計算機アーキテクチャ 145－5
（2001．11．29）

マルチプロセッサにおける共有変数用キャッシュ

山脇 彰† 岩根雅彦†

同期機能を持つ共有変数メモリTSVMのキャッシュであるTSVCを搭載したマルチプロセッサオンチップ(MOC)を提案する．MOCにおいて，TSVMはTSVCと主メモリで実現され，スレッド間の同期通信はTSVCの一貫性制御とともに行われる．タスクの生成と同時に共有変数をTSVCに読込んでおくため，タスクの実行中にPEは共有変数を高速にアクセスできる．MOCの開発にあたり，TSVMのプログラムとの親和性とTSVCの基本機能の性能を検証した．TSVMを用いて並列化したプログラムと同期通信メモリTCSMでのコード量を比較し，TSVCはTCSMに対し最大で26%のコード量を削減でき，最大で1.88の速度向上を得た．

A Cache Architecture for Shared Variables on Multiprocessor

AKIRA YAMAWAKI † and MASAHIKO IWANE†

The Tagged Shared Variable Memory(TSVM) is the concept of a structured memory with the synchronization mechanism, and consists of the Tagged Shared Varibale Cache(TSVC) and a main memory in Multiprocessor-On-a-Chip(MOC). The synchronization and communication between threads are performed simultaneously with consistency control of TSVC. Because a shared variables are loaded to TSVCs in generating a task, a PE in the MOC can access them via TSVC during executing the task. In developmemt of MOC, the adaptivity to a parallel program of TSVM and the validity of the fundamental function of TSVC is compared with TCSM using two programs on multiprocessor MTA/TSVM consists of 4 scalar processors. TSVM can reduce the amount of assembler code of up to 26% for TCSM. The maximum speed up ratio of TSVC to TCSM is 1.88.

1. はじめに

マルチプロセッサオンチップ(MOC)は，プログラム内の命令レベル並列性に加え，スレッドレベル並列性も利用し性能向上を図るマルチスレッド実行を対象としており[1)~6)]，複数スレッドを協調動作させるには同期機構が必要である．スレッド間の同期機構として，共有メモリや共有レジスタに同期機能を付加し，アクセスと同時に同期をとるI-structure[7)]，NSRF[8)]，グローバルレジスタ[9)]，TCSM[6)]などがある．TCSMでは，カウンタにより容易に1対多での同期通信を実現でき，CAMによってエントリの動的な割当てと保護を行う．しかし，TCSMは変数の生存期間を考慮しておらず，通常のメモリとして扱えない．そこで，共有変数の取扱いに注目してTCSMを同期機能を持った共有変数メモリTSVM(Tagged Shared Variable Memory)[10)]に拡張し，それに対するキャッシュTSVC(Tagged Shared Variable Cache)を搭載したMOCを提案する．そのようなMOCにおけるキャッシュは，データ及び命令キャッシュからなるハーバード・アーキテクチャに対し，データキャッシュをローカル変数と共有変数キャッシュに分離した構成となる．

本論文では，プログラムの実行環境及びTSVMとTSVCの概要を示し，TSVCを用いたマルチスレッド実行の概念について述べる．そして，TSVCを搭載したMOCの開発にあたり，TSVMの並列化プログラムとの親和性，及びTSVCの基礎的な機能の有効性を検証する．

† 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

2. 共有変数用キャッシュを搭載したMOC

2.1 プログラムの実行環境

2.1.1 プログラムの実行モデル

MOCにおけるマルチスレッド実行モデルはタスク(TSK)，スレッド(TH)，マイクロスレッド(MT)からなり，その概念図を図1に示す．TSK(初期TH)はプログラムの実行環境であり，その生成(主メモリへのローディング)時にプログラムの実行に必要な資源

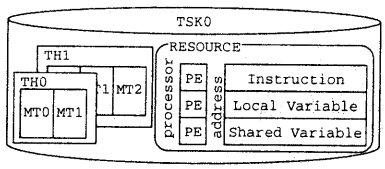

図1　プログラムの実行モデル

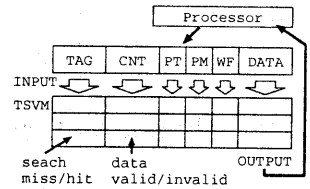

図2　TSVM の概念図

```
恒久変数
 shared perm 変数の型,変数名,CNT,WF
一時変数
 shared temp 変数の型,変数名,CNT,WF
```

図3　共有変数の宣言

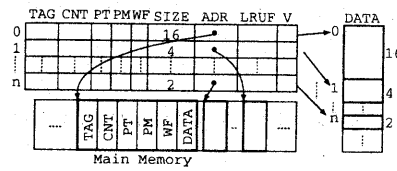

図4　物理 TSVM の構成

が確保される．TH は任意の長さをもつ一連の命令実行であり，プログラム内でユーザによって指定される．TH は基本ブロックやイタレーション，及びサブルーチンであり[1)~5)]，その内部を一般的な並列化手法により MT として静的に分割し，より細かなレベルの並列性も利用する．TSK は 1 つ以上の TH，TH は 1 つ以上の MT からなり，同一 ID をもつ MT は同一の 1 台のプロセッサで実行される．MT が並列化の処理単位，TH がプロセッサ割当のスケジューリング単位で，TH を実行させるには属する MT 数分のプロセッサが必要となる．TH の切替えにおいては，TH に属するすべての MT がサスペンドされる．

2.1.2　アドレス空間

変数に関して，各 TH や MT がローカルに使用するものと，それらが共有して使用するものとに分類し，アドレス空間をローカル変数，共有変数の各領域に分ける．ここでの共有変数とは，プログラムの実行中に異なる MT によって参照または更新される変数を指し，doall で扱う配列のように，MT ごとに分割された変数 (配列要素) に対して異なる MT 間で依存がないならば，共有変数 (配列) としない．共有変数領域には TSVM をマッピングし，複数の MT は TSVM を介して同期通信を行う．共有変数のうち，並列処理過程で一時的な通信に使うものを一時変数，そうでないものを恒久変数として宣言する．恒久変数に関しては，同期通信を行うモードと行わないモードに選択でき，前者を通信モード，後者を通常モードと呼ぶ．

2.2　同期付き共有変数と TSVM

2.2.1　論理 TSVM の概要

TSVM は同期機能を持った共有変数を実現するための論理的な構造化メモリであり，その概念を図 2 に示す．TSVM は記憶領域の動的割当てを実現するために CAM(Content Addressable Memory) で構成され，各エントリは TSVM のエントリを表す共有変数 ID(TAG)，参照回数 (CNT)，恒久変数及び一時変数を示すフラグ (PT)，恒久変数の通信及び通常モードを示すフラグ (PM)，当該エントリの読出し終了を待ち合わせるフラグ (WF)，変数の内容 (DATA) の各フィールドからなる．初期値は全フィールドが 0 であり，TAG は TSVM の一致検索を行うフィールドで，TAG をタスク ID，共有変数 ID の連結としタスクごとに TSVM のエントリを保護する．

共有変数は図 3 のように宣言され，TSVM に割当てられる．恒久変数では shared perm に続き，変数の型，変数名，CNT，PM，WF の順に指定し，一時変数では shared temp に続き，PM を除いて同様であり，恒久変数では PT が 1，一時変数では 0 となる．一時変数に関しては，同期通信完了後に TSVM から消滅させエントリの有効利用を図り，恒久変数に関しては，明示的に開放されない限り TSVM に残す．

2.2.2　物理 TSVM の構成

MOC において TSVM はプロセッサコアが搭載する TSVM キャッシュ(TSVC) と主メモリにより実現され，その構成を図 4 に示す．TSVC は，TSVM の各フィールドに加えて，データ長 (SIZE)，共有変数のメモリ上のアドレス (ADR)，エントリの有効及び無効を表すフラグ (V)，TSVC が満杯時にメモリと置換えるエントリを選択するために使用されるフィールド (LRUF) からなるディレクトリ部と，DATA フィールドを構成するコンテンツ部からなる．DATA の 1 エントリは SIZE で示されるバイト長であり，SIZE の最大値とコンテンツ部の容量は実装依存である．共有変数はメモリ上で TAG，CNT，PT，PM，WF，DATA からなる構造体と等価であり，ADR にはその先頭番地が格納される．ディレクトリ部に V=0 のエントリ

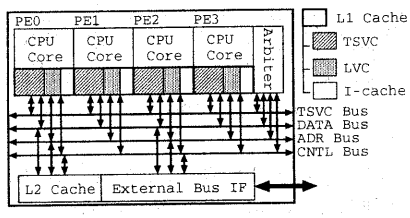

図5　TSVC を持つ MOC の構成

が存在しない，またはコンテンツ部に空きがない状態を TSVC が満杯であると呼ぶ．

2.3　物理 TSVM を搭載した MOC の構成

TSVC を搭載した MOC は 4 台のスカラプロセッサとローカル変数に対するキャッシュ(LVC:Local Variable Cache)，及び命令キャッシュを搭載しており，その構成を図 5 に示す．各 TSVC はスヌープ機能を持ち，一貫性制御に用いる TSVC バスに接続されている．LVC は，TH や MT に共有されない変数が対象なので一貫性制御を行わない．DATA バスはデータ用の，ADR バスはアドレス用のバスであり，CNTL バスはその他の制御用バスである．また，全 PE に共有される L2 キャッシュ，集中型のバス調停回路，チップ外部とのインターフェースを持つ．

3.　MOC におけるマルチスレッド実行

3.1　TSVC の割当てと開放

TSVC も TSK の保護する資源と考え，TSK の生成時に共有変数を TSVC に割当てておき，プログラム実行中における主メモリアクセスの削減を図る．そして，TSVC の開放を TSK の消滅とともに行い，開放に関するオーバヘッドの隠蔽とエントリの有効利用を図る．各 TSVC の割当ては，1 台の PE が実行する放送読出し動作により行われる．放送読出し動作とは，各 TSVC の最優先度の空きエントリに対して同時に共有変数を読込ませる動作である．ただし，満杯状態の TSVC が存在するならば，LRU で選択したエントリを主メモリに書戻し，そのエントリに共有変数を読込む．読込みが完了したら，そのエントリの V が 1 にセットされる．TH の実行中には，ほぼ全ての共有変数が各 TSVC に存在し続けると考えられるため，false sharing の発生を抑える効果も期待できる．TSVC の開放は，ある PE のリセット動作により実行される．リセット動作とは，複数の TSVC に対し，TSK に保護された全エントリの V を 0 にする動作である．

TSVC の割当てと開放の例を図 6 に示す．時刻 T1 において，TSK0 の生成とともに，PE0 が放送読出しによって同一 TSK(TSK0) に属する PE0 と PE1 の

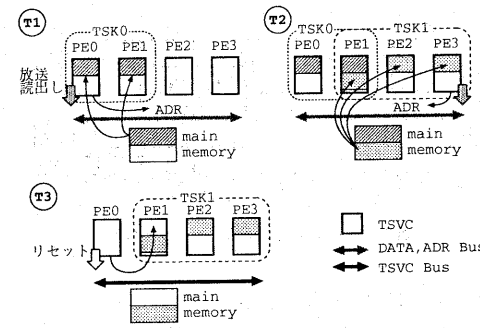

図6　TSVC の割当てと開放の例

TSVC に，T2 では，TSK1 の生成とともに，PE3 が TSK1 に属する PE1，PE2，PE3 の TSVC に共有変数を 1 エントリずつ読込んでいる．T3 では，TSK0 の消滅に伴い PE0 がリセット動作で TSK0 に属する PE0，PE1 の TSVC を一度に開放している．

3.2　TSVC の書込み及び読出し動作

3.2.1　TSVC と主メモリ間の置換え

共有変数の読書きに関して，各 PE は，通常，TSVC にのみアクセスするが，新たな TSK が生成された際に TSVC と主メモリ間で置換えが発生し，すでに TSVC に存在しない共有変数をアクセスする可能性がある．そのため，書込み及び読出し動作では，共有変数が TSVC に存在するかどうかの判定を行い，存在しないのであれば主メモリから TSVC に共有変数を読込む．

判定では，まず，TAG によって TSVC の全エントリを一致検索し，全エントリで不一致または一致したエントリの V が 0 ならば TSVC ミスヒット，一致したエントリの V が 1 ならば TSVC ヒットとなる．TSVC ミスヒットならば，TSVC に共有変数が存在しないことになり，主メモリから共有変数を読込む．その際，TSVC が満杯ならば，LRU により選択された TSVC の 1 エントリがメモリに書き戻され，そのエントリに共有変数が読み込まれる．TSVC が満杯ではないならば，空きエントリのうち最優先度の 1 エントリに共有変数が読み込れる．共有変数の読込みが完了したら，そのエントリの V は 1 にセットされる．

3.2.2　ブロッキングの判定

書込み及び読出し動作では，生産者と消費者間での同期をとるために，エントリへのアクセスをブロックするかどうかの判定が行われる．ただし，エントリの PT が 1 かつ PM が 0(恒久変数で通常モード) ならばその判定は行われない．判定において，CNT が非ゼロのエントリへの書込み及び CNT がゼロのエントリに対する読出しはブロックされ，プロセッサコアのパ

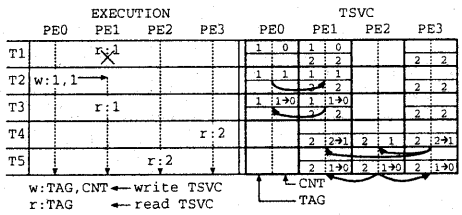

図7　TSVCの書込み及び読出し動作例

イプラインはストールされる．

3.2.3　書込み及び読出し動作の実行

書込み及び読出し動作がブロックされなかったならば，書き込み動作において，TSVCはエントリに各データを書込むと同時に，TSVCバスに1エントリのLRUFを除いた全フィールドを出力する．そして，他のTSVCはTSVCバス上のデータを満杯時ではLRUで書き戻したエントリに，そうでないならば最優先度の空きエントリに書込み，自PEが同一TAGでの読出しブロック状態ならばブロックを解除する．

読出し動作において，TSVCはエントリのDATAを自プロセッサコアに出力してCNTを1減じ，LRUFを除いた全フィールドをTSVCバスに出力する．そして，CNTが非ゼロかつWFが1ならば読出しをブロックし，CNTがゼロかつ一時変数ならばVを0にする．他のTSVCはTSVCバス上のTAGを用いて，TSVCヒット/ミスヒットを判定する．そして，TSVCヒット時には，そのエントリのCNTを1減じ，TSVCミスヒット時にはLRUで書き戻したエントリもしくは最優先度の空きエントリにTSVCバス上のデータを書込む．その結果，エントリのCNTが0であり，自PEが同一TAGでの書込み又は読出しブロック状態ならばブロックを解除する．また，そのエントリが一時変数ならばVを0にする．

3.2.4　書込み及び読出し動作の実行例

TSVCに対する書込み及び読出し動作の例を図**7**に示す．例では，すでに，PE1がTAG=2の共有変数を消費者PE2，PE3に対して生産しており，PE2に関しては，そのエントリが主メモリに書戻されTSVCに存在しないとする．また，TAG=1の共有変数は一時変数，TAG=2は恒久変数とする．

時刻T1において，PE1がTAG=1でTSVCを読出しているが，データが生産されていない(CNTが0である)ためブロックされている．T2において，PE0がTAG=1によってTSVCに書込み，PE1のTSVCにも放送された各データが書込まれている．このため，PE1のブロックが解除され，P1はT3においてTAG=1によるTSVCの読出しを実行し，PE0のTSVCも更新している．その結果，TAG=1の共有変数に対する同期通信が完了し，共有変数が一時変数であるため，T4以降においてそのエントリは消滅している．T4において，PE3がTAG=2によるTSVCの読出しを実行し，PE1はTSVCヒットしたエントリのCNTを1減じ，PE2はLRUで書戻されたエントリか最優先度の空きエントリに放送された各データを書込んでいる．T5では，PE2のTSVCの読出し結果が放送され，全エントリのCNTが0となりTAG=2の共有変数に対する同期通信が完了している．

```
STSVM DATA,SIZE,ADR,PID,TAG,CNT,WF//書込み動作
LTSVM DATA,SIZE,ADR,PID,TAG        //読出し動作
BTSVM DATA,SIZE,ADR,PID            //放送読出し動作
RTSVM MASK,ADR,PID,TAG             //リセット動作
CTSVM MASK,ADR,PID,TAG,PM          //モード変更動作
```

図8　TSVCに関する命令

3.3　共有変数のモード変更

モード変更動作は，あるPEがTSKに属する全TSVCに対してTAGで指定される恒久変数のモードを変更する動作である．モード変更を実行したPEはTSVCヒットしたエントリのPMを指定した値に変更し，TSVCバスを通じて，他のTSVCも同様に更新する．これにより，並列処理の実行中に以降で更新されない恒久変数を通常モードに変更し，各MTが同期をとらずに参照のみ行えるようにする．

3.4　物理TSVMに関する命令

物理TSVMへのアクセスはプロセッサコアが専用の命令を実行することで行われ，それらの命令とTSVCに対して行われる動作を図**8**に示す．TSVCに対して，STSVMは書込み，LTSVMは読出し，BTSVMは放送読出し，RTSVMはリセット，CTSVMはモード変更を行う命令であり，全ての命令において，DATA及びMASKはレジスタで指定しその他は即値で指定する．MASKはTAGと同一のビット長を持ったビットパターンで，MASKのビットを0にすることでTSVCはTAGの同一ビットを無視した検索を行う．PIDは各ビットがPEに対応した4ビット長のビット列であり，ビットを0にすることでPEを指定する．TAGとPIDはコード中でシンボルとして与えられ，タスクの生成時に設定される．

4.　評価及び考察

4.1　実験環境

TSVMのプログラムとの親和性及びTSVCの基礎的な機能であるPE間の同期通信の検証のために，TSVMを構造化メモリとして実装した図**9**のマルチプ

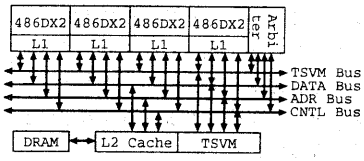

図9　MTA/TSVMの構成

表1　MTA/TSVMの基礎データ

説明	CLK
TSVM読出しのバスサイクル時間	4
TSVM書込みのバスサイクル時間	4
TSVM読出し失敗のバスサイクル時間	3
TSVM書込み失敗のバスサイクル時間	4
メモリ書込みのバスサイクル時間	4
メモリ読込みのバスサイクル時間	4

ロセッサMTA/TSVMを開発した．CPUは486DX2で，CPU内にL1キャッシュ，マザーボード上にL2キャッシュをもつ．TSVMをTSVCとしてではなく全CPUが共有する形で実現しているため，LVCも同様にL2キャッシュによって実現する．Arbiterは集中型のバス調停回路で，優先度をラウンドロビンで制御する．TSVMはXilinx社のFPGA(XC4020)により実装し，その仕様を限定して，TAGは8ビット，CNTは3ビット，DATAは32ビット，PT，PM，WFは各1ビット，エントリ数は16で，ハードウェア量は約7000ゲート規模である．TSVMをメモリ空間にマッピングし，L1キャッシュがキャッシングしない設定にしており，TSVMのアクセスをTAG，CNT，P-T，PM，WFをアドレスに埋め込んだmov命令により行う．また，TSVMのブロッキングはバスバックオフ機能により実現している[12]．MTA/TSVMの基本データは表1に示す通りであり，表中の数字はバス調停の2クロックを含む．測定はプログラムを1度実行し，コードがL1キャッシュに存在する状態で行った．

4.2　並列化プログラムに対する親和性

4.2.1　TSVCによる並列化

TSVMの並列化プログラムとの親和性を検証するために，**図10**(a)のプログラム(exam)を文レベルで並列化する．並列化手法の概念はTCSM[11]と同様であり，exam全体をTHとし，TH内部の文をリストスケジューリング[13]により各PEに割当てる．各PEが実行する文の集合をMTとし，MT間の真依存による先行制約を生産文の定義先とその消費文の参照先をTSVMに割当てることで満たす．ここでは，並列化前に全変数が共有される可能性があると考え，図10(a)内の全変数をTSVMに割当てた．そして，図10(b)のように最終的な結果であるx,y,zを恒久変数に，計算過程で一時的に使用される変数

```
float x,y,z;                    shared perm float x,0,0;
float dx,dy,dz;
int k;                          shared temp float dx,0,0;
S1: x=1;                        shared temp int k,0,0;
S2: y=1;                        S1: x,4=1;
S3: z=1;
S4: k=0;                        S4: k,1=1;
    do{                             do{
S5: dx=A*(y-x);                 S5: dx,1=A*(y-x);
S6: dy=x*(B-z)-y;               S6: dy,1=x*(B-z)-y;
S7: dz=x*y-C*z;                 S7: dz,1=x*y-C*z;
S8: x+=D*dx;
S9: y+=D*dy;                    S10: z,1=z+D*dz;
S10: z+=D*dz;                   S11: k,1=k+1;
S11: k=k+1;                     S12: }while(k<100);
S12: }while(k<100);             S13: cmode(mask,tag,0);
  (a) source code               (b) code after assigning TSVM
```

図10　ソースコードとTSVMへの割当て

```
        MT0                             MT1
 S1: x,4=1;                      S2: y,4=1;
 S3: z,3=1;                      S4: k,1=0;
     do{                             do{
 S7:  dz,1=x*y-C*z;              S6:  dy,1=x*(B-z)-dy;
S10:  z,1=z+D*dz;                S5:  dx,1=A*(y-x);
 S9:  y,1=y+D*dy;                S8:  x,1=x+D*dx;
S12: }while(k<100);              S11: k,2=k+1;
S15: tmp=dm;                     S12: }while(k<100);
S13: cmode(mask,tag,0);          S14: dm,1=0;
   (a) parallelizing program using TSVM

        MT0                             MT1
 S1: x=1;                        S2: Ry=1;
 S3: Rz=1;                           y=Ry;
     z=Rz;                       ↻ WT(y,Ry,1);
   ↻ WT(z,Rz,1);                 S4: k=0;
     Ry=RT(y);                       Rz=RT(z);
     do{                             do{
 S7:  dz=x*Ry-C*z;               S6:  Rx=x;

S12:  Rk=RT(k);                  S12: }while(k<100);
    }while(Rk<100);
   (b) parallelizing program using TCSM
```

図11　例題プログラムの並列化

k,dx,dy,dzを一時変数とし，最終結果の算出にともないS13(cmode(MASK,TAG,PM))で恒久変数のモードを通常モードに変更している．ただし，図10の表記は疑似コードで，各代入文の左辺におけるカンマの左側が変数名，右側がCNTで値は定義に対する参照数である．図10(b)に対して並列化した結果を**図11**(a)に示す．ただし，全MTが同一の制御パスを通るように条件文S12は全PEに割当てており，S11のCNTを割当て後に変更している．また，S13はMT1の実行完了後に行われる必要があるため，MT1によるダミーの一時変数(dm)への書込み(S14)及びMT0によるdmの読出し(S15)で条件同期[6]を行い先行制約を満たす．

4.2.2　TCSMによる並列化との比較検討

図10(a)をTSCMを用いて並列化した結果は図11(b)に示すようになり，図中のR*はレジスタ変数，WT，RTはTCSMへの書込みと読出しを表す．TCSMは同期通信時にのみ用られるため，データはメモリ上に格納されている．したがって，生産者は生産データをメモリへ書込むとともに更に消費者との同

表2　並列化の結果 (p:並列度)

プログラム	p=2		p=3		p=4	
	TS	TC	TS	TC	TS	TC
exam	53	54	58	73	62	74
gcd	63	66	73	77	81	83

表3　実験結果 (速度向上)

プログラム		p=2	p=3	p=4
exam	TS	1.87	2.26	2.09
	TC	1.44	1.20	1.20
gcd	TS	1.30	1.13	1.04
	TC	0.99	0.89	0.82

期通信のためにTCSMにも書込む (図11(b)の矢印).

各プログラムに対する並列化結果を**表2**に示す. gcdは最大公約数を求めるプログラム[14]であり, 表2の数字は, TSVM(TS)及びTCSM(TC)で並列化した各MTが実行するアセンブラの行数の総和である. TSVMでは恒久変数のモード変更命令が挿入されるが, TCSMよりも行数を抑えており, PE間通信のために挿入されるTCSMアクセス命令の方が多い. 以上から, TSVMは無駄な通信コードを削減でき, TCSMより並列化プログラムとの親和性に優れる.

4.3 MTA/TSVM上での評価及び考察

実験においては逐次プログラムの実行時間と, TSVMを用いて並列化したプログラムの実行時間, 及びTCSMでの並列化手法に沿って並列化したプログラムの実行時間を測定した. 逐次実行ではMTA/TSVM上の1台のPEでプログラムを実行し, TCSMの並列化ではTCSMの代わりにTSVMを使用している. 表2の各プログラムに対するMTA/TSVM上での実験結果を**表3**に示す. 表3のTS及びTCは逐次の実行時間をTSVM及びTCSMの実行時間で割った値である. TSVMがすべてにおいて逐次及びTCSMよりも良い結果となり, 通信コードの削減が性能向上にも寄与した. 台数効果が小さい原因として各PEが共有するTSVMへのアクセス競合が考えられ, TSVMのキャッシュ化による命令実行とバス転送のオーバラップで更なる速度向上が期待できる.

5. 結　び

共有変数の取り扱いに注目し, ローカル変数キャッシュと同期付き共有変数キャッシュ(TSVC)を持ったMOCを提案し, マルチスレッド実行環境におけるTSVCの概念を示した. また, TSVMのプログラムとの親和性を検証するためにTCSMでの並列化と比較検討した結果, コード量を最大26%少なくでき, TSVMはTCSMより親和性に優れることがわかった. また, 実機での評価によりTSVMは逐次に対して最大2.26, TCSMに対して最大1.88の速度向上を得たことから, 更なる性能向上を達成できることがわかった. 今後は, より多くのプログラムによるTSVMの評価やより効率的な並列化手法の検討を行う. そして, TSVMのキャッシュ化による高速性の検証と詳細な機構の検討も行っていく.

参考文献

1) Hammond, L. et al.：THE STANFORD HYDRA CMP, IEEE MICRO Magazine, Vol. 20, No. 2, pp. 71–84 (2000).
2) 鳥居淳ほか:オンチップ制御並列プロセッサMUSCATの提案, 情処学論, Vol. 40, No. 5, pp. 1622–1631 (1998).
3) Tsai, J.Y. et al.：The Superthreaded Processor Architecture, IEEE Trans on Comp, Vol.48, No. 9, pp. 881–901 (1999).
4) 小林良太郎ほか：非数値計算向けスレッド・レベル並列処理マルチプロセッサ・アーキテクチャSKY, 情処学論, Vol.42, pp. 349–366 (2001).
5) Venkata, K. and Torrellas, J.：A Chip-Multiprocessor Architechture with Speculative Multithreading, IEEE Trans on Comp,Vol. 48, No. 9, pp. 866–880 (1999).
6) 岩根雅彦, 山脇彰, 田中誠：マルチプロセッサオンチップにおける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 (2000).
7) Nikhil, A.S. et al.：I-Structure:Data Structures for Parallel Computing, ACM Trans on Prog Lang Sys, Vol. 11, No. 4, pp. 598–632 (1989).
8) Nuth, P.R. and Dally, W.J.：The named state register file: Implementation and performance, IEEE Proc. 1st Int. Symp. on HPC, pp. 4–13 (1995).
9) 岩下茂信, 宮嶋浩志, 村上和彰：次々世代汎用マイクロプロセッサアーキテクチャPPRAMの概要, 情処学研報, ARC-113-1 (1995).
10) 松岡孝, 原口恵美子, 岩根雅彦：マルチプロセッサMTA/TSMの開発, 情処学第60回全国大会予稿集, pp. 105–106 (2000).
11) 山脇彰, 田中誠, 岩根雅彦：同期通信用メモリに対する並列化手法と評価, 情処研報, Vol. 2001, No. 22, pp. 37–42 (2001).
12) 山脇彰, 岩根雅彦:同期通信用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 (2001).
13) Rewini, H.E. et al.：Task Scheduling in PARALLEL and DISTRIBUTED SYSTEMS, Prentice Hall (1994).
14) Wolfe, M.：HICH PERFORMACE COMPILERS FOR PARALLEL COMPUTING, Addison-Wesley (1996).