図 5 TSVM キャッシュラインの状態遷移図
Fig. 5 State transition diagram of TSVM cache line.

RXLT（RelaXed Load Tsvm）命令は，タグによって指定された同期共有変数のワードをレジスタにロードする．RXST（Relaxed Store Tsvm）命令は，タグによって指定された同期共有変数に対して，タグ，カウンタ，gr，wf，および，ワードを書き込む．

RXLT と RXST は，TSVM キャッシュのライン（一貫性制御を 1 ワード単位で行う場合はワード）の状態によって，データ依存ハザードを回避するためにストールされる（RXLT に対しては真依存，RXST に対しては逆依存）．ラインの状態遷移図を**図 5** に示す．図 5 (a) はプロセッサコアからのアクセスによる状態遷移を，図 5 (b) は一貫性バスからのアクセスによる状態遷移を表す．I（Invalid）はラインが TSVM キャッシュ上に存在しないことを意味し，E（Empty）はカウントが 0 の場合である．FE（Far-Empty）と NE（Near-Empty）はラインが full であることを意味し，前者はカウンタが 2 以上，後者はカウンタが 1 の場合である．以下，プロセッサコアと一貫性バスについてそれぞれ動作を説明する．

プロセッサコア：I ラインのアクセス時にラインがメモリからフィルされ（満杯時は置換えが発生），カウンタの値に従って当該ラインは FE，NE，または，E 状態に遷移する．RXLT はラインが E，または，ラインが FE で wf が 1 の場合，読み出し失敗でストールされる．読み出しの成功時に，RXLT は当該ラインのワードをプロセッサコアに返して，そのワードのカウンタをデクリメントし，更新したラインを一貫性バスに出力する．RXST はラインが FE，または，NE の場合に，書き込み失敗でストールされる．書き込みの成功時に，RXST は当該ラインのフラグとワード，および，カウンタを更新し，そのラインを一貫性バスに出力する．なお，一貫性制御が 1 ワード単位ならば，タグやカウンタ等に加えて 1 ワードのみが一貫性バスに出力される．

一貫性バス：一貫性バスには，前述のとおり，あるプロセッサコアの実行結果によって，RXLT では 1 減じられたカウンタを持つラインが，RXST では更新されたラインが出力される．それをスヌープした TSVM キャッシュは，一貫性バス上のラインで自ラインを更新し，カウンタの値に従って当該ラインの状態は E, FE，または，NE に遷移する．更新ラインが I ラインならば，ある空きエントリが更新される（満杯時はあるラインが書き戻される）．このとき，ラインが FE，または，NE になったら当該ラインに対する RXLT のストールが解除され，ラインが E になったら RXST，および，RXLT のストールが解除される．

なお，TSVM キャッシュ間でラインは，カウンタを含めて，厳密に一貫性が保たれる．しかしながら，RXLT，および，RXST が成功したら，プロセッサコアは一貫性制御の完了を待たずに次の命令へと実行を移す．また，プロセッサは TSVM キャッシュアクセスによるストール時に割込みを受け付けられる必要がある．そうでないと，あるマイクロスレッドがストールにより停止したら，システムも停止してしまう可能性がある．

3. ミスペナルティ隠蔽機構

3.1 ライン転送オーバヘッドの削減

TSVM キャッシュでは，タグ・アドレス変換のオーバヘッドを TATB（Tag to Address Translation Buffer）が削減し，SHT（Snoop History Table）が無駄なキャッシュフィル，および，フラッシュを削減する．TATB と SHT の構成をそれぞれ**図 6** (a) と同図 (b) に示す．

TATB の CAM（Content Addressable Memory）において，ディレクトリ部の左側は tid を，右側は thid を保持し，コンテンツ部ではそれぞれ tdt[tid] と thdt[thid] を保持する．CAM は tid と thid の一致検索によってアクセスされる．TSVM キャッシュミス時に tid と thid のヒットおよびミスによって以下のア

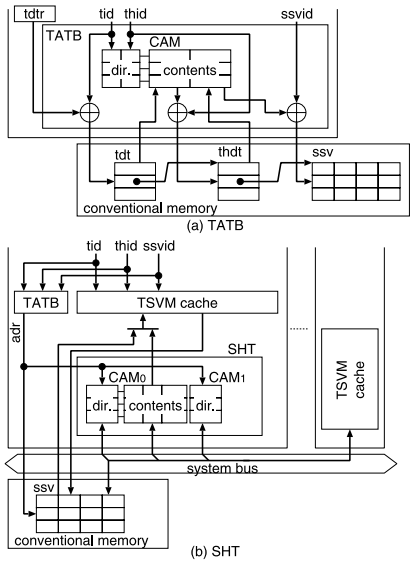

図 6 TATB と SHT の構成
Fig. 6 Organization of TATB and SHT.

クションが発生する．

(1) tid と thid がヒット：メモリ上の同期共有変数（ssv）にアクセスする．
(2) tid でヒット，thid でミス：CAM にエントリを確保し，tid，thid，および，ヒットしたエントリの tdt[tid] を確保したエントリに登録する．次いで，メモリ上の thdt[thid] を確保エントリに登録した後に，メモリ上の同期共有変数にアクセスする．
(3) tid でミス：CAM 上に新たにエントリを確保し，tid と thid を登録する．次いで，メモリ上の tdt[tid] と thdt[thid] を確保エントリに登録した後に，メモリ上の同期共有変数にアクセスする．

SHT の CAM_0 はラインフィルの削減を，CAM_1 はラインフラッシュの削減を担当する．CAM_0 は他 TSVM キャッシュがフィルしたラインを保持し，CAM_1 は他 TSVM キャッシュがフラッシュしたラインのアドレスを保持する．双方とも，つねにシステムバスをスヌープしており，内容を最新値に保つ．TSVM キャッシュミス時にラインをフィルする際，CAM_0 が検索され，一致エントリが存在するならば，CAM_0 中のラインが TSVM キャッシュに返される．TSVM キャッシュミス時にラインをフラッシュする際，CAM_1 が検索され，一致エントリが存在するならば，ラインフラッシュはキャンセルされる．ただし，他 TSVM キャッシュが発生したラインフィル/フラッシュの最新値のみが保持されるので，メモリとの一貫性を保つために，自 TSVM キャッシュの更新時に CAM_0 と CAM_1 上の同一ラインを保持したエントリは無効化される．

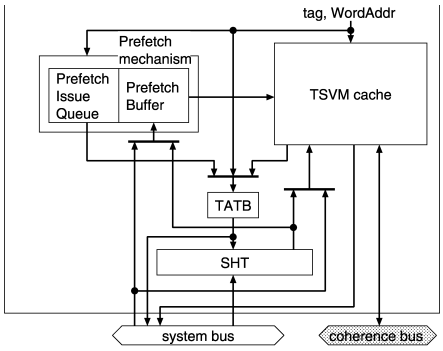

図 7 TSVM キャッシュのプリフェッチ機構
Fig. 7 Prefetch mechanism for TSVM cache.

TATB，SHT ともに，新たなエントリへの登録時に空きがない場合は単純に上書きすればよい．エントリ数や登録するエントリの選択方法は実装依存である．同様に，ディレクトリ部の幅も，tid，thid，および，アドレスの大きさに依存する．

3.2 プリフェッチ機構

TSVM キャッシュのプリフェッチはソフトウェア・プリフェッチであり，命令フォーマットを以下に示す．

```
PFT Register, tag_WordAddr
```

PFT 命令は，タグで指定された同期共有変数群（ライン）をメモリからフィルしてくる．レジスタは，単なるダミーから，ストライド幅やとってくるラインの個数の指定等に使用できる．PFT 命令は，他のソフトウェア・プリフェッチと同様に，プログラマやコンパイラ，または，その両方によって挿入される．

PFT 命令をサポートする機構は**図 7** のプリフェッチ発行キューとプリフェッチバッファからなる．

プリフェッチ発行キューは，プロセッサコアが発行した PFT 命令を蓄えるバッファである．プロセッサコアが PFT 命令を発行した際に，TSVM キャッシュとプリフェッチ発行キューが同時に検索される．無駄な PFT 命令が発行されないように，TSVM キャッシュ，または，プリフェッチ発行キューに同一タグ（同一ライン）が存在する場合，その PFT 命令は無視される（プロセッサコアは次命令の実行に移る）．そうでない場合は，プロセッサコアの発行した PFT 命令はプリフェッチ発行キューに登録される．プリフェッチ発行キューが満杯でない限り，プロセッサコアは PFT 命令を発行し続けることができ，PFT 命令の発行後に次命令をすぐさま実行する．プリフェッチバッファは FIFO 方式で PFT 命令をシステムバスに対して発行する．発行時に TSVM キャッシュが満杯ならば，キャッシュラインの書き戻し（キャッシュフラッシュ）が発

生する．つまり，PFT 命令は，ラインフィルのみではなく置換えのオーバヘッドに対しても隠蔽を図る．プリフェッチ機構によるラインフィルや置換え時にも，TATB と SHT の組合せによって高速化が図られる．

プリフェッチバッファは，プリフェッチによるラインフィルの結果を一時的に格納するバッファであり，TSVM キャッシュに対する一貫性バスを介した同期通信とキャッシュフィルが並列に実行される．プリフェッチバッファは，PFT 命令がプリフェッチ発行キューに登録されるとき，ある 1 つの空きエントリを確保し，そのエントリの添え字をプリフェッチ発行キューに登録する．プリフェッチが完了すると，取得ラインがプリフェッチ発行キュー内の添え字に対応したプリフェッチバッファのエントリに書き込まれる．PFT 命令に後続する TSVM キャッシュへのアクセスがプリフェッチバッファ内でヒットしたときにのみ，プリフェッチバッファから TSVM キャッシュにラインが移される．

まだシステムバスに発行されていないプリフェッチの対象ライン（プリフェッチ発行キューの先頭以外にあるライン）にプロセッサコアが PFT 命令以外でアクセスした場合，そのプリフェッチは失敗しプリフェッチ発行キューから登録が削除される．つまり，そのプロセッサコアはミスペナルティを完全に被る．しかしながら，プリフェッチ中のライン（プリフェッチ発行キューの先頭にあるライン）にプロセッサコアが PFT 命令以外でアクセスした場合，プロセッサコアが被るミスペナルティはすでに発行されたプリフェッチが完了するまでのレイテンシのみとなる．

プリフェッチ発行キュー，プリフェッチバッファの幅は，タグ，nb_{tcd}，および，nb_{tcc} の大きさに依存し，深さとともに実装依存である．

4. 実験および考察

4.1 実験環境

TSVM キャッシュを搭載したチップマルチプロセッサを評価するために，**図 8** に示すクロックサイクル精度のシミュレータを開発した．

開発した CMP のモデルは 8 台のプロセッサコアを持ち，各プロセッサコアは 16 KB の命令キャッシュと 16 KB のデータキャッシュからなる L1 キャッシュを持つ．プロセッサコアは，32 ビットで MIPS-II 命令のサブセットを持つ 5 段パイプのスカラプロセッサである（MMU は未実装）[14),15)]．データキャッシュは，G2 PowerPC コア[16)] を参考にして図 8 に示す構成とした．少ない容量でもミスペナルティをどの程度隠蔽できるか評価することも目的であるため，ここでは 15 KB の一般変数キャッシュと 1 KB の TSVM キャッシュとした．一般変数キャッシュ，および，TSVM キャッシュともにシングルポートであり，アクセスの優先度はスヌーピングが最も高く，次いでプロセッサコアの順である．システムバス，および，一貫性バスは，データ幅が各々 64 ビットと 32 ビットの共有バスであり，集中型アービタにより回転式の優先度で調停される（分割転送は未サポート）．

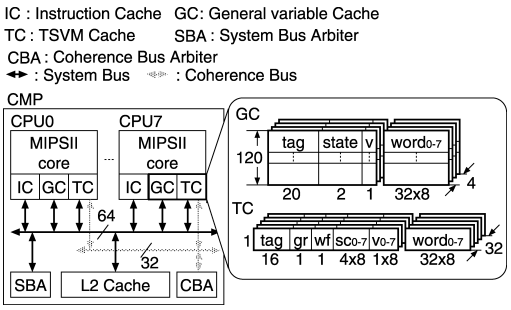

図 8 TSVM キャッシュを搭載した CMP モデル
Fig. 8 Model of CMP with TSVM cache.

一般変数キャッシュはライトバック型で，ウェイ数は 4 である．ラインは 8 ワード（32 ビット × 8）構成で，LRU により置き換えられる．一貫性は更新型である Dragon プロトコルにより 1 ライン単位で保たれる☆．また，プロセッサコアが発行したメモリアクセス命令の順序は厳密に保たれる．

TSVM キャッシュの連想度はラインごとにタグの比較器を設けて 32 にした．空きエントリは優先度付きエンコーダにより選択され，置換えラインはランダムに選択される．タグのフォーマットに関して，同時に実行可能なタスク，および，スレッドの最大数は 8 個なので，tid と thid はそれぞれ 3 ビットとした．また，TSVM キャッシュが溢れるだけの十分な同期共有変数を指定できるように ssvid を 10 ビットとした．つまり，タグの幅は 16 ビットとなる．カウンタの幅は 8 個のマイクロスレッド間でのバリア同期をサポートできるよう 4 ビットとした[6)]．1 対多通信は最大 1 対 7 であり，カウンタの幅は十分である．TSVM キャッシュの一貫性制御は 1 ワード単位で実行される．メモリ上に，TSVM キャッシュの 1 ラインは，**図 9** に示すような構造体（同期共有変数群）として割り当てられる．また，プロセッサコアが発行した RXLT と RXST 命令の順序は厳密に保たれる．

☆ 本実験では生産者–消費者間の同期通信（条件同期）を用いる．文献 8) において条件同期では無効化型の一貫性プロトコルよりも更新型の方が大幅な性能向上を達成したので，更新型として一般的な Dragon プロトコルを選択した．

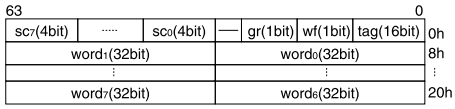

図 9 TAG，および，メモリ上での同期共有変数
Fig. 9 TAG format and structure of SSV on memory.

表 1 基本性能
Table 1 Basic performance.

TSVM キャッシュ	
種別	設計値
ヒット時のアクセス	1
一貫性制御	4†
タグ・アドレス変換	4†/8†
ラインフィル	2/9†
ラインフラッシュ	2/9†
一般変数キャッシュ	
種別	設計値
ヒット時のアクセス	1
一貫性制御	7†

† バス調停にかかる 1 クロックを含む

ミスペナルティ隠蔽機構に関して，TATB と SHT のエントリ数は 4 に固定した．また，プリフェッチ発行キュー，および，プリフェッチバッファのエントリ数も 4 に固定した．実装した PFT 命令は 1 ラインをプリフェッチバッファに転送するだけであり，ストライド転送や任意の転送数はサポートしていない．

測定に関して，命令キャッシュと一般変数キャッシュ，および，L2 キャッシュはつねにヒットすると仮定した．つまり，これらのキャッシュに対しては既存のプリフェッチ等のミスペナルティ隠蔽手法が完全に成功することと等価である．以上の条件下における，キャッシュメモリの単体性能（クロック数）を**表 1** に示す．TSVM キャッシュにおけるラインフィルとラインフラッシュに関して，スラッシュの左側が SHT にヒットした場合の値であり，右側がミスした場合の値である．TATB にミスした場合，タグ・アドレス変換分が加算される．タグ・アドレス変換に関してスラッシュの左側は tid がヒットして thid がミスした際の値であり，右側は tid がミスした場合の値である．

以降では，単一スレッド中の複数マイクロスレッドによる並列処理のみを対象とする．コンパイラには gcc-2.95 を使用し，最適化オプションは -O3 とした．

4.2 一貫性制御の動作詳細

一般変数キャッシュと TSVM キャッシュの一貫性制御について，本実験での実装における動作の詳細を，それぞれ**図 10** (a) と同図 (b) に示す．ただし，一般変数キャッシュに関しては，実験で関係のある更新トランザクションのみを示す．

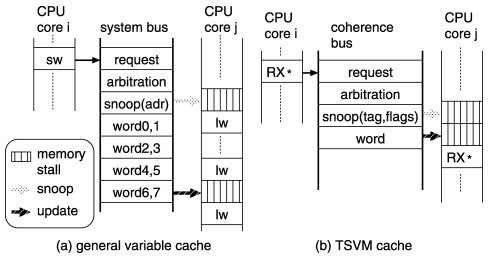

図 10 一貫性制御の動作詳細
Fig. 10 Detail of coherence maintenance.

一般変数キャッシュは，プロセッサコアが共有状態のラインにストア命令を発行すると，他キャッシュの同一ラインも更新するためにシステムバスを要求する．そして，バス調停を経た後にアドレスが出力され，他キャッシュはそれをスヌープする．最後に，1 ライン分の 8 ワードが出力される．更新すべきラインをスヌープした他キャッシュは，最後のワードが出力されるまでは一時レジスタにそれまで観測したワード群を保存し，最後のワードを観測したと同時にそのワードと一時レジスタの内容を自ラインに書き込む．したがって，プロセッサコアは，自キャッシュが他キャッシュが発行した更新トランザクションによってヒットラインを更新中でも，スヌープと最後のワード転送時以外で自キャッシュ上の他ラインをアクセスできる．ただし，同一ラインへのストア命令は，一貫性制御の完了までストールされる．一般変数キャッシュは，一貫性制御のために 4 エントリのバッファを持ち，そのバッファに空きがある限り，プロセッサコアは更新トランザクションの完了を待たない．

TSVM キャッシュは，プロセッサコアによる RXLT および RXST 命令の発行を受けて，一貫性バスを要求し，調停後にスヌープ（タグ，カウンタ，wf，および，gr が出力される）を経てワードを出力する．一般変数キャッシュと同様にスヌープとワードの出力時以外は，一貫性制御中でもプロセッサコアは TSVM キャッシュにアクセスできる．ただし，同一タグへのアクセス時には一貫性制御の完了を待つ．TSVM キャッシュも一貫性制御用に 4 エントリのバッファを持ち，RXLT と RXST 命令の成功時に，そのバッファに空きがある限り，プロセッサコアは一貫性制御の完了を待たない．

4.3 評価プログラム

4.3.1 並列化および TSVM 命令の挿入

キャッシュミスペナルティは，たとえば大容量の配列要素へのアクセス等によって，キャッシュミスが頻発する際に問題となる．ここでは，**図 11** (a)，(b) の

```
for( i = 1; i < n ; i++ )
  z[ i ] = z[ i - 1 ] + x[ i ] * y[ i ];
```
(a) loop1

```
for( k = 2; k <= nwall(l), k++ ){
  cp[l][k] = cp[l][k-1] + (3. * (dpds[l][k]) + dpds[l][k-1])
           + dpds(1+mod(k,nwall[l]), l)
           + dpds(1+mod(k+nwall[l]-3, nwall[l]), l))
           / (4. * delt * cupst)
  cpm = max (cpm, cp[l][k])
}
```
(b) loop2

図 11 評価プログラム
Fig. 11 Evaluation programs.

```
for( i = 1+mid; i < n; i += nm ){
  tmp1 = x[ i ] * y[ i ];
  RXLT( z[ i -1 ], tmp2 );
  tmp1 = tmp1 + tmp2;
  RXST( z[ i ], 1, tmp1 );
}
```
(a) loop1

```
for( k=2+mid; k<=nwall[l]; k+=nm ){
  tmp1 = (3. * .... * cupst);
  RXLT( cp[l][k-1], tmp2 );
  tmp1 = tmp1 + tmp2;
  RXST( cp[l][k], 1, tmp1 );
  RXLT( cpm[mid], tmp2 );
  tmp2 = max (tmp2, tmp1);
  RXST( cpm[(mid+1)%nm], tmp2 );
}
```
(b) loop2

図 12 ループプログラムの doacross 型による並列化
Fig. 12 Parallelization of loop programs on doacross fashion.

```
prologue:  PFT( z[1+mid], dummy );
kernel:    for( i = 1 + mid ; i < n -nm; i += nm ){
             PFT( z[ i + nm ], dummy );
           }
epilogue:
```
(a) loop1

```
prologue:  PFT( cp[l][2+mid], dummy );
           PFT( cpm[mid], dummy );
kernel:    for( k=1+mid; k<=nwall[l]-nm; k+=nm){
             PFT( cp[l][k+nm], dummy );
           }
epilogue:
```
(b) loop2

図 13 PFT 命令の挿入
Fig. 13 Insertion of PFT.

ループを評価に用いた．前者は今回作成した例題プログラムであり，変数は整数である．後者は NAS kernel Benchmark Program の一部であり，変数の型は 32 ビットの単精度浮動小数点とした．

図 11 の各ループは依存距離が 1 のループ伝搬依存を持ち，それらを**図 12** のように同期共有変数を用いて doacross の形で並列化した．RXLT の第 1 引数はタグを，第 2 引数は読み出しデータを表す．RXST の第 1 引数はタグを，第 2 引数はカウント値を，第 3 引数は書き込みデータを表す．nm はマイクロスレッド数，mid は，マイクロスレッド ID を意味する．また，一時変数（tmp1，tmp2）はレジスタ変数である．

PFT 命令の挿入に関しては，**図 13** のように，

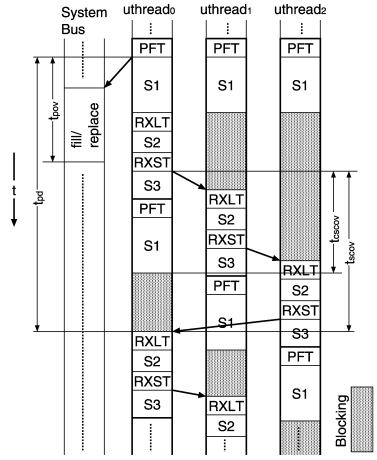

図 14 doacross ループの実行形態
Fig. 14 Execution of doacross loop.

doacross ループをプロローグ，カーネル，および，エピローグ部に分割し，最初の 2 つの部分に PFT 命令を挿入した．図中の PFT に関して，第 1 引数がタグを第 2 引数はダミーのレジスタを意味する．また，nm と mid は図 12 と同じである．プロローグ部の PFT 命令は最初のイタレーションで使用される同期共有変数をプリフェッチする．カーネル部のそれは，次イタレーションで使用する同期共有変数をプリフェッチする．エピローグ部は，繰返しの最後で無駄な PFT 命令を発行しないようにアンローリングされた，最後のイタレーションである．

実験ではシミュレーション時間の関係上，繰返し回数を 512 としたが，扱う同期共有変数の総量は 2 KB となり，今回のシミュレータにおける TSVM キャッシュの 2 倍の大きさになる．つまり，キャッシュ溢れの影響も実験結果に含まれる．

4.3.2 ループの特性

考察で用いるパラメータの定義を通して，2 個の doacross ループの特性を明らかにする．各マイクロスレッドは，**図 14** に示すように，ループ伝搬する配列要素に対する RXST と，RXLT によって同期通信しながら，割り当てられた繰返しを実行する．図中の PFT 命令から S3 が 1 つの繰返しを構成する．

t_{scov} は，あるマイクロスレッドにおいて，RXST による定義後から，TSVM キャッシュが，次繰返しにおける RXLT が成功する状態になるまでの時間であり，同期通信のオーバヘッドを表す．RXLT が成功する状態になるとは，その RXLT の参照に対する RXST の定義が一貫性制御によって TSVM キャッシュ上に反映し終えたことを意味する．

表 2 doacross ループの特性（平均値）
Table 2 Characteristics of doacross loops (average).

ループ	t_{cscov}	t_{scov}	t_{pd}	t_{pov}
loop1	19	10	34	14
loop2	76	20	146	14

t_{cscov} は，マイクロスレッドが，RXST による定義後から次繰返しの RXLT の直前までに要する時間であり，同期通信のオーバヘッドを隠蔽できる許容時間を表す．つまり，$t_{cscov} < t_{scov}$ ならば，次繰返しでマイクロスレッドはブロックされる．t_{cscov} が t_{scov} に対して大きいほど粒度が大きな doacross ループといえる．なお，t_{scov} は TSVM キャッシュミスや，システムバスの競合等によって動的に変化する．

t_{pd} は，あるマイクロスレッドが，PFT 命令の完了からその PFT 命令が対象とする TSVM 命令の直前までに費やした時間である．t_{pov} は，PFT 命令の発行から，実際にシステムバス上でラインのフィルや置換えが行われ，そのプリフェッチが完了するまでの時間である．つまり，$t_{pd} < t_{pov}$ ならば，プリフェッチが間に合わず，次の繰返しでマイクロスレッドは TSVM キャッシュミスを起こす．t_{pd} は t_{scov} の増加にともなって大きくなりうる．また，t_{pov} も，システムバスの競合によって動的に変化するブロッキング時間が長いほど，同期待ちによる性能低下が顕在化してくるが，プリフェッチの遅れ（t_{pov}）に対する耐性が高まる．

上記のパラメータに関して，2 つの doacross ループを 1 台のマイクロスレッドで実行した結果を**表 2** に示す．表中の値はクロック数であり，繰返しにわたった平均値である．また，1 つのマイクロスレッド（1 台のプロセッサコア）のみが動作するので，システムバス，および，一貫性バスの競合は発生しない．

4.4 速度向上

1 台のプロセッサで図 11 のループを実行した場合に対する速度向上比を**図 15** と**図 16** に示す（縦軸）．横軸は測定条件（記号）とマイクロスレッド数（数字）である．

横軸の b（ベース）は 3 つの隠蔽機構がない場合の結果であり，それに続く文字は以下の意味を持つ．

t：TATB を追加した場合

s：SHT を追加した場合

p：プリフェッチ機構を追加した場合

また，m1，m2 は，従来型のデータキャッシュ（図 8 の TSVM キャッシュが 0 KB で一般変数キャッシュが 16 KB）を用いた結果であり，**図 17** に示すように，ループ伝搬依存を同期フラグ[17] によって解消している☆．nm と mid は図 12 と同じである．m1 ではループ伝搬依存する配列要素の更新と同期フラグの更新時に Dragon プロトコルに従った更新トランザクションがそれぞれ発生する．一方，m2 は，一貫性制御をより緩和した場合であり，同期フラグの更新時にのみ，同一ライン上に存在する同期フラグと依存データを含めた更新トランザクションが発生する．これは，software-controlled updating[10] における更新トランザクションを発生させる専用ストア命令を同期フラグに適用して実現した．

結果から，全体的に loop1 よりも loop2 の方が良好な台数効果を示した．これは，表 2 で示したように，loop2 の方が比較的粒度が大きいためと推測できる．

m1 は，ループと並列度にわたって，平均で 1.31 倍，

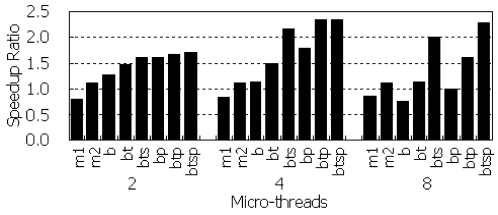

図 15 loop1 の速度向上比
Fig. 15 Speedup ratio of loop1.

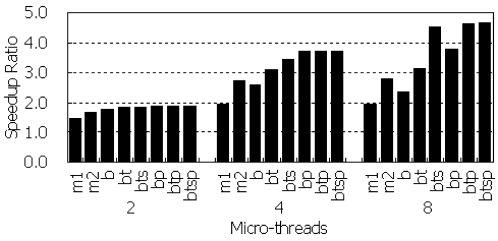

図 16 loop2 の速度向上比
Fig. 16 Speedup ratio of loop2.

```
struct shared_data_with_sync_flag{
   int d;
   int f;
}z[N], *curr, *prev;

*curr = &z[mid+1];   *prev = &z[mid];

//evaluation part
for( i = mid + 1 ; i < N; i += nm ){
        ⋮
  while(prev->f==0);  //waiting for synchronization flag
  curr->d=prev->d..;  //use and define with loop carried dependence
  curr->f = 1;        //setting synchronization flag
        ⋮
  curr+=nm; prev+=nm;
}
```

図 17 従来型キャッシュにおける doacross ループ
Fig. 17 Doacross loops on conventional cache memory.

☆ 同期フラグを含むすべての変数が一般変数キャッシュでつねにヒットするため，システムバス上で観測されるライン転送は，Dragon プロトコルに従った更新トランザクションだけである．

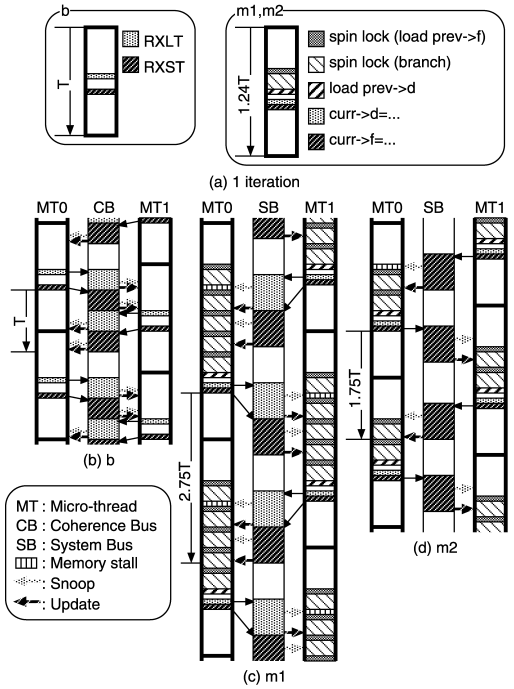

図 18 loop1 の動作解析
Fig. 18 Analysis of loop1 execution.

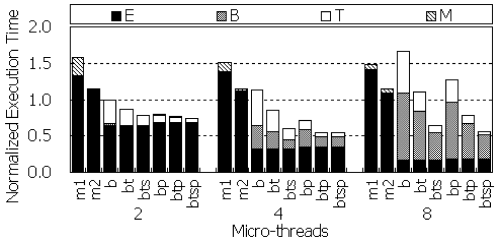

図 19 loop1 の実行時間の内訳
Fig. 19 Breakdown of loop1 execution.

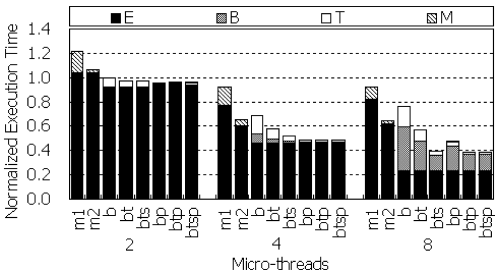

図 20 loop2 の実行時間の内訳
Fig. 20 Breakdown of loop2 execution.

最大で 2.00 倍，m2 は平均で 1.74 倍，最大で 2.78 倍の速度向上であった．一方，TSVM キャッシュだけ (b) では，同様に，平均で 1.66 倍，最大で 2.61 倍の速度向上にとどまり，loop1 の並列度 8，loop2 の並列度 4 以降で従来型キャッシュに対して性能の低下が見られた．これは，b のキャッシュミスペナルティが並列度の増加とともに増大したためと推測される．それに対して，btsp は，3 つのミスペナルティ隠蔽機構の導入によってすべての場合で良好な性能向上を達成でき，平均で 2.77 倍，最大で 4.66 倍の速度向上を得た．

また，m1 と m2 に対して，b はループと並列度にわたった平均で 1.11 倍（最大で 1.58 倍），btsp は，同様に，平均で 1.91 倍（最大で 2.77 倍）の速度向上を得た．このような性能差が生じる理由を明らかにするため，b，m1，m2 に関して，loop1 の並列度 2 におけるプロセッサコアとバスの動作を解析した．その結果を**図 18** に示す．図 18 (a) は 1 繰返しの命令実行であり，従来型キャッシュではスピンロックと同期フラグのセットが追加されたことにより，1 繰返しの実行時間が TSVM キャッシュに対して 1.24 倍になった．図 18 (b) から (d) が，b，m1，および，m2 の動作であり，それぞれ，キャッシュミスが発生していない定常状態においてマイクロスレッド 0 が繰返しを 2 回実行した際の様子を示す．従来型キャッシュの t_{scov} を現繰返しの同期フラグのセットから次繰返しで確認すべき同期フラグが一貫性制御によってセットされるまでと考えると，b と比較して，t_{scov} が m1 で 2.75 倍，m2 で 1.75 倍となった．このように，繰返しの実行時間と同期通信のオーバヘッドの差が TSVM キャッシュと従来型キャッシュとの性能差につながった．

4.5 導入機構の効果

4.5.1 実行時間の内訳

導入機構の効果を明らかにするため，各ループにおけるマイクロスレッドの実行時間の内訳をとった．各々を**図 19** と**図 20** に示す．縦軸はマイクロスレッドにわたった平均実行時間であり，並列度 2 における b を 1 として正規化されている．内訳に関して，E は演算の実行時間である．また，B，および，T は TSVM キャッシュアクセスにおけるストール時間であり，前者がブロッキング，後者がその他の要因による．M は一般変数キャッシュアクセスによるストール時間（プロセッサコアと一貫性制御の衝突による）である．横軸は並列度であり，記号は図 15 と同じである．

m1，m2 では，演算時間が他に比べて増大している．これは，スピンロックによって演算時間が増大したためである．また，b の従来型キャッシュに対する性能低下は，TSVM キャッシュのストール時間が増大したためであり，同期通信の融合だけではなくミスペナルティ隠蔽機構の導入が性能向上にとって重要なことが

表 3 TATB の効果
Table 3 Effects of TATB.

b 対 bt	loop1			loop2		
	2	4	8	2	4	8
速度向上	1.15	1.32	1.49	1.02	1.19	1.33
削減率	0.67	0.62	0.49	0.67	0.57	0.58

表 4 SHT，および，SHT と TATB の効果
Table 4 Effects of SHT and SHT with TATB.

bt 対 bts	loop1			loop2		
	2	4	8	2	4	8
速度向上	1.10	1.44	1.75	1.00	1.11	1.45
削減率	0.66	0.52	0.37	0.94	0.45	0.41
b 対 bts	loop1			loop2		
	2	4	8	2	4	8
速度向上	1.26	1.90	2.61	1.03	1.33	1.92
削減率	0.44	0.32	0.18	0.64	0.26	0.24

分かる．

4.5.2 TATB および SHT の効果

表 3 に，b に対する bt の速度向上と TSVM キャッシュアクセスにおけるストール時間の削減率を示す．なお，A に対する B の削減率とは B を A で割った値である．bt は b に対して，TSVM キャッシュアクセスのストール時間を 49%〜67%削減できたため，1.02〜1.49 倍の速度向上を示した．これは，TATB の導入によって t_{scov} を削減できたからと推測される．

続いて，bt と bts，および，b と bts の比較結果を**表 4** にまとめる．bt と比較し，bts は 1.00〜1.75 倍の速度向上を得た．これは，bt に対して 37%〜94%に TSVM キャッシュアクセスによるストール時間を削減できたことが原因である．つまり，SHT の導入による無駄なラインフィルとラインフラッシュの削減が t_{scov} をさらに削減し，性能を改善できたと推測される．b との比較結果から，TATB と SHT の組合せはストール時間を 18%〜64%まで削減でき，性能を 1.03〜2.61 倍に向上させた．

4.5.3 プリフェッチ機構の効果

b，bt，および，bts に対してプリフェッチを適用しなかった場合と適用した場合の比較結果を**表 5** に示す．

図 19 と図 20 より，プリフェッチ命令の挿入によって総演算時間が loop1 で 9%，loop2 で 2%増加した．しかしながら，表 5 より，TSVM キャッシュアクセスにおけるストール時間を 8%〜53%まで削減できたので，プリフェッチ命令を挿入しなかった場合に対し，1.01〜1.61 倍の速度向上を達成した．プリフェッチを適用した場合，各繰返しで各々のマイクロスレッドが t_{pd} 内にプリフェッチを完了したらキャッシュミスは発生しない．つまり，t_{scov} を削減でき，さらなる性能向上につながったと推測される．b に対する btsp の比較結果を**表 6** に示す．結果として，TATB，SHT，および，プリフェッチ機構によってストール時間を 7%〜21%まで削減でき，1.04〜2.96 倍の性能向上を得た．

表 5 プリフェッチの効果
Table 5 Effects of prefetching.

loop1				
p		b 対 bp	bt 対 btp	bts 対 btsp
2	速度向上	1.25	1.13	1.05
	削減率	0.29	0.31	0.37
4	速度向上	1.57	1.57	1.09
	削減率	0.29	0.21	0.42
8	速度向上	1.30	1.41	1.14
	削減率	0.53	0.39	0.41
loop2				
p		b 対 bp	bt 対 btp	bts 対 btsp
2	速度向上	1.04	1.02	1.01
	削減率	0.19	0.31	0.33
4	速度向上	1.42	1.20	1.08
	削減率	0.08	0.12	0.27
8	速度向上	1.61	1.48	1.03
	削減率	0.22	0.17	0.41

表 6 TATB，SHT およびプリフェッチの総合効果
Table 6 Total effect of TATB, SHT and prefetching.

b 対 btsp	loop1			loop2		
	2	4	8	2	4	8
速度向上	1.33	2.07	2.96	1.04	1.43	1.97
削減率	0.16	0.13	0.07	0.21	0.07	0.10

表 7 導入機構の組合せによるストール時間の削減率
Table 7 Reduction ratio of stall time due to combinations of introduced mechanisms.

b 対	bts	bp	btp	btsp
loop1	0.31	0.37	0.18	0.12
loop2	0.38	0.16	0.13	0.12

b に対する bts，bp，btp，および，btsp の TSVM キャッシュアクセスによるストール時間の削減率を**表 7** に示す．表中の数値は並列度にわたった平均値である．プリフェッチを適用した場合が，TATB および SHT を組み合わせた場合よりも，比較的，ストール時間を大きく削減できた．しかし，図 15 と図 16 から分かるように，bp や btp よりも bts の方が良い性能を示す場合がある．これは，プリフェッチのみではミスペナルティを十分に隠蔽できず，TATB や SHT との組合せが性能向上にとって重要であることを意味する．

4.6 プリフェッチと同期通信の関係

プリフェッチと同期通信のオーバヘッドの関係を明らかにするため，bp，btp，および，btsp に関して，t_{pd}，t_{pov}，および，TSVM キャッシュのヒット率を測定した．その結果を**表 8** に示す．表中の p はマイクロ

表 8 loop1 および loop2 の t_{pd}，t_{pov}，および，ヒット率
Table 8 t_{pd}, t_{pov} and hit ratio of loop1 and loop2 execution.

p		loop1			loop2		
		bp	btp	btsp	bp	btp	btsp
2	t_{pd}	36	35	35	148	147	147
	$\hat{t}_{pov}$	47	29	22	34	22	22
	$\bar{t}_{pov}$	33	20	12	26	17	12
	hit	0.90	1.00	1.00	1.00	1.00	1.00
4	t_{pd}	66	52	51	153	151	151
	$\hat{t}_{pov}$	102	43	22	51	22	22
	$\bar{t}_{pov}$	44	24	10	34	17	10
	hit	0.90	1.00	1.00	1.00	1.00	1.00
8	t_{pd}	192	119	90	237	201	200
	$\hat{t}_{pov}$	546	190	22	391	98	18
	$\bar{t}_{pov}$	109	39	8	63	18	7
	hit	0.88	0.97	1.00	0.98	1.00	1.00

スレッド数である．t_{pd}，$\hat{t}_{pov}$，および，$\bar{t}_{pov}$ はクロック数であり，t_{pd} と $\bar{t}_{pov}$ は繰返しとマイクロスレッドにわたった平均値，$\hat{t}_{pov}$ は繰返しとマイクロスレッドにわたった最大値である．ヒット率（表中の hit）はマイクロスレッドにわたった平均値である．

表 8 から分かるように，並列度の増加にそってヒット率はほぼ同じである．これは，並列度の増加にともなって t_{scov} が増加し，その結果，t_{pd} も増加したため，プリフェッチの遅れに対する耐性が高まったからである．また，btsp はすべての場合でミスペナルティを隠蔽できており，同期待ちが性能向上のリミットになっている．

4.7 バストラフィックの削減効果

導入機構がシステムバスのトラフィックに与える影響を明らかにする．その結果を**図 21** と**図 22** に示す．前者が loop1，後者が loop2 に関する結果であり，左のグラフが b の並列度 2 を 1 として正規化した総システムバスサイクル数であり，右側のグラフが，総システムバスサイクル数を各マイクロスレッドの総実行時間の平均値で割ったシステムバス使用率を示す．また，図中の記号は図 15 と同じである．

図 21 と図 22 から，本実験ではプリフェッチを適用しなかった場合と適用した場合のバスサイクル数は等しくなった．これは，図 7 のプリフェッチ発行キューと図 13 で示したループのリストラクチャリングによって，無駄なプリフェッチ命令が削除されたためと推測される．

表 9 は，b に対する bt，bs，および，bts の総システムバスサイクル数の削減率を示す．bt は b に対して総システムバスサイクル数を 57%〜60%に削減した．これは，TATB がフィルとフラッシュのレイテンシを削減した効果である．次いで，bs は b に対して

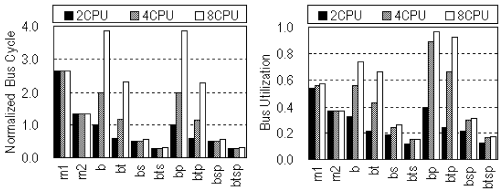

図 21 loop1 におけるシステムバスサイクル数とバス使用率
Fig. 21 Bus cycles and utilization on loop1 execution.

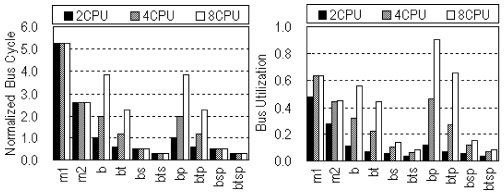

図 22 loop2 におけるシステムバスサイクル数とバス使用率
Fig. 22 Bus cycles and utilization on loop1 execution.

表 9 システムバスサイクルの削減率
Table 9 Reduction ratio of system bus cycles.

	p	b 対 bt	b 対 bs	b 対 bts
loop1	2	0.57	0.50	0.29
	4	0.57	0.25	0.14
	8	0.59	0.14	0.08
loop2	2	0.57	0.50	0.28
	4	0.57	0.25	0.14
	8	0.60	0.13	0.08

総システムバスサイクル数を 13%〜50%に削減した．これは，SHT が無駄なフィルとフラッシュを削減した効果である．つまり，バストラフィックの削減には，キャッシュフィルのレイテンシを削減するよりも，無駄なフィルとフラッシュの削減がより効果的なことが分かる．結果として，TATB と SHT の組合せはバストラフィックを 8%〜29%に削減させた．また，図 21 と図 22 から，b のバス使用率は平均で 44%（最大で 74%）に達したが，btsp のそれは平均で 11%（最大で 17%）に改善したことが分かる．

表 8 にあるように，bp では loop1 の並列度 4 と 8，および，loop2 の並列度 8 で t_{pov} が増大（$\hat{t}_{pov}$ が t_{pd} を大きく上回った）した．btp では loop1 の並列度 8 で t_{pov} の増大が観測された．図 21 と図 22 から，bp では loop1 の並列度 4〜8 にかけてバス使用率が 89%〜97%に達し，loop2 では並列度 8 で 91%に達した．btp では loop1 の並列度 8 でバス使用率が 92%に達した．つまり，システムバスの競合が頻発しており，そのために t_{pov} が増大したと推測される．一方で，btsp のバス占有度は，loop1 の並列度 8 における 17%が最大であり，t_{pov} が増大しなかった．

また，m1 では依存データと同期フラグの更新時に，

表 10 ハードウェア量に関する検討
Table 10 Estimation of hardware count.

gr_{tnb}	tnb_{pro}	tnb_{org}	nb_{gcd}	nb_{gcc}	nb_{tcd}	nb_{tcc}	ns_{org}	ns_{gc}	ns_{tc}	nw_{gc}	nw_{tc}
0.0078	143,968	142,848	23	256	58	256	128	120	1	4	32

m2 では同期フラグの更新時に一貫性制御が実行されるが，並列度に依存せずそれらの更新数は一定なので総バスサイクル数も一定になった．しかしながら，btsp の総バスサイクル数はループと並列度にわたった平均で m1 の約 1/14，m2 の約 1/7 であった．

5. ハードウェア規模

TSVM キャッシュと一般変数キャッシュの割合は，適用分野のプログラムがどの程度の量の同期共有変数を含むか，ハードウェア量の増加をどこまで許すか等の複数の要因に依存する．最適な割合を求めることは本論文の範囲外であり，これ以上の議論はしないが，今回設計したデータキャッシュに関して，式 (1)〜(3) を用いてハードウェア量を一次近似的に見積もることは可能である．用いたパラメータと総ビット数の増加率を**表 10** に示す．

さらに，今回導入したミスペナルティ隠蔽機構のハードウェア量についてもビット数で検討する．TATB（図 6 (a)）の幅は，ディレクトリ部で 7 ビット（tid + thid + v ビット），コンテンツ部で 64 ビットであり，深さは 4 ビットなので，TATB の総ビット数は 284 である．SHT（図 6 (b)）に関して，CAM_0 の幅が，ディレクトリ部で 33 ビット（アドレス + v ビット），コンテンツ部で 314 ビットであり，深さが 4 なので CAM_0 の総ビット数は 1,388 である．さらに，CAM_1 の幅が 33 ビット（アドレス + v ビット），深さは 4 なので，CAM_1 の総ビット数は 132 である．したがって，SHT の総ビット数は 1,520 になる．プリフェッチ機構に関して，プリフェッチ発行キューの幅は 26 ビット（タグ 16 ビット + ワードアドレス 3 ビット + バスコマンド 4 ビット + プリフェッチバッファの添え字 2 ビット + v ビット）であり，プリフェッチバッファの幅は 315 ビット（1 ライン + v ビット）である．ともに深さは 4 なので，プリフェッチ機構の総ビット数は 1,364 である．したがって，導入機構の総ビット数は，TATB の総ビット数 284 と SHT の総ビット数 1,520，および，プリフェッチ機構の総ビット数 1,364 を合わせて 3,168 となる．既存データキャッシュに対して TSVM キャッシュと 3 つのミスペナルティ隠蔽機構を合わせた総ビット数の増加率は，表 10 の結果を用いると $((tnb_{pro} + 3168) - tnb_{org})/tnb_{org} = 0.030$ となる．ちなみに，元の 16 KB のデータキャッシュとは別に 1 KB の TSVM キャッシュを追加した場合（17 KB のデータキャッシュで，そのうち 1 KB が TSVM キャッシュ），総ビット数の増加率は 7%となる．

6. 関 連 研 究

6.1 ミスペナルティ低減手法

TSVM キャッシュは，タスク，および，スレッド単位でエントリを有効利用できるように tid と thid をタグに持つ．このタグを仮想アドレスと見なすと，アドレス変換のレイテンシを削減するために TLB を適用しうる．そこで，従来の TLB を TSVM キャッシュ向けに若干拡張した TATB を導入した．

スヌープキャッシュのミスペナルティ低減手法には read snarfing，software-controlled updating，および，それらの改良版である cache injection がある[10]．これらは無駄なキャッシュフィルを削減して性能向上を図るが，ここでは 2 つの手法を包含し，SHT を導入する動機となった cache injection のみを取り上げる．cache injection は，あらかじめミスするであろう共有変数領域をプロセッサ内部の injection table に登録しておく．そして，スヌープコントローラは，バス上を流れるラインが injection table 上でヒットした場合，そのラインをキャッシュに取り込む．つまり，複数のプロセッサが発生する同一ラインのフィルをあるプロセッサが実施する 1 回のフィルへと削減し，ミスペナルティを低減させる．また，injection table とその制御を追加するだけであり，一貫性プロトコルを変更する必要はない．ただし，injection table を管理する専用命令（共有変数領域の登録と開放）をコンパイラ，または，プログラマが挿入する必要がある☆．そこで，cache injection をベースに特殊な命令が必要ない SHT を導入した．SHT は他 TSVM キャッシュが最近行ったキャッシュフィル（フラッシュ）の履歴を保持し，自 TSVM キャッシュミス時に，ラインをメモリではなく，なるべく他がすでにフィルしたラインが存在するであろう SHT から取得することでミスペナルティの低減を図る．特に，マイクロスレッド群は密

☆ 取り込むべきラインがバス上で観測される前に，injection table にアドレスが登録されるよう注意深く挿入する必要がある．

な（中〜近細粒度）並列処理を実行するので，各マイクロスレッドが局所的に同一ライン（同一の同期共有変数群）でミスする可能性が高いであろうから，SHT が有効に機能するであろう．

一方，プリフェッチはラインフィルのオーバヘッドを隠蔽することによってミスペナルティの低減を図る．プリフェッチには，専用命令を要するソフトウェア・プリフェッチと，要しないハードウェア・プリフェッチがある[9]．ここでは，実現の容易性やハードウェア量の抑制を考慮してソフトウェア・プリフェッチを選択した．そして，2 つの簡単なハードウェア（バストラフィックを増大させ性能に悪影響を与える可能性がある無駄なプリフェッチを削減する機構と，同期通信を阻害しないように一貫性制御とラインフィルを並列して実行するための機構）を付加した．bundling[11] は，既存スヌープキャッシュに追加することで，プリフェッチのバストラフィックとスヌープのルックアップ回数を削減できるが，ハードウェアプリフェッチがベースであり，さらに外部メモリに特殊な 1 ビットを要するため採用しなかった．また，今回検討しなかったが，文献 12) で示される共有データの並べ替えやプリフェッチ挿入アルゴリズムはコンパイラの支援によるミスペナルティ削減手法であり，ここでの導入機構とは相補的な関係にある．

6.2 同期付きキャッシュメモリ

共有変数を用いた効率的な同期通信を実現するために，キャッシュの一貫性制御と同期通信を統合する研究が行われてきた．

一般的に相互排除は共有変数に対するロック/アンロックで実現される．そのような相互排除を効率的に実現できるキャッシュメモリ[18),19)] が提案されている．TSVM キャッシュは同期共有変数を用いて，条件同期（生産者–消費者間同期通信），相互排除，および，バリア同期を統一的でかつ容易に実現できる[6)〜8)]．

他の同期付きキャッシュに I-structure キャッシュがある[17),20),21)]．I-structure キャッシュでも，1 対 1，および，1 対多の条件同期を実現できる．TSVM キャッシュは，さらに，相互排除とバリア同期も簡単に（数命令で）実現でき，それらは従来型キャッシュや I-structure キャッシュと比較して大幅な性能向上を得た[6),8)]．さらに，TSVM キャッシュではカウンタの内容は自動で減ぜられ，同期通信の完了後に暗黙的に empty になる．つまり，ループによって繰り返し使用される同期共有変数を容易に再利用できる．一方で I-structure キャッシュでは，full/empty ビットは暗黙的にリセットされずに，ページ単位でビットの意味を反転させることにより再利用される．正しく並列処理を行うには意味を反転した次繰返しの実行前に現繰返しの完了を保障する同期が必要となる．この同期のオーバヘッドはループ展開を適用すれば相対的に小さくなるが，同期共有変数のリネーミングを必要とし，使用する同期共有変数の増大につながる．ゆえに，TSVM キャッシュは I-structure キャッシュに対して，同期共有変数の使用量を抑えながらより良好な性能向上を得た[6)]．

一般的に，I-structure キャッシュは外部に同期共有変数と等価な full/empty ビットを備えた実メモリを用いるが，TSVM キャッシュは外部に特殊なメモリを必要とせず，TSVM キャッシュを搭載した CMP を既存システムに容易に適用できる．その代わり，TSVM キャッシュではミス時にタグをアドレスに変換し，キャッシュ–メモリ間で構造体を転送する必要がある．そのオーバヘッドを今回の導入機構によって隠蔽できた．

I-structure キャッシュである MISC[17)] は，all-read，all-write，それらを組み合わせた all-read-write 等のプロトコルを提供する．それらのプロトコルは，それぞれ，共有バス上で観測される読み出し，書き込み，および，その両方をキャッシュに積極的に取り込む．そして，同期共有変数を用いた doacross ループの並列化では all-read-write プロトコルの性能が最も良いと結論付けられている．TSVM キャッシュは，書き込み，および，読み出しともにブロッキングされなければ，書き込みではデータとカウンタが，読み出しではカウンタの内容が更新される．それらの一貫性を保つために書き込みと読み出しの成功時には更新されたラインが一貫性バス上に出力される．その結果，他の TSVM キャッシュは一貫性バス上のラインを，ミス時も含めて，取り込むため all-read-write プロトコルと等価である．MISC は，満杯時に，他キャッシュがフィルするラインを取り込まない．TSVM キャッシュは，本体が満杯であったとしても SHT が他キャッシュのフィルを取り込み，無駄なラインフィルの削減を図る．さらに，プリフェッチによりミスペナルティの隠蔽も可能である．MISC では無関係なキャッシュ間でラインを取り込まないように，プロセッサごとにキャッシュをグループ化できるが，TSVM キャッシュは，gr とタグの組合せで，タスク，および，スレッドごとにグループ化できる[5)]．毎回同期のチェックを行うことは非効率的なので，MISC は同期のチェックを行うプロトコルと行わないプロトコルを提供する．TSVM キャッシュは，ブロックされずに値を読み出すだけの非ブロック読み出し[5)] を提供する．これは，カウンタが減ぜられないので，一貫性制御を発生しない．つまり，ブロッ

クをともなった書き込み，および，読み出し（RXLT や RXST 等）との組合せによって一貫性制御による同期通信を必要最低限にとどめおくことが可能である．

JUMP-1 の I-structure キャッシュ[20] はプリフェッチ機構を持つが，その効果は明らかにされていない．また，文献 21) の I-structure キャッシュは，データフロー計算機に特化しており，さらにプリフェッチ等のミスペナルティ隠蔽機構を持たない．

7. むすび

TSVM キャッシュは CMP において，プロセッサコア間の同期通信を一貫性制御と同時に実現することで効率的な並列処理の実現を目指した同期付きキャッシュメモリである．本論文では，TSVM キャッシュに対してミスペナルティ隠蔽機構の導入を試み，同期と通信の融合に加えて，さらなる性能向上を目指した．導入機構は，過去行われたタグ・アドレス変換の結果を再利用する TATB と，他 TSVM キャッシュのラインフィル/フラッシュを保持することによって自ら発生する同一ラインへのそれらを削減する SHT，ならびに，ソフトウェア・プリフェッチ機構である．

2 つの粒度が異なる doacross ループを用いた実験を通して，TSVM キャッシュの従来型キャッシュに対する速度向上は平均で 1.11 倍（最大で 1.58 倍），バス使用率が平均で 44%（最大で 74%）だったが，導入機構は速度向上を平均で 1.91 倍（最大で 2.77 倍），バス使用率を平均で 11%（最大で 17%）に改善させた．様々な粒度を持った並列処理に対し，CMP 上でミスペナルティ隠蔽機構を持った TSVM キャッシュは，性能向上とバストラフィックの削減に有効であることが見込まれた．

また，それらの性能向上はミスペナルティではなく同期待ちがリミットであり，従来の同期付きキャッシュに対し，TSVM キャッシュでは比較的大きいと考えられるミスペナルティを隠蔽できた．さらに，同期付きキャッシュを用いた場合，並列度の増加にともなって同期通信のオーバヘッドが増加するとプリフェッチの遅れに対する耐性が高まることも定量的に確認された．

実験では 16 KB のデータキャッシュに 1 KB の TSVM キャッシュを導入した．簡単な見積りではあるが，ハードウェア量は 3%程度の増加と見込まれた．つまり，TSVM キャッシュを導入するにあたり，既存のハードウェア量の増加を抑えるために TSVM キャッシュは小容量とし，その結果被る不利益（競合性ミスや容量性ミス等）は今回導入したミスペナルティ隠蔽機構で解決を図る方策に対する有効性も見込まれた．

今後の課題として，より多くのプログラムを用いて，TSVM キャッシュの性能や連想度，置換えアルゴリズム，および，導入機構の深さ等について評価する．また，同期共有変数に対する効率的なプリフェッチ命令の挿入方法についても検討する．さらに，ハードウェア量と速度のより詳細な検討も行う．なお，今回導入したミスペナルティ隠蔽機構と既存機構との評価，および，スヌープ・キャッシュに関する様々な高速化手法と TSVM キャッシュとの比較も今後の課題としてあげる．

参考文献

1) Austin, T., Blaauw, D., Mahike, S. and Mudge, T.: Mobile Supercomputers, *IEEE Computer*, Vol.37, No.5, pp.82–84 (2004).
2) Codrescu, L., Wills, D.S. and Meindln, J.: Architecture of the Atlas Chip-Multiprocessor: Dynamically Parallelizing Irregular Applications, *IEEE Trans. Comput.*, Vol.50, No.1, pp.67–82 (2001).
3) Kimura, K., Kodaka, T., Obata, M. and Kasahara, H.: Multigrain Parallel Processing on Compiler Cooperative OSCAR Chip Multiprocessor Architecture, *IEICE Trans. Electronics*, Vol.E86-C, No.4, pp.570–579 (2003).
4) Hammond, L., Hubbert, B., Siu, M., Prabhu, M., Chen, M. and Olukotun, K.: The Stanford Hydra CMP, *IEEE MICRO Magazine*, Vol.20, No.2, pp.71–84 (2000).
5) Yamawaki, A. and Iwane, M.: Organization of Shared Memory with Synchronization on Multiprocessor-on-a-chip, *Proc. 9th Int. Conf. on Parallel and Distributed Systems*, pp.83–91 (2002).
6) 山脇　彰，岩根雅彦：同期共有変数キャッシュの同期状態を拡張した効果，信学論，Vol.J87-D-I, No.12, pp.1136–1139 (2004).
7) Yamawaki, A. and Iwane, M.: Evaluation of Mechanisms Introduced to Improve Performance of TSVM Cache, *Proc. Int. Conf. on Parallel and Distributed Computing and Networks*, pp.502–507 (2004).
8) Yamawaki, A. and Iwane, M.: Implementation and Evaluation of Novel Cache Architecture for Communicating Shared Variable with Synchronization on a SOC-multiprocessor, *The 1st Workshop on Embedded Parallel Architectures*, pp.1–6 (2004).
9) Vanderwiel, S.P.: Data Prefetch Mechanisms, *ACM Computing Surveys*, Vol.32, No.2, pp.174–199 (2000).
10) Milenkovic, A.: Achieving High Performance

in Bus-Based Shared-Memory Multiprocessors, *IEEE Concurrency*, Vol.8, No.3, pp.36–44 (2000).
11) Wallin, D. and Hagersten, E.: Bundling: Reducing the Overhead of Multiprocessor Prefetchers, *Proc. 18th Int. Symp. on Parallel and Distributed Processing*, pp.74–83 (2004).
12) Tullsen, D.M. and Eggers, S.J.: Effective Cache Prefetching on Bus-Based Multiprocessors, *ACM Trans. Comput. Syst.*, Vol.13, No.1, pp.57–88 (1995).
13) Nuth, P.R. and Dally, W.: The named state register file: Implementation and performance, *Proc. 1st Int. Symp. in High-Performance Computer Architecture*, pp.4–13 (1995).
14) Hennesy, J.L. and Patterson, D.A.: *Computer Architecture: A Quantitative Approach, 3rd Edition*, Morgan Kaufmann (2003).
15) Yamawaki, A., Chayamichi, H. and Iwane, M.: Easily Customizable Open Soft Processor Cores, *The 2004 1st IEEE Technical Exhibition Based Conference on Robotics and Automation*, pp.79–80 (2004).
16) Motorola: *G2 PowerPC Core Reference Manual Rev.1*, Motorola. Inc (2003).
17) 松本 尚：スヌープキャッシュ制御機構の DOACROSS ループへの適用，情報処理学会論文誌，Vol.34, No.4, pp.616–627 (1993).
18) Tarui, T., Nakagawa, T., Ido, N., Asaie, M. and Sugie, M.: Evaluation of the Lock Mechanism in a Snooping Cache, *Proc. 6th Int. Conf. on Supercomputing*, pp.53–62 (1992).
19) Ramachandran, U. and Lee, J.: Cache-Based Synchronization in Shared Memory Multiprocessors, *Journal of Parallel and Distributed Computing*, Vol.32, No.2, pp.11–27 (1996).
20) Goshima, M., Mori, S., Nakashima, H. and Tomita, S.: The Intelligent Cache Controller of a Massively Parallel Processor JUMP-1, *Proc. Intl. Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*, pp.116–124 (1997).
21) Kavi, K.M. and Hurson, A.: Design of cache memories for dataflow architecture, *Journal of Systems Architecture*, Vol.44, pp.657–674 (1998).

(平成 17 年 6 月 17 日受付)
(平成 17 年 11 月 1 日採録)

山脇 彰（正会員）
1974 年生．1999 年九州工業大学大学院工学研究科電気工学専攻博士前期課程修了．同年三菱電機（株）に入社．2000 年より九州工業大学助手．デジタル回路，システム LSI アーキテクチャ，リコンフィギャラブルコンピューティングの研究に従事．電子情報通信学会会員．

岩根 雅彦（正会員）
1946 年生．1968 年京都大学工学部数理工学科卒業．同年（株）東芝に入社．1988 年より九州工業大学教授．工学博士．計算機アーキテクチャの研究に従事．電子情報通信学会，IEEE Computer Society 各会員．