計算機アーキテクチャ 142－7
ハイパフォーマンス
コンピューティング 85－7
（2001．3．8）

同期通信用メモリに対する並列化手法と評価

山脇 彰† 田中 誠† 岩根雅彦†

半導体の集積度向上により，様々なマイクロプロセッサアーキテクチャが実現可能となっている．その選択肢の一つに複数のプロセッサコアを1チップに搭載したシングルチップマルチプロセッサ(SCMP)がある．SCMPはプログラムのスレッドレベルから命令レベルの並列性を抽出し性能向上を遂げるマルチスレッド実行環境を対象としている．そのような環境におけるスレッド間の先行制約を満たす機構として，タスクに属するスレッド間の高速な1対多の条件同期，相互排除およびバリア同期を実現できる同期通信用メモリTCSMIIがある．TCSMIIは無効なエントリに対する読み出しブロックを行うため，データの真依存に関してTCSMIIによる通信で先行制約を満たすことができる．TCSMIIはレジスタを介してアクセスされることから並列化の際にレジスタ変数を導入し，生産文と消費文を分割し，TCSMII命令を挿入する．並列化したプログラムの実行時間をSCMPと等価なマルチチップ・マルチプロセッサMTA/TCSMIIで実測した．結果より，逐次プログラムの実行時間に対して平均1.16～2.23の速度向上を得たことからTCSMIIの有効性が確認できた．

A Parallelizing Method and Evaluation of The Communication and Synchronization Memory

AKIRA YAMAWAKI,† MAKOTO TANAKA † and MASAHIKO IWANE†

The progress of semiconductor technology has made processors mounted on a single chip. the Single Chip-MultiProcessor(SCMP) extracts multiple threads and instruction parallelism across them from the application. In multithreading environment, it is an important problem how the interthread dependence is satisfied. TCSMII is a communication and synchronization memory that achieves a high-speed condition synchronization, mutual exclusion, and barrier synchronization between threads. TCSMII performs blocking to the consumer thread when it read a invalid data that is not written by the producer thread yet. Therefore, TCSMII satisfies the ture dependence between a consumer thread and the producer thread. Since TCSMII is accessed using a register, the register variable is used for inserting the instraction of TCSMII. we evaluate the parallelizing programs using TCSMII on MTA/TCSMII. The result shows that TCSMII achieves 1.16～2.23 speed-up on an average to the serial programs.

1 はじめに

半導体の集積度向上により，様々なマイクロプロセッサアーキテクチャが実現可能となっている．その選択肢の一つとして，複数のプロセッサコアを1チップに搭載したシングルチップマルチプロセッサSCMP(Single Chip Multi-Processor)がある．SCMPは，複数のスレッドから命令レベル並列性を抽出し性能向上を図るマルチスレッド実行環境を対象にしている．そのような環境においては，協調動作するスレッド間の先行制約を満たす機構が必要である．

それに対し，Superthread[2]は，スレッド間で真依存のある変数を先行スレッドから後続スレッドへの単方向のメモリバッファを経由して通信し，同期命令で実行順序を保つ．MUSCAT[3]はスレッド間の世代定義により，データ依存を先行スレッドから後続スレッドへの単方向に制限し，後続スレッドのデータアクセス抑止命令により先行制約を満たす．

SCM[4]では，スレッド(タスク)間の先行制約を同期フラグを用いて保証している．これらの機構は基本的に同期と通信を分離させてる．SKY[5]では，レジスタのデータ依存に対する先行制約は，受信側のリザベーションステーションにおけるWait，WENフラグの状態により満たし，その同期と通信は命令レベルで行われる．

一方，マルチスレッド環境下において，文レベル以上のスレッド間での高速な1対多の条件同期，相互排除およびバリア同期を実現できる同期通信用メモリTCSMII[1)6)]がある．TCSMIIは同期と通信を同時に取り扱い，真依存による先行制約をデータの通信と同時に満たすことができる．ここでは，スレッド間の真依存に関する先行制約をTCSMIIの条件同期により満たして並列化を行う方法を示す．そして，TCSMIIを用いて並列化したプログラムの実行時間と，通常のメモリにより並列化したプログラムおよび逐次における実行時間との比較を，TCSMIIを搭載したSCMPと等価なマルチチップマルチプロセッサMTA/TCSMII上で行い，TCSMIIの有効性を示す．

2 同期通信用メモリTCSMII

2.1 TCSMIIの概要

TCSMIIはマルチスレッド環境下において条件同期，相互排除，バリア同期を統一的に扱える同期通信用メモリであり，

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

その概念図を図 1に示す．タグフィールドはエントリを識別

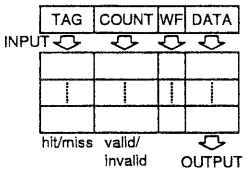

図 1 TCSMII の概念図

するためのタグを格納し，検索キーとして使用される．カウントフィールドは通信回数を格納し，その非ゼロでデータの有効を，ゼロで無効を表す．WF(Wait Flag) は TCSMII を用いた高速バリア同期[6] を実現するための 1 ビットのフラグであり，データフィールドは通信データを格納する．

TCSMII の基本動作は書き込み，読み出し，リセット動作である．このうち，書き込み動作とリセット動作は TCSM[1] と同じである．読み出し動作は成功時と失敗時に分けることができる．読み出し成功とは，タグを入力しヒットしたエントリのカウントが非ゼロの場合であり，読み出し失敗とは，タグを入力し全エントリでミスしたか，ヒットしたエントリのカウントがゼロの場合である．読み出し成功時に，TCSMII はデータを出力してカウントをデクリメントするが，WF が 0 か 1 かで動作が異なる．WF が 0 で，カウントのデクリメント後にそれが 0 となった場合，同一タグによる書き込みブロックを解除する．WF が 1 で，カウントのデクリメント後にそれが非ゼロとなった場合，読み出しを行ったスレッドのバスをブロックし，スレッドは待ち状態となる．カウントがゼロになった場合は，同一タグによる書き込みブロックと待ち状態を解除する．

TCSMII に関する命令を図 2に示す．STCSM2 命令は書き込み動作，LTCSM2 は読み出し動作，RTCSM2 はリセット動作を行う．RTCSM2 命令のマスクデータは，タグの任意のビットをマスクするためのビットパターンを指定する．ビットを 1 にすることにより，そのビットを無視した検索を行うことができる．

2.2 TCSMII による条件同期

TCSMII はマルチスレッド環境下において，タスクに属する複数のスレッドおよびマイクロスレッドにより使用される．ここでのタスクはプログラムの実行環境であり，スレッドはタスクに内在する粗粒度の並列性に対応し，マイクロスレッドはスレッド内の文レベル以上の並列性に対応する[1]．以後，スレッドとマイクロスレッドを総称してスレッドとよぶ．タグをタスク ID と変数名の連結とすることで，TCSMII をタスク毎に自動的に保護できる．

TCSMII に対する無効なエントリの読み出しブロックで 1 対 1 および 1 対多での条件同期を実現し，有効なエントリに対する書き込みブロックで相互排除を実現する[1]．また，バリア変数をタグとして，スレッドに対する WF を用いた TCSMII 読出し後のバスブロックと，TCSMII の同一タグによる上書きブロックを利用し，任意参加型の高速バリア同期を実現する[6]．ここでは，TCSMII による条件同期を使用するので，それについて説明する．

1 対多の条件同期を用いて，スレッド間で 1 対 n 通信を行

```
STCSM2   Data, Tag, Count, WF
LTCSM2   Data, Tag
RTCSM2   MaskData, Tag
format
```

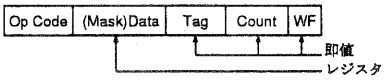

図 2 TCSMII の命令

生産者スレッド

```
wtcsm2( tag, n, R1 );
```

消費者スレッド(n-1)

```
Reg = rtcsm2( tag );
        ⋮
Reg = rtcsm2( tag );
```

(a) **1対多通信 (条件同期)**

```
wtcsm2( tag, nt, reg){
  STCSM2 reg, tag, nt, 0;
}
```

```
rtcsm2( tag ){
  register int reg;
  LTCSM2 reg, tag;
  return reg;
}
```

(b) **マクロ**

図 3 TCSMII を用いた 1 対多通信

うプログラム例を図 3(a) に示す．wtcsm2 は書き込み動作を実行するマクロであり，第 1 引数にタグ，第 2 引数に通信回数，第 3 引数にデータ (レジスタ) を指定する．rtcsm2 は読み出し動作を実行するマクロであり，第 1 引数にタグを指定し，データをレジスタに返す．生産者スレッドがデータをあるタグを用いて TCSMII に書き込む前に，消費者スレッドが同一のタグでもって TCSMII からデータを読み出した場合，その消費者はブロックされる．生産者スレッドが通信回数を n としてデータを生成したあと，消費者スレッドはブロックを解除されデータを読み出し，同時にエントリのカウントは 1 だけ引かれる．n 個の消費者スレッドが読み出しを完了した時点で，カウントは 0 となりタグで指定されるエントリは解放される．

3 TCSMII のための並列化手法

3.1 概要

並列化にあたって，対象となるソースコードを基本ブロックとループブロックに分類する．ループブロックは Doall, Doacross 可能な最外周ループである．基本ブロック内部はステートメントレベルで並列化し，ループブロックはイタレーション単位で並列化を行う．そして，それぞれをスレッドとする．ここでは並列して動作するスレッド間の先行制約を，TCSMII を用いた条件同期により満たす．

スレッドに割当てられた，ステートメント間の先行制約には真依存，逆依存，出力依存があり，それぞれの依存関係にあるステートメントはコードの実行順を保つ必要がある．真依存関係に関しては，同期と通信を同時に行うことができる TCSMII の特性を最大限利用できる．また，逆依存と出力依存に対しては真依存と同様の方法で先行制約を満たすことができるため，真依存に関して焦点を当てる．真依存関係にあるステートメントに対して，データの定義側を定義文，データの参照側を参照文とよぶ．また，定義文が割り当てられているスレッドを定義スレッド，参照文が割り当てられているスレッドを参照スレッドとよぶ．

並列化手順の主な流れは，(1) ソースコードを基本ブロックとループブロックに分割して，(2) 各ブロックの制御フローグラフを作成し，データフロー解析を行う．そして，(3) 基本ブ

ロックに対しては内部のステートメントをリストスケジューリング [7] によりスレッドに静的に割当て，(5) ループブロックはイタレーションをインデックス毎にスレッドに静的に割当てる．(6) すべてのブロックに関して，スレッド間の先行制約を TCSMII を用いて満たす．手順において，入力はソースコードと並列度であり，出力は並列化された単一のコードである．

並列化されたコードは並列度と同一台数のプロセッサに割当てられ，スレッドとプロセッサが 1 対 1 に対応付けされる．全スレッドは制御フローグラフのトップに存在する基本ブロックから実行を開始する．そして，各ブロック出口の分岐文の実行結果にしたがって後続のブロックに制御を移し，並列化されたコードを実行していく．

並列化の対象となるソースコードを関数やループボディなどに適用し，それらの前後で fork/join によりスレッドの生成/消滅を行うことで fork–join モデルに適用可能である．また，並列化されたコードを動的にプロセッサに割り当てることも可能である．

3.2 データフロー解析

TCSMII 命令の挿入にあたって，プログラム中の各ステートメント間に存在する真依存関係を調べる必要がある．そのために，まず，並列化の対象となるソースコードを基本ブロックとループブロックに分割し，それらに対して制御フローグラフを作成する．作成した制御フローグラフを用いて以下のデータフロー方程式を解き基本ブロックおよびループブロックの入口に到達する定義 [8] を求める．ループブロックに対しては内部の制御構造を考慮しない．

$$d\_in[B,x] = \cup_{B' \in pred(B)} d\_out[B',x]$$
$$d\_out[B,x] = d\_gen[B,x] \cup (d\_in[B,x] - d\_kill[B,x])$$

B はブロック，$pred(B)$ はブロック B の先行ブロックの集合である．$d\_gen[B,x]$ はブロック B 内で変数 x に対して，最後に x を定義したステートメントの集合であり，$d\_kill[B,x]$ はブロック B 内に定義がある変数 x について，ブロック B 以外のブロックにおける定義の集合である．$d\_in[B,x]$ はブロック B の入口に到達する可能性のある変数 x を定義するステートメントの集合であり，$d\_out[B,x]$ はブロック B の出口に到達する可能性のある変数 x を定義するステートメントの集合である．変数 x は，各ブロックで参照される変数であり，各ブロック B の $d\_in[B,x]$ は空集合で初期化しておく．

次に，各ブロック内のあるステートメント S に対して，到達する定義を以下の手順で求める．変数 x に関して，ブロック B 内のステートメント $S_i (i = 1, \cdots n)$ に到達する定義を $reach(S_i, x)$ とする．

(1) $reach(S_k, x) := d\_in[B,x]$ とする．

(2) $i = 1, \cdots, k-1$ なる各 S_i に対して，S_i で殺される定義を $reach(S_k, x)$ から除き，S_i での定義を $reach(S_k, x)$ に加える．

逆依存に関しては，前述のデータフロー解析を制御フローグラフのボトムからトップに向かって逆向きに適用することにより検出可能である．出力依存は $reach(S_i, x)$ の変数 x をブロック内で定義される変数にすることで検出できる．

例題プログラムを図 4(a)，その制御フローグラフを同図 (b) に示し，それぞれの参照文 $S_i (i = 1, \cdots, 7)$ に対する

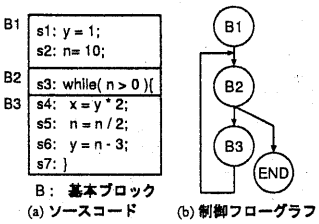

図 4　例題プログラム

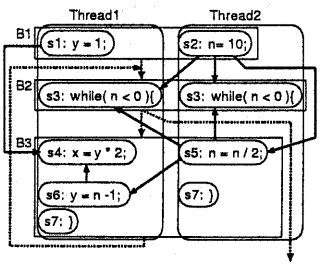

図 5　ブロック毎の並列化結果

$reach(S_i, x)$ を表 1に示す．図 4のプログラムは S_6 から S_4 にループ伝搬依存があり，イタレーション毎に並列化しても速度向上が見込めない例である．

表 1　例題プログラムにおける到達する定義

ブロック	文	変数	$d\_in[B_i, x]$	$reach(S_i, x)$
B_1	S_1	y	ϕ	ϕ
B_1	S_2	n	ϕ	ϕ
B_2	S_3	n	S_2, S_5	S_2, S_5
B_3	S_4	y	S_1, S_6	S_1, S_6
B_3	S_5	n	S_2, S_5	S_2, S_5
B_3	S_6	n	S_2, S_5	S_5

3.3 ステートメントのスケジューリング

基本ブロック内部のステートメントはリストスケジューリングによりスレッドに割り当てられる．リストスケジューリングは，ステートメントをノードとしてノード間の先行制約を示したタスクグラフ [7] を用いる．リストスケジューリングを各ブロック毎に実行させるため，各ブロック内の依存情報のみを用いてタスクグラフを作成する．タスクグラフの各ノードの優先順位にしたがって，ノードをスレッドに割り当てていくが，ブロック出口の条件分岐文のみは全スレッドに対して割り当てる．これにより，全スレッドは現行の基本ブロックの実行完了後に，後続ブロックに制御を移しプログラムの実行を続けていく．

図 4の例題プログラムに対する各ブロック毎の並列化結果を図 5に示す．ただし，並列度は 2 である．図中の実線による有向辺はデータフローを，点線による有向辺は制御フローを表している．

3.4 基本ブロックに対する TCSMII 命令の挿入

前節までに，基本ブロック内の各ステートメントのスレッドへの割当てが完了する．そこで，異なるスレッドに割当てられた定義文と参照文に対し，TCSMII 命令を挿入する．ただし，同一スレッドに割り当てられた定義文と参照文に関する真依存は，コードの実行順により自動的に解消されるため考慮しない．また，ある定義文とそれに対する参照文の集合

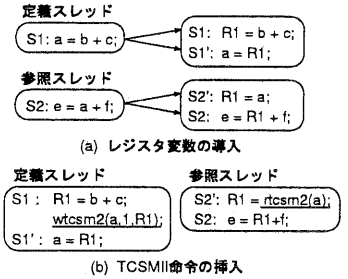

図 6 式の分割と TCSMII 命令の挿入

もしくは部分集合がそれぞれ異なるスレッドに割り当てられている場合，定義文と制御フローにおいて最も先行する参照文に対して TCSMII 命令を挿入することにより，後続する参照文の先行制約は満たされる．したがって，そのような真依存も考慮しない．例えば，図 5の場合 S_5 と S_3，S_6 間の真依存であり，S_5 と S_6 間の先行制約が満たされると，S_5 と S_3 間の先行制約も同時に満たされる．したがって，S_3 に関する真依存は考慮しない．

TCSMII はレジスタを介してアクセスされるため，レジスタ変数を導入し図 6(a) に示すように定義文と参照文を分割する．そして，同図 (b) のように，定義スレッドに対して，定義文の直後に定義を参照スレッドに送信する wtcsm2 命令を挿入し，参照スレッドに対して，参照文の直前に定義スレッドからの定義を受信する rtcsm2 命令を挿入する．上記の処理が参照文に対してすでに適用されている場合は，参照文に対してなにも行わない．タグはタスク ID と定義される変数名の連結であるが，図 6ではタスク ID を簡単のため省略している．カウントは参照スレッドの数である．参照スレッドは受信したデータをレジスタもしくはローカルなメモリ領域に保持しておく．

上記の処理が完了した時点で，逆依存と出力依存に対して TCSMII 命令を挿入する．命令の挿入は，レジスタ変数を導入しないこと，挿入場所が異なることを除いて真依存と同様である．命令の挿入位置であるが，逆依存に関しては，参照文の直後に wtcsm2 命令を挿入し，定義文の直前に rtcsm2 命令を挿入する．タグはこの先行制約を示す一意のものとし，カウントは定義スレッド数である．出力依存に関しては，コード順で先行の定義文の直後に wtcsm2 命令を挿入し後続の定義文の直前に rtcsm2 命令を挿入する．タグは先と同様であり，カウントは後続の定義文を割当てられたスレッド数とする．双方とも，データの通信は行わない．

3.5 基本ブロック間での命令移動とカウントの再設定

基本的に前節の方法で TCSMII 命令を挿入していくが，基本ブロックを跨いで定義文と参照文間に挿入した TCSMII 命令に対して以下の処理を施す．

一番目に，図 7(a) に示すような先行ブロック (B1) の定義文に対して，1 つ以上の後続ブロック (B2,B3) で異なるスレッドに割当てられた参照文が存在する場合を考える．このとき，前節と同様の方法で TCSMII 命令を挿入した場合，定義スレッドはカウントを参照スレッド数として TCSMII に書き込むが，制御パスによって，一部の後続ブロックでしか読み出しが実行されない，または，どの読み出しも実行され

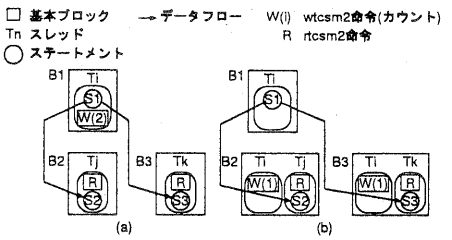

図 7 ブロック間で定義文に対して 1 つ以上の参照文が存在する場合

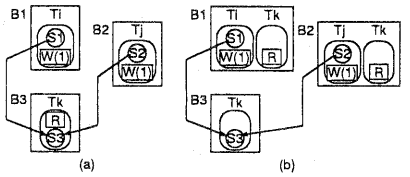

図 8 ブロック間で参照文に対して 2 つ以上の定義文が存在する場合

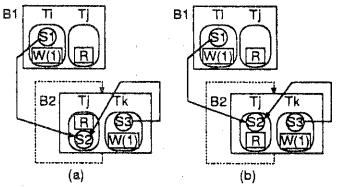

図 9 先行ブロックに対して後続ブロックが複数回実行される場合

ない可能性がある．その場合，カウントが非ゼロのまま残り，TCSMII のエントリは解放されない．そのために，以降の同一タグによる TCSMII 書き込みが不正にブロックされてしまう．したがって，このような場合は同図 (b) のように，後続ブロックにおいて定義スレッドが実行するコードの先頭に wtcsm2 命令を挿入する．また，カウントは後続ブロックにおける参照スレッド数にする．

2 番目に，図 8(a) に示すような，あるブロック (B3) の参照文に対して異なる定義スレッドに割り当てられた定義文が 2 つ以上の先行ブロック (B1,B2) に存在する場合を考える．このような場合は，先行ブロックの書き込みが不正にブロックされる可能性がある．例えば，制御フローが B1 → B3 と B1 → B2 → B3 であったとすると，全スレッドの実行が B1 → B2 と移った際に，B2 での書き込みは不正にブロックされてしまう．したがって，同図 (b) のように，先行ブロックで参照スレッドが実行するコードの最後 (条件文の直前) に後続ブロックの rtcsm2 命令を含むステートメントを移動する．

3 番目に図 9(a) に示すように，ある変数 x に対して，先行ブロックのステートメント (S_1) と後続ブロックのステートメント (S_2) 間に真依存があり，後続ブロックで S_3 から S_2 へのループ伝搬依存によりスレッド間 (T_j–T_k) で x を通信する必要がある場合を考える．

これは，2 番目の問題に対して後続ブロックと同一の先行ブロックが存在する場合と等価である．このとき，T_j は，B1 から B2 に制御を移した後の S_2 の実行時に，T_i の定義した値ではなく，T_k が定義した値を保持していることになる．し

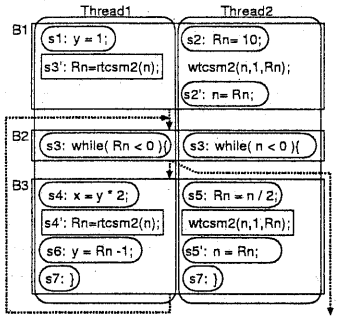

図 10 TCSMII 命令挿入後のプログラム

```
for ( I = 0; I < N; I++ ){
  S1 : x[ I ] = a[ I ] + b[ I ];
  S2 : y[ I ] = x[ I - 1 ] + c[ I ];
}
```

(a) ループプログラム

```
for ( I = TID; I < N; I+=NT ){
  S1: R1 = a[ I ] + b[ I ];
  S1': x[ I ] = R1 ;
  S2': R2 = x[ I - 1 ];
  S2: y[ I ] = R2 + c[ I ];
}
```

(b) 式の分割

```
for ( I = TID; I < N; I+=NT ){
  S1: R1 = a[ I ] + b[ I ];
  wtcsm2( I, 1, R1 );
  S1': x[ I ] = R1 ;
  S2': R2 = rtcsm2( I - 1 );
  S2: y[ I ] = R2 + c[ I ];
}
```

(c) TCSMII命令挿入後

TID : Thread ID
NT : Number of Threads

図 11 Doacross 型ループに対する TCSMII 命令の挿入

たがって，このような場合は同図 (b) のように，後続ブロックの rtcsm2 命令をブロックの最後 (分岐文の直前) に移動する．図 4の例題プログラムに対して，並列化後に TCSMII 命令を挿入した結果を図 10に示す．

3.6 ループブロックに対する TCSMII 命令

ループブロックは Doall，Doacross の形で並列化を行う．Doall の場合は，ブロック内部に関してスレッドの先行制約を考慮する必要はない．Doacross の場合，ループ伝搬依存によるスレッド間の先行制約を TCSMII による条件同期で保証する．そのため，ループ依存関係にあるステートメント間に真依存関係があるとみなし，基本ブロックと同様の方法で TCSMII 命令を挿入する．タグは配列名とインデックスの連結とし，カウントはループ伝搬依存するスレッド数である．Doacross における TCSMII 命令の挿入例を図 11に示す．

ループブロックに対して先行ブロックや後続ブロックに真依存関係がある場合，その先行制約を満たす必要がある．ループブロックはインデックス毎にイタレーション単位でスレッドに割り当てられており，どのスレッドがどの配列要素やスカラ変数を参照するのか，または，定義するのかが判明している．また，スレッドがループブロックに到達した時点で，各スレッドは最新の値を保持している．したがって，ループブロックにおいて，先行ブロックの定義スレッドとループブロックの参照スレッド間での通信を TCSMII を用いてループ開始前に終了させる．後続ブロックの参照スレッドとループブロックの定義スレッド間での通信は後続ブロック側で行う．

4 MTA/TCSMII 上での評価

4.1 実験環境

実験で使用するマルチプロセッサ MTA/TCSMII は，MTA/TCSM[1] に対して任意参加型バリア同期機構への拡張とともに，TCSM アクセスを IO 命令から mov 命令に変更し，キャッシュ無効化信号をメモリに書き込んだプロセッサには送らないようにした改良機である．MTA/TCSMII は，8 台の 486DX2 からなる単一バス結合共有メモリ型マルチプロセッサであり，MPU 内に L1，マザーボード上に L2 キャッシュを持つ．L1 キャッシュはライトスルー，L2 キャッシュはライトバックである．TCSMII およびバスアービタは単一バスに接続されている．バスアービタは集中型で，バスの優先度は回転式である．

測定において，L1 と L2 キャッシュともにオンとし，プログラムのコードおよびデータが L1 キャッシュに存在することを確認してから測定した．MTA/TCSMII の基本データを表 2に示す．実験では，スレッド ID とプロセッサ ID を 1 対 1 に対応させている．

4.2 評価プログラム

TCSMII に対して，逐次実行の他に，通常のメモリを用いた場合 (メモリ) との比較も行う．メモリでは，ステートメント間の先行制約を同期フラグを用いて満たす．定義スレッドは定義文の直後でフラグを 1 にセットし，参照スレッドは参照文の直前でそのフラグに対してビジーウェイトを行う．

評価プログラムは 3 つであり，1 番目は，整数 a,b の最大公約数 g および $ax+by=g$ なる整数 x,y を求めるプログラム (gcd)[9] である．gcd は初期値を設定する部分と，最大公約数を求めるメインのループ，整数 x,y を求める部分から構成される．メインのループは，反復回数が不明であるためイタレーション毎に並列化できない．したがって，内部をステートメントレベルで並列化していく．ここで整数 a,b はプログラム実行開始時に与えられているとする．

2 番目は，以下の式を変形して 4 元の 1 次連立微分方程式とし，ルンゲクッタ法で解くプログラム (runge)[10] である．

$$m_1\frac{d^2x_1}{dt^2} = -k_1x_1+k_{12}(x_2-x_1)$$
$$m_2\frac{d^2x_1}{dt^2} = -k_{12}(x_2-x_1)+k_2(-x_2)+u(t)$$

初期値は $m_1=m_2=1, k_1=K_{12}=k_2=1, u(t)=0, x_1=-1, x_2=1$ である．runge は初期値を代入する部分と，4 次のルンゲクッタ法を計算するメインのループから構成される．メインのループはループボディの最後尾のステートメントから最前列のステートメントに対してループ伝搬依存があり，イタレーションレベルでの並列化を行っても速度向上が得られない．そのため，内部をステートメントレベルで並列化する．また，評価で用いるプログラムにおいて実数の精度は単精度である．

3 番目は 1 次連立方程式 ($Ax=b$) を解くヤコビの反復法 (jacobi)[11] である．jacobi は，x の初期値を設定する部分と，ヤコビの反復法を行うメインのループから構成される．初期

表 2 MTA/TCSMII の基礎データ

略称	説明	CLK
W_{tcsm2}	wtcsm2 のバスサイクル時間	4
R_{tcsm2}	rtcsm2 のバスサイクル時間	4
RB_{tcsm2}	TCSMII 読出し失敗のバスサイクル時間	3
WB_{tcsm2}	TCSMII 書込み失敗のバスサイクル時間	4
W_{mem}	メモリ書込みのバスサイクル時間	4
R_{mem}	メモリ読込みのバスサイクル時間	4(3)
BTS	Lock 付き Test&Set 命令のバスサイクル時間	9
AND	Lock 付き AND 命令のバスサイクル時間	8

():バースト転送時の追加分

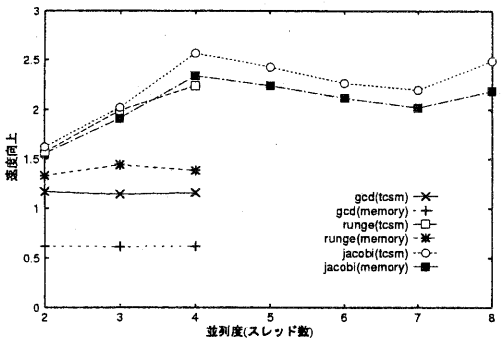

図 12 実測結果 (速度向上)

値の設定部分は Doall 型のループ (LB1) である．メインループは，繰り返し回数が収束判定によって決まるためイタレーションで並列化できない．メインループの内部は，新しい解を求めるループ (LB2) と x を更新するループ (LB3) の 2 つの Doall ループで構成されている．LB2 は x に関して，LB1 および LB3 と真依存関係にあり，LB1 と LB3 の最後で x を通信する．

4.3 実験および考察

2 つのプログラムに対する実測結果を図 12に示す．グラフの横軸はスレッド数，縦軸は逐次処理に対する TCSMII およびメモリを用いた並列処理の速度向上である．

最大公約数を求めるプログラムは $a = 92736, b = 57314$ とした結果である．並列度は最大で 4 となった．TCSMII は平均で 1.16，最大で並列度 2 における 1.17 の速度向上を得た．一方，MEMORY は平均で 0.62 と速度向上が得られていない．これは，TCSMII はデータへのアクセスと同時に同期をとれるが，メモリでは，同期フラグとデータへのアクセスを必要とするためと考える．ステートメントの平均実行時間は 6.3 クロックであり，このことから，TCSMII を用いた並列処理は，粒度の小さい場合においても有効であることがわかる．

4 次のルンゲクッタ法のプログラムの並列度は最大で 4 となった．TCSMII において，平均の速度向上が 1.91，最大で並列度 4 の 2.22 となり，最大公約数と違い良好な結果が得られた．これは，ステートメントが主に浮動小数点演算からなるため，実行時間の平均が 55 クロックとなり，比較的粒度が大きく通信のオーバヘッドを隠蔽したためと考える．メモリにおいても，平均で 1.38 の速度向上を得ているが，TCSMII と比較して悪い結果となった．このことから，TCSMII を用いた並列処理はステートメントの粒度が大きい場合でも有効であることが示せた．

ヤコビの反復法では 8 元の連立一次方程式を用いた．TCSMII における速度向上は平均で 2.23，最大で並列度 4 の 2.57 である．一方，メモリにおける速度向上は平均で 2.05，最大で並列度 4 の 2.34 となり，TCSMII の方が良い結果を得た．並列度 5 から 7 にかけて速度向上が横ばいであるが，これはループブロックの繰り返し回数が 8 であり，イタレーションを 2 回実行するスレッドが実行時間を占めることによる．全体的に，並列度が増加するにつれ速度向上の伸びが小さくなっている．これは，スレッド数の増加にともない通信回数も増加するためである．また，MTA/TCSMII が共有バス結合型のマルチプロセッサであり，L1 キャッシュがライトスルーであることも原因と考えられるが，ここでの議論の範疇ではないと考える．このことから，ループブロックを含んだ並列処理においても TCSMII は有効であると考える．

5 結び

スレッドを基本ブロック内部のステートメント，ループブロックのイタレーションとし，真依存による生産者スレッドと消費者スレッド間の先行制約を TCSMII を用いて満たす並列化手法を示した．並列化において，TCSMII はレジスタを介してアクセスされることから，生産文と消費文をレジスタ変数を用いて分割し，そこに TCSMII アクセス命令を挿入した．本手法にしたがって並列化した 3 つのプログラムに対して実測を行った結果，逐次プログラムに対して平均 1.16〜2.23，通常のメモリに対して平均 1.09〜1.87 の速度向上を得たことから，TCSMII の有効性が確認できた．

今回，例題プログラムと評価プログラムに本手法を適用して問題は生じなかったが，今後は，並列化手法に対する適用性のより詳細な検証を行っていくとともにそのアルゴリズムを確立する．また，並列化されたコードのプロセッサへの割当て方法の検討を行う．

参 考 文 献

1) 岩根，山脇，田中: 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，J83-D-I, No.3, pp.317-328(2000).
2) J-Y.Tsai, J.Huang, C.Amlo, D.J.Lilja, P-C.Yew: The Superthreaded Processor Architecture, IEEE Trans on Comp, Vol.48, No.9, pp.881-901(1999).
3) 鳥居，近藤，木村，池野，小長谷，西: オンチップ制御並列プロセッサ MUSCAT の提案，情処学論，Vol.39, No.6, pp.1622-1631,(1998).
4) 木村，尾形，岡本，笠原: シングルチップマルチプロセッサ上での近細粒度並列処理，情処学論，Vol.40, No.5, pp.1924-1934,(1999).
5) 小林，岩田，安藤，島田: 非数値計算プログラムのスレッド間命令レベル並列を利用するプロセッサ・アーキテクチャSKY，並列処理シンポジウム JSPP'98, pp.87-94,(1998).
6) 山脇，田中，岩根: 同期通信用メモリを用いた高速バリア同期機構，情処学第 62 回全国大会予稿集，4P-5,(2001 年 3 月予定).
7) H.El-Rewini, T.Lewis, H.Ali: Task Scheduling in PARALLEL and DISTRIBUTED SYSTEMS, Prentice Hall,1994.
8) 佐々:プログラミング言語処理系，岩波書店 (1994).
9) M.Wolfe: High Performance Compilers for Parallel Computing, Addison-Wesley Publishing Company(1996).
10) 新濃，船田: 数値解析の基礎 理論と PAD・PASCAL・C，培風館 (1991).
11) B.Carnahan,H.A.Luther,J.O.Wilkes(著) 藤田，森，名取，三井，中村，花田，沼田 (共訳): 計算機による数値計算法，日本コンピュータ協会 (1982).