計算機アーキテクチャ 149－23
（２００２．８．２３）

# 共有変数の同期を考慮したキャッシュ構成とその予備実験

山脇 彰† 岩根雅彦†

共有メモリ型マルチプロセッサにおけるマルチスレッド実行環境では，複数のスレッドが共有変数を介して通信と同期を行う．TSVM は共有変数へのアクセスと同時に同期を行う論理的な構造化メモリである．物理 TSVM はマルチプロセッサオンチップ (MOC) において，TSVM キャッシュ(TC) と通常のメモリによって実現される．CPU コアの L1 キャッシュは，データキャッシュを TC と一般変数キャッシュ(GVC) に分離したキャッシュアーキテクチャである．基礎実験として，スタンドアロンの TSVM を搭載した実機での実測結果から TC によって性能向上が見込まれた．

# Organization and Preliminary Experiment for Cache Considering Synchronization of Shared Variable

Akira Yamawaki † and Masahiko Iwane†

In a multithreaded environment, an inter-thread communication and a synchronization are performed through a shared variable in a shared memory. TSVM is a structured memory with synchronization that performs an inter-thread comminucation and a synchronization simultaneously. In Multiprocessor-On-a-Chip(MOC), the physical TSVM consists of TSVM Cache(TC) and a conventional memory. CPU core has L1 cache including TC, General Variable Cache(GVC), and instruction cache. This cache architecture is the one that devides a conventional data cache into TC and GVC. As preliminary experiment, we estimate performance of TC using the measurement data on MTA/TSVM with the stand-alone TSVM. The estimation shows that TC achieves a good performance.

## 1. はじめに

共有メモリ型マルチプロセッサにおけるマルチスレッド実行環境では，複数のスレッドが共有変数を介して通信と同期をする．このような並列処理の効率化を目的として，同期機能をもった論理的な構造化メモリが提案されており，共有メモリに full/empty ビットを付加した I-structure[2] や write once variable[3]，共有メモリにカウンタを付加し，記憶領域を CAM 化してエントリの動的な配置を可能にした TCSM[1] がある．更に，TCSM を一般的なメモリと一元化した TSVM(Tagged shared variable memory)[4] がある．

ここでは，オンチップマルチプロセッサ (MOC)[5),6),8)] において，TSVM を通常のメモリ (DRAM,SRAM) と TSVM キャッシュ(TC) によって実現する共有メモリ構成を提案する．プロセッサの L1 キャッシュは，同期機能をもった共有変数 (SV) をキャッシングする TSVM キャッシュ(TC) と，一般変数をキャッシングする一般変数キャッシュ(GVC)，および命令キャッシュからなる．このキャッシュアーキテクチャ(IYA) は従来のデータキャッシュを TC と GVC に分離したキャッシュアーキテクチャである．

I–structure キャッシュが検討または実装されているが[9),10)]，外部に full/empty ビットを持ったメモリを必要とする．それに対し，MOC ではメモリが通常のメモリであり，SV を識別するタグはアドレス変換機構によりメモリアドレスに変換されるため，外部に特殊な機構を必要としない．

本論文では対象とするプログラミングモデルについて述べ，論理 TSVM と共有変数の概念を示す．そして，IYA を採用した MOC と物理 TSVM の構成を示し，物理 TSVM の動作について説明する．最後に基礎実験として，スタンドアロンの TSVM を搭載した実機による測定結果をもとに TC の性能を見積もる．

## 2. プログラミングモデル

### 2.1 マルチスレッド実行環境

マルチスレッドでは，タスクはプログラムの実行環境であり，タスクに属するスレッド群はそのタスクに

† 九州工業大学 工学部 電気工学科
Depertment of Electronic Engineering, Faculty of Engineering, Kyushu Institute of Technology

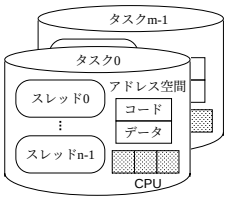


**図 1**　マルチスレッド実行環境

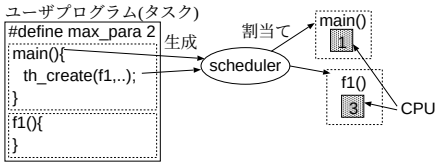


**図 2**　プログラミングと実行モデル

保護された資源を共有しながら協調動作する．その概念を**図 1** に示す．スレッドはユーザによって指定される任意長の命令実行である．図 1 では 1 つの CPU が 1 つのタスクに属する表現になっているが，1 つの CPU が複数のタスクに属すことも許す．

### 2.2　プログラミングと実行モデル

ユーザスレッドは**図 2** のように関数 (f1) として指定され，main 関数は初期スレッドとなる．

タスクの最大並列度 (CPU 数) がプログラムの先頭で宣言され (#define max_para 2)，タスクの生成時にその並列度分の CPU が確保される．ユーザスレッドは生成手続き (th_create) によって生成され，同時にそのスレッドを実行する CPU が同一タスクに属する CPU 群から選択される．生成されたスレッドはスケジューラが管理するキューに登録される．スレッドは選択された CPU に動的に割り当てられ，消滅するまでその CPU に割り当てられ続ける．スレッドはサスペンド (I/O 待ち，スリープ) や割り込み (プロセッサ間，タイマ) により切り替わる．各スレッドは，共有するアドレス空間のデータ領域にマッピングされた TSVM を介して同期と通信を同時に実行する．

### 2.3　マイクロスレッドの利用

スレッドの粒度が文レベル程度に小さい場合，スレッドの生成や動的な割当てのオーバヘッドを無視できない．その際に，スレッドを文レベル以上で並列化し，その並列化したコード列を仮想的な CPU であるマイクロスレッドに静的に割り当てる．その概念を**図 3** に示す．main 関数の並列度はプログラムの先頭で (#define main_para 2)，ユーザスレッドの並列度は生成手続きの引数で指定される．これらの並列度はタスクの最大並列度以下にする．

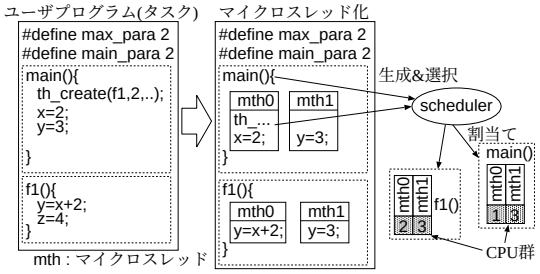


**図 3**　マイクロスレッドの導入

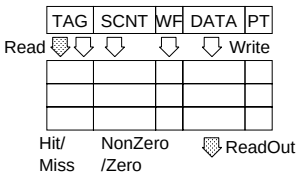


**図 4**　Concept of TSVM

スレッドの生成時にそのスレッドを実行する CPU 群 (同一タスク内) が決定される．このとき，1 つの CPU が複数のスレッドに属することも許す．スレッドは CPU 群単位で割り当てられ，1 つのマイクロスレッドを同じ 1 台の CPU が実行し続ける．スレッドは消滅まで同一の CPU 群に割り当てられ続け，切り替え時には全マイクロスレッドが切り替えられる．マイクロスレッドもスレッドで局所的な TSVM を介して同期通信する．

### 2.4　変数の概念

タスク内のスレッドが共有するアドレス空間上の変数に対し，異なるスレッドやマイクロスレッド間で定義および参照される変数を共有変数とよぶ．そして，共有変数のうち通信に同期を必要とするものを同期変数 (SV) とよぶ．SV は TSVM に割り当てられ，アクセスと同時に同期も実行される．SV のうち一時的な通信に用いる変数を一時変数，その他を恒久変数とよぶ．一時変数を同期通信の完了後に自動的に消滅させ，エントリの有効利用を図る．また，SV 以外の変数を一般変数 (GV) とよぶ．

### 2.5　論理 TSVM

#### 2.5.1　論理 TSVM 構成

TSVM は記憶領域の動的な配置機能と同期機能を持った論理的な構造化メモリであり，その概念を**図 4** に示す．TSVM は CAM によって構成され，各エントリは，検索のためのタグ (TAG)，同期回数 (SCNT)，当該エントリに対する読出し完了の待合せフラグ (WF)，一時変数か恒久変数かを示すフラグ (PT)，SV の内容 (DATA) のフィールドをもつ．PT が 1 で

恒久変数を，PTが0で一時変数を意味する．TAGをTIDとスレッド識別子(THID)，およびSVIDの連結とし，タスク，スレッドごとにエントリを保護する．

#### 2.5.2 論理TSVMの基本動作

基本動作は書込みと読出しおよびリセットであり，TAGの一致検索によりエントリが指定される．

書込みでは，TSVMはSCNTが非0のエントリへの書き込みをブロックし，そのようなエントリが存在しないのであれば空きエントリ(SCNTとPTが0)に書き込む．

読出しでは，TSVMはSCNTが非0のエントリからDATAを出力してSCNTを1減じ，そのようなエントリが存在しないのであれば読出しをブロックする．また，WFが1のエントリに対する読み出しは，すべての読み出しが完了する(SCNTが0になる)までブロックされる(待ち)．更に，SCNTに関係なく値だけを読み出す(非ブロック読み出し)ことも可能である．

ブロックと待ちによってスレッドは切り替わらない．書き込みブロックおよび待ちは同一TAGによる読み出し後にSCNTが0になったら解除され，読み出しブロックは同一TAGによる書き込み後に解除される．

リセット動作では，TAGで指定されたエントリを空にする．また，TAGの任意のビットをマスクし，複数のエントリを一度に解放できる．

### 2.6 オブジェクトファイルの概要

一般的なオブジェクトファイル[12]に対し，基本的にスレッド間で使用されるSV(GSV:Global SV)の領域，および，物理TSVMのアドレス変換機構で使用されるTAGとメモリアドレスの対応表(TAT:Tag to Address Table)が追加される．マイクロスレッドの適用時では，各スレッドのコードはマイクロスレッドごとに分割され，それらのマイクロスレッド間で使用されるSV(LSV:Local SV)の領域も追加される．各SVはTSVMの各フィールドを持った構造体と等価であり，それらのフィールドは初期化される．また，TAGのTHIDが0のSVをGSVとしLSVと区別するため，スレッドの識別子は1以上にする必要がある．

## 3. 物理TSVM

### 3.1 概　　要

物理TSVMはTSVM Cache(TC)と通常のメモリからなり，その概念を**図5**に示す．

プログラム実行時の主メモリアクセス削減を図るため，タスク生成時に使用される全GSVをそのタスクに属するTC(CPU)群にローディングする．また，マイクロスレッド適用時には，スレッドごとにもTCが

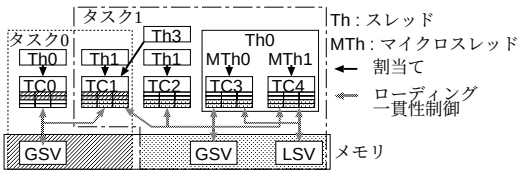


**図5**　TSVMキャッシュの概要

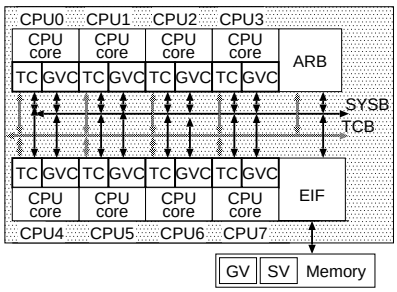


**図6**　IYAを採用したMOCの構成

確保され，スレッドの最初の実行時にそのTC群に全LSVがローディングされる．このときTCに空きエントリが存在しないならば，TCは主メモリにSVを書き戻して空きエントリを作る．SVを介した同期通信はTCの一貫性制御と同時に暗黙的に実行される．一貫性制御はGSVでは同一タスクに，LSVでは同一スレッドに属するTCにのみ反映する．

TCは主メモリとの置き換えを1エントリ単位で行いfalse sharingの削減も図る．文献[13]によれば，1ラインが4バイトのキャッシュでは，容量が1KB以上で90%から98%のヒット率を達成している．TCではメモリは単なるバックアップと考える．

### 3.2 マルチプロセッサオンチップ構成

IYAを採用したMOCは**図6**のように，8台のCPUを搭載し，そのデータキャッシュはTCとGVCからなる．TCはCPU間の同期通信機能を提供し，SVはメモリ上で構造体として格納される．また，集中型のバスアービタ(ARB)と外部とのインターフェース(EIF)も搭載される．MOCは外部に特殊な構造を持ったメモリを必要としないため，CPUコアとEIFを変更することで様々なシステムに接続できる．

### 3.3 物理TSVM構成

物理TSVMは基本的にTC，メモリ上のSVおよびTAGをアドレスに変換する機構からなり，その構成を**図7**に示す．TCは論理TSVMに対してデータのサイズを表すSIZE，エントリとメモリの置き換えに使用されるAC(ACcess flag)，エントリを置き換え禁止にするフラグNR(Not Replace)を付加した構成である．MASKはTAGの任意のビットをマスクする

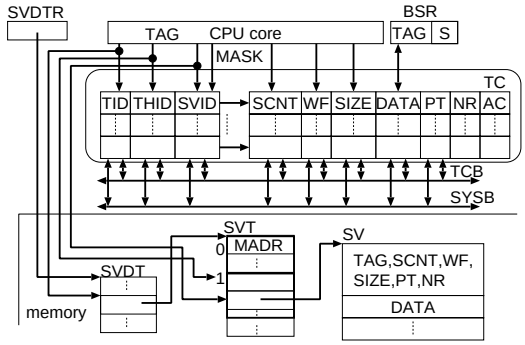


**図 7**　物理 TSVM 構成

**表 1**　物理 TSVM に関する命令のフォーマット

| opcode | operand |
|---|---|
| STSVM | Reg(DATA), TAG, CNT, WF |
| LTSVM | Reg(DATA), TAG |
| NBLTSVM | Reg(DATA), TAG |
| BLTSVM | TAG |
| RTSVM | Reg(MASK), TAG |

ためのビットパタンである．

アドレス変換機構において，SVDT(SV Discriptor Table) はシステム中に 1 つだけ存在し，SVT(SV Table) はタスクごとに 1 つずつ存在するテーブルである．SVDTR(SVDT Register) は SVDT のベースアドレスを格納する．SVDT の各エントリは SVT のベースアドレスを格納し，TID により指定される．SVT は固定長のフレームに分割され，フレームは THID により指定される．0 番目 (THID が 0) のフレームが GSV のメモリアドレス (MADR) を格納し，1 番目以降 (THID が 1 以上) のフレームが LSV の MADR を格納する．そして，SVID がフレーム内オフセットとなる．アドレス変換機構は一般的なページング機構と同様であり TLB 化可能である．

BSR(Block Status Register) は TSVM のブロッキングとその解除を実現するためのレジスタであり，ブロック時の TAG と状態 (S) を格納する．S が 01 で書き込み，10 で読み出しブロックを，11 で待ちを表す．

### 3.4　物理 TSVM 命令

TSVM 用機械命令のフォーマットを**表 1** に示す．STSVM は書き込み動作を行う命令で，TAG で指定した TC のエントリに CNT，WF，Reg にある DATA を書き込む．このとき，レジスタの種類によって SIZE の値が自動的に決定される．LTSVM は読み出し動作を，NBLTSVM は非ブロック読み出し動作を行う命令で，TAG で指定した SV の DATA を TC から Reg に読み出す．BTSVM は TAG で指定した SV を同一タスクまたはスレッドに属する TC にローディングする命令であり，プログラムの実行前に使用される SV をローディングするために実行される．RTSVM はエントリの削除を行う命令で，Reg のマスクデータで TAG をマスクし，TC の TAG と一致したすべてのエントリを 0 にリセットし，解放する．

### 3.5　置き換えアルゴリズム

TC はアクセスされると入力された TAG により一致検索を行う．その結果，一致したエントリが存在するならば TC ヒット，そうでないならば TC ミスとよぶ．TC ミス時に TC はメモリ上の SV をローディングし，満杯ならばあるエントリと置き換える．ただし，NR が 1 のエントリは置き換え対象にならない．

置き換えるエントリの選択は，AC の値とランダム値を組み合わせた擬似的なワーキングセット法により行われる．AC の初期値は 0 であり，エントリがアクセスされると自動的に 1 になる．そして，ある時間の経過後に全エントリの AC は 0 にリセットされる．メモリへ書き戻すエントリは AC が 0 のエントリからランダムに選択し，全エントリの AC が 1 ならば，全エントリからランダムに選択する．

## 4.　物理 TSVM の動作概要

### 4.1　スタートアップ時の設定

システムの起動では，**図 8**(a) のように特定の CPU が SVDT を作成し，他の CPU は SVDTR にその SVDT のベースアドレスを書き込む．

タスクの生成では，図 8(b) のように，特定の CPU(CPU1) が指定された最大並列度分の CPU を記録する．そして TAT を元に SVT を作成し，SVDT の TID 番目のエントリにその SVT のベースアドレスを書き込む．その後，main 関数をキューに登録する．記録された CPU 群は，TAG の THID と SVID が 0 で NR が 1 の SV を TC に NBLTSVM によってローディングする．この SV は TC がどのタスクに属するかを識別するために使用される．これにより TC を異なるタスクで多重化する (図 8(c))．

マイクロスレッド適用時では，スレッドの生成時にもそのスレッドに属する CPU 群は TAG の SVID が 0 で NR が 1 の SV を TC にローディングする．この SV は TC がどのスレッドに属するかを識別するために使用される．これにより異なるスレッドで TC を多重化する．

### 4.2　SV のローディング

タスクで使用される全 GSV に対する BLTSVM 列が main 関数の先頭に付加される．初期スレッドを実行する CPU は BLTSVM 列によりメモリ上の GSV

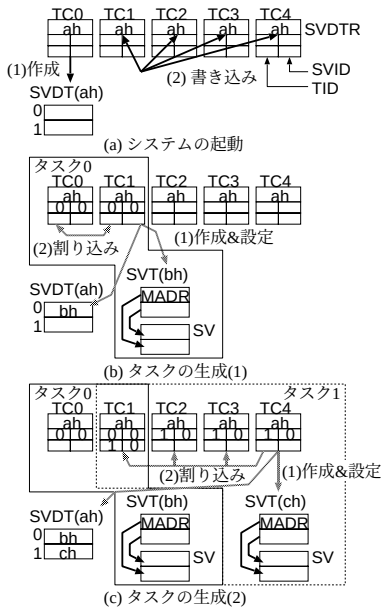


**図 8**　スタートアップ時の動作

を TC にローディングし，その GSV を TCB に出力する．TCB 上の GSV と同一の TID と THID(0) を NR が 1 のエントリにもつ (同一タスクに属する)TC は TCB 上の GSV をローディングする．マイクロスレッド適用時では特定のマイクロスレッドの先頭に LSV に対する BLTSVM 列が付加され，スレッドの最初の実行時に LSV が，同一の TID と THID(1 以上) を NR が 1 のエントリにもつ (同一スレッドに属する)TC にローディングされる．SV は最優先度の空きエントリにローディングされ，空きがなければ置き換えられる．

### 4.3　書き込みおよび読み出し動作

STSVM，LTSVM，NBLTSVM では TC ミスならば最優先度の空きエントリかメモリに書き出したエントリに SV をローディングする．

STSVM の場合，SCNT が非 0 ならば書き込みはブロッキングされる．ブロッキングでは基本的にパイプラインがストールされるが外部割り込みは受付可能とする．このとき BSR に TAG と 01 を書き込む．SCNT が 0 ならば当該エントリにデータを書き込み，TCB にそのエントリの SV を出力する．

LTSVM の場合，SCNT が 0 ならばブロッキングする．このとき，BSR に TAG と 10 を書き込む．SCNT が非 0 ならばエントリの SCNT を 1 減じる．WF が 0 ならば，DATA を CPU に出力し，WF が 1 で SCNT が非 0 ならばブロッキングする．このとき，BSR に TAG と 11 を書き込む．そして，TCB にそのエントリの SV を出力する．NBLTSVM は，当該エントリの DATA を CPU コアに出力するだけである．BSR の内容により，TC は CPU コアをブロックし続ける．

### 4.4　一貫性制御

STSVM の結果が TCB に出力された場合，同一タスクまたはスレッドに属する TC 群は TCB 上の TAG によって TC ヒットしたエントリに TCB 上の SV を書き込む．TC ミスならば最優先度の空きエントリかメモリに書き出したエントリに書き込む．その結果，SCNT が非 0 で BSR と SV の TAG が一致し，S が 10 ならば，BSR をリセットして CPU コアの読み出しブロックを解除する．

LTSVM の結果が TCB に出力された場合，同一タスクまたはスレッドに属する TC 群は TCB 上の TAG によって TC ヒットしたエントリの SCNT を 1 減じる．TC ミスならば，最優先度の空きエントリかメモリに書き出したエントリに TCB 上の SV を書き込む．この結果，SCNT が 0 で BSR と SV の TAG が一致し，S が 01 または 11 ならば，BSR をリセットして CPU コアのブロックを解除する．

通常，TC は CPU コアと TCB で排他的にアクセスされるが，同時にアクセスされた場合は一貫性制御を先に完了させる．また，ある TC が SV をメモリからローディングしている際にその SV と同じ TAG を持った SV が TCB に出力されたならば，TCB 上の SV が最新値のため，その TC はメモリからのローディングを中断し TCB 上の SV をローディングする．

### 4.5　TC の解放

TC の解放は，特定の CPU が RTSVM 命令を実行することによって行われる．RTSVM 命令により TC は解放され，そのときの TAG と MASK を TCB に出力する．同一タスクまたはスレッドに属する TC 群も TCB 上のデータでエントリを解放する．タスクの消滅時には SVID をマスクすることにより，そのタスクで使用されていた全 GSV が解放される．スレッドの消滅時には THID と SVID をマスクすることで，そのスレッドで使用されていた全 LSV が解放される．

### 4.6　スレッドの切り替え

TC によってブロックされているスレッド (CPU) は割り込みによって切り替わる際に BSR の内容をメモリに退避し，再割当て時には退避した内容を BSR に復帰させる．スレッドの退避中にブロック解除の条件が満たされている可能性があるため，再割当て時に BSR の TAG によって TC が一致検索される．TC ミス時にはメモリから SV をローディングする．そして，BSR の S の内容と当該 SV の SCNT の状態により一貫性制御と同様にしてブロックが解除される．

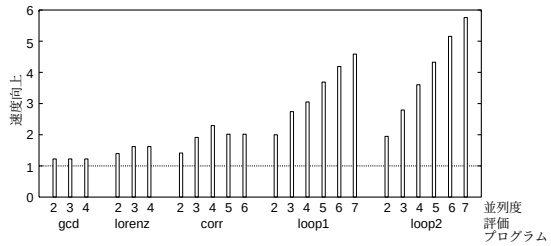


**図 9** 評価プログラムごとの TC の速度向上 (見積もり)

## 5. 基礎実験

### 5.1 実験環境

TC の性能を見積もるために，スタンドアロンの TSVM をもつバス結合共有メモリ型マルチプロセッサ MTA/TSVM を使用した．MTA/TSVM は 8 台の L1 キャッシュをもつ 486DX2，TSVM，バスアービタ，BTT(Block Tag Table)[11]，L2 キャッシュ，および DRAM が共有バスに接続されている．バスアービタは集中型で回転式によりバスを調停する．また，BTT は個々の 486DX2 に対してバスバックオフ機能 (BOFF)[14] により TSVM アクセスのブロッキングを実現する [11]．TSVM をメモリマップ IO として実現しているので，mov 命令によって TSVM 命令を実装している．実験では L1 キャッシュをオンにし，一度プログラムを実行してすべてキャッシュヒットする状態で測定した．また，プログラムの実行中にスレッドの切り替えは起こらない．

#### 5.1.1 性能見積もり

MTA/TSVM における実測結果から見積もった評価プログラムごとの TC の速度向上を**図 9** に示す．使用した評価プログラムは lorenz，corr，loop1，loop2 の 4 つである．lorenz は気象現象をモデル化したローレンツアトラクタを求めるプログラム，corr は相関係数を求めるプログラムでありマイクロスレッドを用いて文レベルで並列化した．また，loop1 と loop2 は doacross ループであり，繰り返しをスレッドに分割した．loop1 は整数演算のみで，loop2 は単精度の浮動小数点演算を含む．予測値は MTA/TSVM のバス動作を解析し，TC として実装した際に発生しないバスアクセス分を削除して求めた．その際に，全 SV が TC に格納されていると仮定した．結果から TC は良好な性能を達成できることが見込まれる．

## 6. 結　　び

MOC での共有変数の同期を考慮した共有メモリ構成と従来のデータキャッシュを TSVM キャッシュと通常変数キャッシュに分離した新しいキャッシュアーキテクチャを示した．スタンドアロンの TSVM を搭載した実機での測定結果をもとに TC の性能を予測した．その結果，TC は良好な性能を達成できることが見込まれた．今後は，実際の TC の開発とともに，TSVM の並列化プログラムとの親和性を検証していく．

**謝辞**　本研究は文部省科学研究費補助金若手研究 (B) (課題番号 14780227) の支援により行った．

## 参 考 文 献

1) 岩根, 田中, 山脇. マルチプロセッサにおける CAM を用いた同期通信用メモリ, 信学論, Vol.J83–D–I–NO.3,pp.317–328, Mar 2000.
2) A.S.Nikhil, R.S.Nikhil, and K.K.Pingali. I–Structure:Data Structures for Parallel Computing, ACM Trans on Prog Lang Sys, Vol. 11, No. 4, pp. 598–632 1989.
3) G.E.Blelloch et.al. Space-Efficiet Scheduling of Parallelism with Synchronization Variables, Proc.of.9th Annual ACM Symp. on Parallel Algorithms and Architectures, pp. 12–33 1997.
4) 山脇, 岩根. マルチプロセッサにおける共有変数用キャッシュ, 情処研報, Vol.2001, No.116, pp.33–38 Nov 2001.
5) R.Barua et.al. Maps : A Compiler-Managed Memory Sytem for Raw Machines, Proc.Int. Symp.on Computer Architecture-26 June 1999.
6) L.Hammond et.al. THE STANFORD HYDRA CMP, IEEE MICRO Magazine, March-April 2000, pp.71–84, Apr 2000.
7) J.Y. Tsai et.al The Superthreaded Processor Architecture, IEEE Trans on Comp, Vol. 48, No. 9, pp. 881–901 1999.
8) L.A.Barroso et.al Pinanha: A Scalable Architecture Based on Single-Chip Multiprocessing, In Proc. of Int. Symp. on Computer Architecture, Jun 2000.
9) K.M.Kavi, and A.R.Hurson. Design of cache memories for dataflow architecture, Journal of Systems Architecture 44, pp.657–674, 1998.
10) M.Coshima et.al. The Intelligent Cache Controller of a Massively Parallel Processor JUMP-1, In Proc. IWIA'97, pp. 116–124, Oct 1997.
11) 山脇, 岩根. 同期通信メモリにおけるカウンタとブロッキングの効果, 信学論, Vol.J84–D–I, NO.9, pp. 1457–1460 Sep 2001.
12) J.R.Levine 著, 榊原他訳. Linkers & loaders, オーム社,2001.
13) A.J.Smith. Cache Memories, Computing Surveys, Vol.14, No.3, pp. 473–530, Sep 1982.
14) Intel Corp. Intel486 Microprocessor and Related Products, Intel Corp, 1995.