情報処理学会論文誌 Vol. 49 No. 4 1657–1668 (Apr. 2008)

FPGA ベースオンチップマルチプロセッサにおける同期付きキャッシュメモリの実装と評価

山脇 彰[†1] 岩根雅彦[†1]

FPGA ベースのオンチップマルチプロセッサ（FOMP）は，開発コストを削減しながら要求性能を達成しうるフルプログラマブルなオンチップマルチプロセッサである．本論文では，そのような FOMP において共有変数を介した低オーバヘッドな同期通信を実現するために，既存のスヌープキャッシュを TSVM（Tagged Shared Variable Memory）キャッシュに拡張し，それによって生じるハードウェアオーバヘッドと消費電力への影響を明らかにする．実験から，拡張にともなった回路規模の増加は 5%，動作速度の低下は 2%であり，大きな影響なく TSVM キャッシュ化できることが確認された．さらに，TSVM キャッシュは電力性能比を 1.22 倍〜2.56 倍に改善できることも確認された．

An Implementation and Evaluation of Snoop Cache with Synchronization on an FPGA Based Multiprocessor

Akira Yamawaki[†1] and Masahiko Iwane[†1]

FPGA based on-chip multiprocessor (FOMP) is an on-chip multiprocessor with fully programmable feature which can reduce development cost and achieve performance requirement. In order to provide an FOMP with the low-overhead communication and synchronization methods via shared variables, this paper attempts to introduce the TSVM (Tagged Shared Variable Memory) cache to a snooping cache on the FOMP. The TSVM cache can improve a performance by combining communication and synchronization with the coherence maintenance. Using an FPGA, we have evaluated how extending a conventional snooping cache affects circuitries and clock speed. As a result, the growth of hardware amount and the degradation of clock speed are only 5% and 2% respectively. It is also confirmed that the TSVM cache improves performance and energy efficiency.

1. はじめに

大規模化が進むシステムオンチップに対しては開発期間やコストの削減が重要な課題であり，その取り組みは，IP の再利用[1] やハードウェア/ソフトウェアコデザイン[2),3)] 等の設計技術の向上から，仕様や設計変更に柔軟かつ迅速に対応できるようハードウェアを軟化させた再構成デバイス[4),5)] の利用まで多岐にわたる．その中で，オンチップマルチプロセッサは，ソフトウェアの柔軟性を最大限活用しながら要求性能の達成を図ったプロセッサ・アーキテクチャである．特に，FPGA を用いたオンチップマルチプロセッサ（FOMP）は，ハードウェアも軟化させたより柔軟性の高いアーキテクチャとして注目されている[6)–8)]．

FOMP では，並列処理をサポートする任意の機構を導入できるため，文献 7) と文献 8) の FOMP は，それぞれ，IPv4 パケットの処理と LU 分解に特化した内部構成をとり，高性能化を達成している．また，文献 6) の FOMP は，プログラマビリティやポータビリティの観点から汎用的な共有メモリ型の SMP（Symmetry Multiprocessor）構成を採用している．そして，共有メモリの SDRAM とプロセッサコア間でのデータ転送を効率的に行える DMA を含んだラッパ（HIBI）を導入して高い性能とスケーラビリティを達成している．

我々は，文献 6) と同様の立場で，多くの共有メモリ型 SMP が採用するスヌープキャッシュに注目し，その機能を拡張することによって低オーバヘッドな同期通信の実現を図る研究を行ってきた[9),10)]．具体的には TSVM（Tagged Shared Variable Memory）と呼ぶ同期機能を付加した構造化メモリの機能を既存のスヌープキャッシュに導入する．TSVM は共有変数に対してロード/ストア命令による共有変数への読み書きと同時に同期を実現する．以降，TSVM の機能を付加したスヌープキャッシュを TSVM キャッシュと呼び，その特徴は以下のとおりである．

(1) 共有変数を用いて統一的に生産者–消費者間同期（条件同期），相互排除，バリア同期を 1〜2 命令で実現できる．
(2) 同期を共有変数の一貫性制御（通信）と同時に実現し，同期通信の低オーバヘッド化を図る．
(3) 既存のスヌープキャッシュに寄生して，ハードウェアオーバヘッドを隠蔽する．

従来の同期構造化メモリの I-structure[11)–13)] は，条件同期を TSVM と同様に 1 命令で

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

実現できるが，相互排除とバリア同期は TSVM よりも複雑になる．また，我々は，文献 14) で，I-structure よりも変数の再利用が容易な TSVM の優位性を示し，文献 10) では，サイクル精度のソフトウェアシミュレータを用いて従来のスヌープキャッシュに対する性能面での優位性を示した．しかしながら，FOMP に対して TSVM キャッシュを導入する際に生じるハードウェアオーバヘッドや消費電力への影響が明らかになっていない．そこで，本論文では，FPGA への実装を通じてそれらを明らかにし，FOMP における TSVM キャッシュの有効性を示す．

以降，2 章で，前提とするプログラムの実行環境とオンチップマルチプロセッサの構成を示し，3 章で，FPGA に実装したプロトタイプ FOMP（FOMP/TSVM）の詳細を示す．4 章では，例を用いて，TSVM キャッシュの動作を補足的に説明する．そして，5 章で FPGA への実装結果を示し，6 章で性能と消費電力を評価する．最後に 7 章でむすぶ．

2. 前　　提

2.1 プログラムの実行環境

想定するプログラムの実行環境は，**図 1** に示すように，一般的なマルチスレッドプログラミングモデルをベースに粒度の小さな並列化も考慮したモデルである．このモデル上で，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保護された資源を共有しながら協調動作する．さらに，スレッドは，より粒度の小さな複数のコード列に静的に並列化されうる．スレッドが並列化された個々のコード列をマイクロスレッドと呼び，ループ単位の中粒度並列性から文列単位の近細粒度並列性に相当する．スレッド間，および，マイクロスレッド間で同期通信に使用される変数を同期共有変数，その他の変数を一般変数と呼ぶ．アドレス空間は仮想的に TSVM と通常メモリに分離され，同期共有変数は前者に，一般変数は後者に割り当てられる．

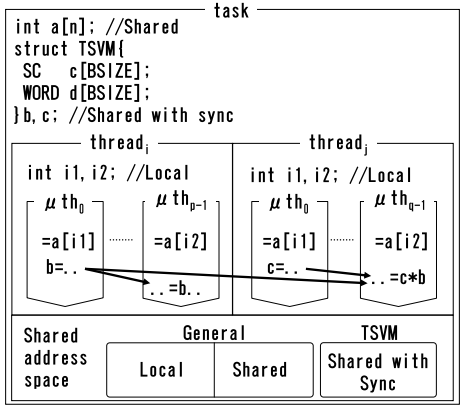


図 1　プログラムの実行環境
Fig. 1　Programming model.

同期共有変数は単なる構造体であり，スレッドやマイクロスレッドは他の変数と同様に読み書きできる．この構造体は，通信データを格納するためのフィールド（WORD d[BSIZE≥1]）を持つ．通信データはワード単体でもよいし，複数ワードからなるブロックでもよい．さらに，各要素が各ワードに対応したフィールド（SC c[BSIZE]）がある．このフィールドの要素を同期カウンタと呼び，各々には通信データを介した同期の回数が書き込まれる（読み出し回数ではない）．

同期カウンタが 0 の同期共有変数は，値が未定義であり，その読み出しは暗黙的に待たされる．同期カウンタが 1 以上の場合，値が定義済みであり，読み出しは待たされずに実行される．読み出しと同時に同期カウンタがデクリメントされ，やがて，同期カウンタが 0（未定義）になり，そのワードは再利用可能となる．定義済みの同期共有変数に対する書き込みも，同期共有変数が未定義になるまで，待たされる．未定義の同期共有変数に対しては，値とともに，同期カウンタも同時に書き込まれる．以降では，同期共有変数が未定義（同期カウンタが 0）であることを empty，定義済み（同期カウンタが 1 以上）であることを full と呼ぶ．

さらに，同期共有変数に対する読み出しと書き込みの動作は，**表 1** のように，細分化される．厳密読み出しは，通常読み出しと同様に，empty な同期共有変数に対して full になる

表 1　同期共有変数に対する読み書き動作
Table 1　Read and write actions for shared variable with sync.

	ブロック条件	値の更新	
		同期カウンタ	データ
通常読み出し	empty	−1	なし
厳密読み出し	前：empty 後：full	−1	なし
無同期読み出し	なし	なし	なし
非遷移読み出し	empty	なし	なし
通常書き込み	full	入力	入力
無同期書き込み	なし	入力	入力

まで待たされ，full な同期共有変数の読み出し後に，同期カウンタを −1 する．ただし，厳密読み出しは，読み出し後に当該同期共有変数が empty になるまで（再利用可能になるまで）待たされる．無同期は読み書きとも，full/empty とは無関係に，待たされないことを意味し，読み出し後に同期カウンタは −1 されない．非遷移読み出しも，同様に，同期カウンタの値を −1 しないが，読み出し時に empty ならば full になるまで待たされる．同期カウンタを 1 とし，非遷移読み出しを利用した場合，同期共有変数は I-structure と等価である．

2.2　対象とするオンチップマルチプロセッサ

図 1 のプログラム実行環境がマッピングされるオンチップマルチプロセッサの概要を**図 2** に示す．オンチップマルチプロセッサは 1 つ以上のクラスタ（$\mathrm{CL}_0, \cdots, \mathrm{CL}_{m-1}$）からなり，各クラスタは 1 つ以上のプロセッサからなる．クラスタ間は任意のチップ内結合網によって接続される．複数のプロセッサを持つクラスタは，共有メモリマルチスレッドプログラミングモデルに適した SMP 構成をとる．各プロセッサは命令キャッシュ（IC）とデータキャッシュ（DC）を持ち，各々が，クラスタ内結合網によって接続される．データキャッシュは TSVM の機能が付加されたスヌープキャッシュ（TSVM キャッシュ）であり，一貫性制御の専用バス（一貫性制御バス）を持つ．

1 つのクラスタに 1 つのタスクが，クラスタ内のプロセッサ群にタスク内のスレッドが割り当てられる．1 つのスレッドは，クラスタ内のプロセッサ数と同数のマイクロスレッドに並列化でき，各マイクロスレッドが異なる 1 台のプロセッサに割り当てられる．コンテキストの切替えはスレッド以上の単位で行われ，粒度の小さいマイクロスレッド単位では行われない．

TSVM キャッシュの概要を**図 3** に示す．TSVM キャッシュはメモリ上の一般変数と同期共有変数（図 1 の構造体）をキャッシングする．それらの変数は，発行されたロード/ストア命令が従来のものか，図 3 に示す同期共有変数へのロード/ストア命令によるものかによって区別される．図 3 の命令は，各々が，表 1 の各操作に対応する．書き込み時に同期カウンタの値はアドレス部で指定され，ライン上に設けられた同期カウンタのフィールド（図 3 の $\mathrm{SC}_0 \cdots \mathrm{SC}_{n-1}$）に書き込まれる．同期カウンタはワード数分あり，各々が各ワード（図 3 の $\mathrm{WORD}_0 \cdots \mathrm{WORD}_{n-1}$）に対応する．

一貫性制御バスは集中型のバス調停回路（CBA）によって調停される．専用バスを介して TSVM キャッシュの一貫性制御を行うことにより，その他の転送との衝突が緩和される．それゆえ，TSVM キャッシュの一貫性制御に関しては，キャッシュ間でデータの通信を高速に行える更新型プロトコル（MOESI 等）を採用する．また，並列処理の正しさを保障するために，一貫性制御に関しては，逐次コンシステンシ[15] が保たれるとする．これは，共有

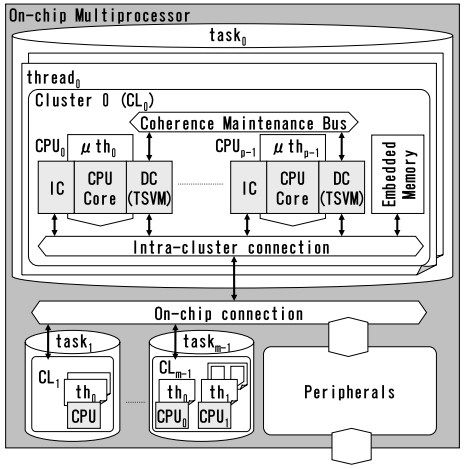
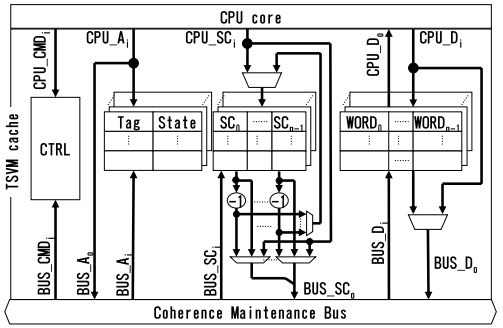


図 2　オンチップマルチプロセッサの概要
Fig. 2　Overview of on-chip multiprocessor.



Shared variable with sync. access instructions

```
LWS     R, Addr     : Load Word with Sync.
STLWS   R, Addr     : Strict LWS
NBLWS   R, Addr     : Non-Blocking LWS
NTLWS   R, Addr     : Non-Transtion LWS
SWS     R, SC_Addr  : Store Word with Sync.
NBSWS   R, SC_Addr  : Non-Blocking SWS
```

図 3　TSVM キャッシュの概要
Fig. 3　Overviwe of TSVM cache.

変数へのアクセスにメモリバリアが挿入されたことと等価である．

クラスタ内のプロセッサ間で低オーバヘッドな同期通信が必要ならば，同期共有変数を TSVM キャッシュにキャッシングし，同期共有変数に対する一貫性制御を発生させる．すると，TSVM キャッシュ間で同期共有変数のデータと同期カウンタが同時に更新され，プロセッサ間の同期通信が同時に行われる．同期共有変数の内容を更新する通常読み出し，厳密読み出し，通常書き込み，および，無同期書き込みは一貫性制御を発生し，その他の操作は一貫性制御を発生しない．

3. プロトタイプ FOMP

3.1 構　　成

開発したプロトタイプ FOMP（FOMP/TSVM）の構成を**図 4** に示す．これは，図 2 における 1 つのクラスタに相当する．設計には VHDL を用い，使用する FPGA は Xilinx 社の Virtex4 を想定した．

プロセッサコアは MIPS R3000 をベースとした 32 ビットのスカラプロセッサであり，パイプラインは 5 段構成で，1 つの遅延スロットを持つ[16)–18)]．命令キャッシュ（IC）とデータキャッシュ（TSVM キャッシュ）は 16 KB の 2 ウェイセットアソシアティブキャッシュであり，1 ラインは 4 ワード構成（1 ワードは 32 ビット）である．TSVM キャッシュはライトバックキャッシュであり，置換するウェイは LRU によって選択される．一貫性制御に関して，一般共有変数は MOESI プロトコル[19)]，同期共有変数は専用プロトコル（3.4.1 項で説明）によって一貫性が保たれる．なお，仮想記憶，および，ノンブロッキングキャッシュは未実装である．

クラスタ内結合網はデータ幅が 64 ビットの共有バス（ICB: Intra-Cluster Bus）とし，一貫性制御バス（CMB）のデータ幅は 128 ビットとした．CMB のデータ幅を 128 ビットとした理由は，本設計においてキャッシュの 1 ラインを 4 ワード構成としており（総ビット数は $4 \times 32 = 128$ ビット），更新ラインの転送を 1 回のバス転送で完了させるためである．集中型のバス調停器（それぞれ，CMBA，ICBA）が各共有バスをラウンドロビン方式で調停する．また，両バスはスプリットバス転送をサポートしていない．

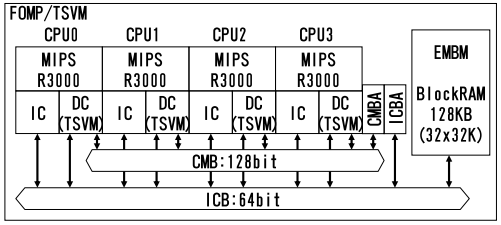


図 4　FOMP/TSVM の構成
Fig. 4　Organization of FOMP/TSVM.

組み込みメモリ（EMBM）は，128 KB のブロック RAM（幅 32 ビット，深さ 32K）であり，評価で使用されるプログラムのコードとデータが格納される．

3.2 同期共有変数へのアクセス命令

図 3 の同期共有変数に関する命令を実現するにあたり，3 番コプロセッサ（COP3）のロード/ストア命令[17)] を**表 2** のように拡張した．これにより，同期共有変数へのアクセス命令を，インラインアセンブラのマクロとして C プログラムから使用できる．同期カウンタの値はベースレジスタ（表 2 中の Rb）中の上位 4 ビットによって指定される．TSVM キャッシュに入力されるアドレスの上位 4 ビットは COP3 のレジスタで設定される．つまり，あらかじめ COP3 レジスタにメモリマップにおける同期共有変数領域を設定する．同期共有変数の構造体は以下のとおりである．

```
struct svs{
  unsigned int SC; //for word alignment
  unsigned int WORD[4]; }
```

これは，TSVM キャッシュの 1 ライン分に相当する．

3.3 TSVM キャッシュの構成

TSVM キャッシュの構成を**図 5** に示す．Virtex4 のブロック RAM は，最小単位で，幅が 36 ビット，深さが 512 の構成を持てる．その単位を基準に，同期カウンタとワード，および，タグと 4 ビットの状態フラグ（3 ビットは MOESI 用，1 ビットは LRU 用）のペア

表 2　同期共有変数アクセス命令と MIPS 命令との対応
Table 2　MIPS instructions for TSVM.

同期共有変数命令		MIPS 命令
SWS	R, SC_Addr	SDC3 R, 0x0001(Rb)
NBSWS	R, SC_Addr	SDC3 R, 0x0002(Rb)
LWS	R, Addr	LDC3 R, 0x0003(Rb)
STLWS	R, Addr	LDC3 R, 0x0004(Rb)
NBLWS	R, Addr	LDC3 R, 0x0005(Rb)
NTLWS	R, Addr	LDC3 R, 0x0006(Rb)

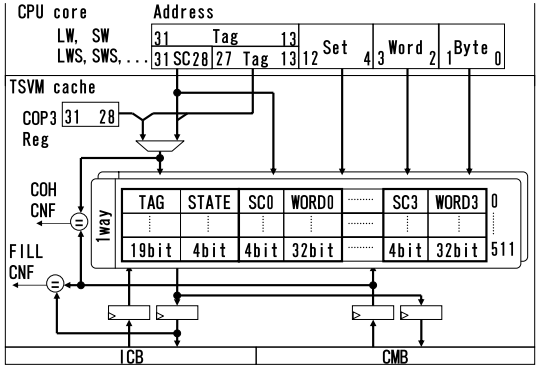


図 5　FOMP/TSVM における TSVM キャッシュの構成
Fig. 5　Organization of TSVM cache on FOMP/TSVM.

でブロック RAM に割り当てた．それゆえ，1 ウェイで 5 個のブロック RAM が使われる．アドレスに関して，セットアドレス，ワードアドレス，および，バイトアドレスで，それぞれ，9 ビット，2 ビット，2 ビットが必要となる．アドレスの総ビット幅は 32 なので，残りの 19 ビットがタグとなる．ブロック RAM は 9 ビット単位の書き込みをサポートしており，タグと状態フラグを個別に更新できる．さらに，デュアルポートなので，同一セット以外は，プロセッサコアとバスが同時にアクセスできる．

プロセッサのアドレス入力には，同期共有変数に対するアドレス（COP3 レジスタとアドレスの 27〜13 ビットの連結）と一般変数に対するアドレスを切り替えるマルチプレクサが挿入される．一貫性制御バスと TSVM キャッシュとの境にあるレジスタは共有バスをわたる遅延とプロセッサ内部の遅延を分離する．

図中の COHCNF は，プロセッサが一貫性制御の実施を試みているラインに対し，他 TSVM キャッシュによる一貫性制御が先に実施されたことを示す．また，FILLCNF はメモリからフィルしているラインに対して，他 TSVM キャッシュによる一貫性制御が発生したことを示す．これらが存在する理由は次節で説明する．

なお，4 つの同期カウンタに対するデクリメンタは，各同期カウンタが 4 ビットなので，4 つの LUT（1 つの LUT が 4 入力 1 出力）によって実現できる．

3.4　キャッシュコントローラ

既存スヌープキャッシュのキャッシュコントローラを，一般変数と同期共有変数が扱えるように拡張する．スヌープキャッシュは，通常，プロセッサからのアクセスを制御するコントローラとバスの情報を用いて一貫性制御を実施するスヌープコントローラを持つ．ここでは，前者を CPU Access Controller（CAC），後者を SNooPing Controller（SNPC）と呼び，各々に対する拡張を**図 6**，**図 7** に示す．図中の 2 重に囲まれた状態は，その状態がさらに複数の状態によって階層的に構成されることを意味する．なお，図中では省略したが，MOESI の状態は一般変数と同期共有変数とで同様に更新されるので，キャッシュ間のライン転送も，オーナとの間で正しく行われる．

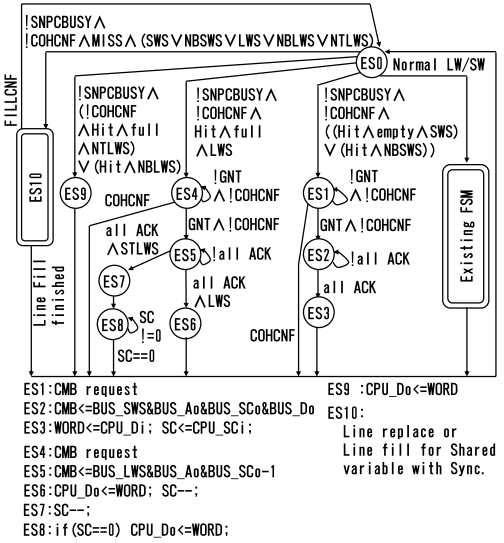
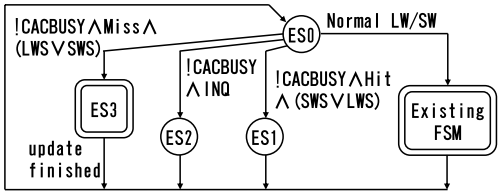


図 6　CPU アクセスコントローラ
Fig. 6　CPU Access Controller.

3.4.1　CPU アクセスコントローラ（CAC）

プロセッサが同期共有変数へアクセスした際に，当該同期共有変数が TSVM キャッシュ上に存在し，かつ，同期待ちが発生しなかったら，CAC は発行された命令に従って，ES0 から ES1，ES4，または，ES9 に遷移する．ただし，SNPC がメモリとの間でラインの転送中（SNPCBUSY が 1）ならば，CAC はその完了まで待つ．ES1 と ES4 は，それぞれ，書き込みと読み出しで一貫性制御を要求する状態である．一方，ES9 は一貫性制御を必要としない読み出し操作なので，値をプロセッサに返すだけである．ES1，または，ES4 に




```
ES1: WORD&SC<=BUS_Di&BUS_SCi;
     ACK     <='1';
ES2: if( Owner ){
       CMB<=BUS_SCo&BUS_Do;
       ACK<='1'; }
     else NACK <='1';
ES3: if( Set is full ) memory<=LRU line;
     else TAG&WORD&SC<=BUS_Ai&BUS_Di&BUS_SCi;
     if  (update finished) ACK <= '1';
```

図 7　スヌープコントローラ
Fig. 7　Snooping Controller.

おいて，他 TSVM キャッシュが一貫性制御を同一ラインに対して実施したならば，図 5 の COHCNF が 1 になり，CAC は状態を ES0 に戻す．これは，当該同期共有変数の最新値は一貫性制御バス上の値であり，SNPC が当該ラインを更新した後に，再びプロセッサにアクセスさせるためである．ES1，または，ES4 において一貫性制御バスの許可（GNT）を取得できたら，CAC は，ES2，および，ES5 で一貫性制御バスにラインを出力する．そして，一貫性制御を実施して他の全 TSVM キャッシュが完了応答（all ACK）を返すまで待つ．完了応答を受けた CAC は，ES3，ES6，または，ES7 において命令に応じた操作を TSVM キャッシュ上の同期共有変数に対して実施する．ES7〜ES8 は厳密読み出しにおける読み出し後の処理を実施する．CAC は，ES7 で同期カウンタを −1 し，ES8 において他 TSVM キャッシュの一貫性制御によって同期カウンタが 0 になるまでプロセッサにデータを返さない．

同期共有変数に対する同期待ち（たとえば，empty な同期共有変数に対する LWS や full な同期共有変数に対する SWS）は，プロセッサにデータが返されないため，メモリステージでのストールによってなされる．これは，キャッシュミス時のストールと同じである．

同期共有変数が TSVM キャッシュになかったら，CAC は ES0 から ES10 に遷移し，メモリに対してラインの置換，または，読み出しを実行する．その際，同期共有変数の構造体がメモリと TSVM キャッシュ間で転送される．ES10 では，まず，セットに空きがなければ，CAC は LRU ラインをメモリに書き戻し，空きがあるか，上記の書き戻しが完了したら，CAC は一貫性制御バスを介して他 TSVM キャッシュにラインがあるか問い合わせる．存在すれば，当該 TSVM キャッシュから一貫性制御バスを介してラインが転送され，そうでなければメモリからラインが読み出される．

ES10 におけるラインの転送中に，他 TSVM キャッシュが同一ラインに対する一貫性制御を実施したならば，FILLCNF が 1 となり，CAC は現在の転送を中断して ES0 に戻る．これは，前述した一貫性制御の要求中におけるラインの衝突と同じ理由による．そして，SNPC が当該ラインを更新した後に，再度，CAC は TSVM キャッシュへアクセスする．

3.4.2　スヌープコントローラ（SNPC）

一貫性制御バスの読み書きに対して当該ラインがあれば，SNPC は ES0 から ES1 に遷移し，バス上のデータでそのラインを更新して一貫性制御の完了を通知する（ACK）．一貫性制御バスからオーナーラインの問合せ（INQ）があれば，SNPC は ES0 から ES2 に遷移し，当該ラインがあれば，そのラインを一貫性制御バスに出力してラインの送信を通知する（ACK）．

一貫性制御バスからの入力に対し，当該ラインがなかったら，SNPC は ES3 へ遷移して一貫性制御バス上のラインを空きウェイに書き込む（空きがなかった場合，LRU ラインが置き換えられる）．これは，キャッシュミスによる無駄なライン転送の削減が目的である[10]．そして，一貫性制御の完了を通知する（ACK）．

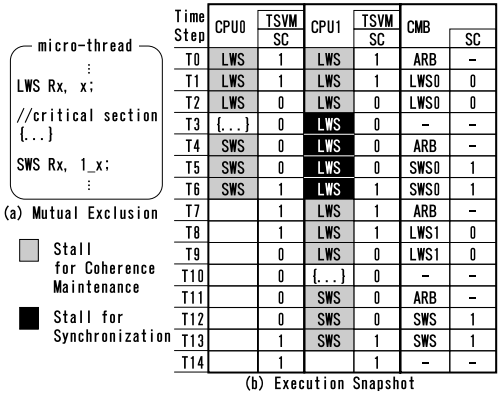
なお，ES1，ES2，または，ES3 への遷移時に，一貫性制御の対象ラインと異なるラインに対して CAC がメモリとの間でデータを転送中（CACBUSY が 1）ならば，SNPC はその完了まで待つ．

4.　TSVM キャッシュの動作詳細

本章では，FOMP/TSVM の動作に関する詳細を，相互排除とバリア同期を例に補足的に説明する．なお，条件同期については文献 10) を参照されたい．

4.1　相 互 排 除

同期共有変数を用いた場合，**図 8** (a) のように，危険領域の入り口で，同期カウンタが 1 の同期共有変数を LWS で読み出し，危険領域の出口で同期カウンタを 1 としてその同期共有変数に書き込めば，相互排除が実現される[20]．図 8 (b) に 2 台のプロセッサが相互排除を実行した際の動作を示す．薄い網掛けは一貫性制御の完了待ちによるストールを，濃い網掛けは同期待ちによるストールを意味する．



Time Step	CPU0	TSVM SC	CPU1	TSVM SC	CMB	SC
T0	LWS	1	LWS	1	ARB	–
T1	LWS	1	LWS	1	LWS0	0
T2	LWS	0	LWS	0	LWS0	0
T3	{...}	0	LWS	0	–	–
T4	SWS	0	LWS	0	ARB	–
T5	SWS	0	LWS	0	SWS0	1
T6	SWS	1	LWS	1	SWS0	1
T7		1	LWS	1	ARB	–
T8		1	LWS	1	LWS1	0
T9		0	LWS	0	LWS1	0
T10		0	{...}	0	–	–
T11		0	SWS	0	ARB	–
T12		0	SWS	0	SWS	1
T13		1	SWS	1	SWS	1
T14		1		1	–	–

(b) Execution Snapshot

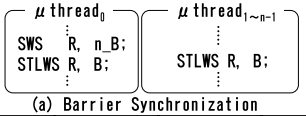
図 8　相互排除
Fig. 8　Mutual exclusion.

図 8 (b) の T0〜T2 で，CPU_0 と CPU_1 は full な同期共有変数の読み出しによる一貫性制御の完了待ちでストールされている．この間，CPU_0 が一貫性制御バス（CMB）を駆動している．そして，T3 において，一貫性制御を完了した CPU_0 は危険領域を実行している．一方，CPU_1 のストールは，同期カウンタが 0 になったため，一貫性制御の完了待ちから同期待ちに切り替わっている（COHCNF を利用した一貫性制御の中断による）．その後，T6 において，危険領域を抜ける際に CPU_0 が実行する SWS によって各 TSVM キャッシュの同期カウンタが 1 になっている．その結果，同期待ちのストールが解除された CPU_1 は，CPU_0 と同様に危険領域を通過する．

4.2　バリア同期

同期共有変数を用いた場合，1 対多の条件同期を応用することによって，**図 9** (a) のように，バリア同期が実現できる[20]．つまり，生産者スレッドがバリア変数（B）に対して，同期カウンタをバリア同期の参加者数にして書き込み，参加者全員でそのバリア変数を厳密読み出しによって読み出すだけである．

図 9 (b) に 3 台のプロセッサがバリア同期を実行した際の動作を，図 8 (b) と同様に示す．バリア同期の参加者（CPU_0〜CPU_2）のうち，生産者は CPU_0 とする．T0〜T2 にかけて，CPU_0 が，同期カウンタを 3 として，バリア変数に書き込んでいる．その間，バリア同期点に到達した CPU_1 は同期カウンタが 0 の同期共有変数に対する厳密読み出しでストールされている．T3〜T5 にかけて，CPU_0 と CPU_1 は，full な同期共有変数に対する厳密読み出しを実施し，一貫性制御の完了待ちでストールされている．T5 で厳密読み出しによる一貫性制御を完了させた CPU_1 は，自 TSVM キャッシュ上の同期カウンタが非 0 なので，T6〜T11 にかけて同期待ちによりストールされている．その後，T9〜T11 にかけて，CPU_0 も同様に同期待ちでストールされている．その間，バリア同期点に到達した CPU_2 が厳密読み出しによる一貫性制御を完了させている．その結果，T11 で同期カウンタが 0 になり，3 つのプロセッサが T12 でバリア同期を抜ける．



Time Step	CPU0	TSVM SC	CPU1	TSVM SC	CPU2	TSVM SC	CMB	SC
T0	SWS	0	STLWS	0		0	ARB	–
T1	SWS	0	STLWS	0		0	SWS0	3
T2	SWS	3	STLWS	3		3	SWS0	3
T3	STLWS	3	STLWS	3		3	ARB	–
T4	STLWS	3	STLWS	3		3	LWS1	2
T5	STLWS	2	STLWS	2		2	LWS1	2
T6	STLWS	2	STLWS	2		2	ARB	–
T7	STLWS	2	STLWS	2		2	LWS0	1
T8	STLWS	2	STLWS	2		2	LWS0	1
T9	STLWS	1	STLWS	1	STLWS	1	ARB	–
T10	STLWS	1	STLWS	1	STLWS	1	LWS2	0
T11	STLWS	0	STLWS	0	STLWS	0	LWS2	0
T12		0		0		0		–

(b) Execution Snapshot

図 9　バリア同期
Fig. 9　Barrier synchronization.

5. FPGA への実装

FOMP/TSVM を実装する FPGA は Virtex4 の xc4vlx100-10ff1148 とした．これは，FOMP/TSVM を搭載するのに十分な大きさを持つからである（スライスの使用量は全体の約 33%）．論理合成と配置配線には Xilinx 社の ISE 8.2.02i を使用し，配置のオプションをスピード重視に変更した．なお，以降では，比較対象である TSVM キャッシュ化前のスヌープキャッシュを搭載した FOMP を FOMP/SC と呼ぶ．

表 3 に実装結果を示す．表 3 の各モジュールは図 4 の各モジュールに対応する．1 行目のスライスとブロック RAM（BRAM）直下の数字は，使用 FPGA の最大数である．スライ

表 3　FPGA への実装結果
Table 3　Result of FPGA implementation.

モジュール		FOMP/TSVM スライス 49,125	BRAM 240	周波数 [MHz]	VHDL 記述量（行）	FOMP/SC スライス 49,125	BRAM 240	周波数 [MHz]	VHDL 記述量（行）
CPU_n	R3000	1,856	0	52.8	8,550	1,840	0	53.6	8,342
	DC	1,633	10	56.8	4,958	1,450	10	58.1	3,612
	IC	681	10	56.3	（TOP）138	681	10	56.3	（TOP）138
		4,042	20	51.9	（TOP）486	3,719	20	53.0	（TOP）486
FOMP	CMBA	18	0	347	185	18	0	347	185
	ICBA	18	0	347	185	18	0	347	185
	CMB	173	0	524	137	169	0	589	102
	ICB	102	0	303	58	102	0	303	58
	EMBM	165	64	189	339	165	64	189	339
		16,085	144	50.1	（TOP）899	15,270	144	51.2	（TOP）899

ス数に関しては，各モジュールごとに論理合成と配置配線を行った結果であり，各モジュールにわたったスライス数の総和は全体のスライス数と一致しない．VHDL 記述量に関して，TOP は最上位階層のみの記述量を意味する．CPU_n はプロセッサコア（R3000）とデータキャッシュ（DC），および，命令キャッシュ（IC）をコンポーネントで利用し，FOMP は 4 つの CPU_n とバス調停回路（CMBA，ICBA），一貫性制御バス（CMB），クラスタ内共有バス（ICB），および，組み込みメモリ（EMBM）をコンポーネントで利用している．また，命令キャッシュ（IC）は，FOMP/SC のデータキャッシュ（DC）をコンポーネントで利用し，プロセッサコアからの書き込みと ICB への書き込みポート，および，CMB とのインタフェースをトップ記述において無効化している（命令キャッシュは FOMP/TSVM と FOMP/SC で同じものである）．

TSVM キャッシュ化によって CPU_n のスライスが約 9%増加した．これには，キャッシュ自身の増加分に加えて，表 2 の命令を実現するために施したプロセッサコアの拡張分も含まれている．動作周波数では約 2%の低下が見られた．これは，クリティカルパスがタグの比較部分であり，タグとなるアドレスを同期共有変数と一般変数で切り替えるマルチプレクサがクリティカルパス上に挿入されたためである（今回の設計において，同期カウンタをアドレスの一部に埋め込み，実際のアドレスは COP3 のレジスタに設定するようにしたことによる）．ブロック RAM の使用数に変化は見られなかった．これは，3.3 節で述べたように，同期カウンタとワードを同一ブロック RAM に収めたことによる（その他のタグと状態フラグは FOMP/TSVM と FOMP/SC で共通である）．また，TSVM キャッシュ化するために VHDL 記述量が，従来スヌープキャッシュでは 1,346 行，R3000 では 208 行増加した．

バス調停回路（CMBA，ICBA），クラスタ内共有バス（ICB），および，組み込みメモリ（EMBM）は FOMP/TSVM と FOMP/SC で共通なため，スライス数と VHDL 記述量は同じである（なお，CMBA と ICBA は同一である）．CMB では FOMP/TSVM においてスライス数と記述量が増加している．これは，一貫性制御時に 4 つの同期カウンタを転送するための信号が CMB に追加されたことによる．

FOMP 単位で見ると，FOMP/TSVM のスライス数は FOMP/SC に対して約 5%の増加，動作速度は約 2%の低下であり，ブロック RAM の使用数は変化しなかった．以上から，大きな影響なく TSVM キャッシュを FOMP へ導入しうることが確認された．

参考として，メモリビット数を用いた TSVM キャッシュのハードウェア量について議論する．図 5 で示したように，TSVM キャッシュと従来スヌープキャッシュともに，ウェイ数が 2 で，各ウェイのセット数は 512 である．また，1 ラインに関して，タグが 19 ビット，状態フラグが 4 ビット，そして，4 ワード分の幅が 128 ビットである．よって，従来スヌープキャッシュでは 1 ラインの総ビット数は $19+4+128=151$ ビットになる．また，4 つの同期フラグの幅が 16 ビットなので，TSVM キャッシュにおける 1 ラインの総ビット数は $151+16=167$ ビットである．以上から，TSVM キャッシュのメモリビット数は $2\times512\times167=171{,}008$，従来スヌープキャッシュのメモリビット数は $2\times512\times151=154{,}624$ となる．TSVM キャッシュは従来スヌープキャッシュに対して約 11%のメモリビット数を増加させる．

6. 性能および消費電力に関する評価

6.1 実験環境

前章では FPGA を用いた実装実験を通して，TSVM キャッシュ化はハードウェア量と動作速度に大きな悪影響を与えずになしうることが見込まれた．本章では，以下のプログラム

を用いて消費電力と性能を評価する．

使用したプログラムはループ間にデータ依存のある doacross ループ（DOACROSS）[10]，2 元連立 2 階微分方程式を 4 次のルンゲクッタ法で解くプログラム（RUNGE）[21]，N 個の要素を N/2 のプロセッサが bitonic 法でソーティングするプログラム（BITONIC）[22]，および，radix ソート（RADIX）[23] である．DOACROSS では，イタレーションが各マイクロスレッドにサイクリックに割り当てられ，依存する配列要素が条件同期によって同期通信される．RUNGE では，イタレーション単位で並列化できない実行の大部分を占めるループ内の各文が，各マイクロスレッドに近細粒度で並列化される[24]．そして，文間に存在するデータ依存が条件同期によって解消される．BITONIC と RADIX では，並列処理のフェーズ間にバリア同期が挿入され，マイクロスレッド間でデータを集計するために，いくつかの相互排除が挿入される．

FOMP/SC では，**図 10** に示す，一般共有変数による条件同期，相互排除，および，バリア同期を用いた．条件同期では，生産者がデータの生成を保障するフラグをセットし，消費者がそのフラグをチェックする．また，相互排除とバリア同期に関しては，キャッシュの一貫性制御との親和性と，多くの組み込みプロセッサ[17),25),26)] によってサポートされていることから，本論文では，LL（Load Linked）と SC（Store Conditional）命令による実装方法を採用した[18]．なお，条件同期における同期フラグ上のスピン（図 10 (a) の (1)），および，バリア同期における同期成立フラグ上のスピン（図 10 (c) の (2)）は，プロセッサコアとキャッシュ間で行われるため，バスを駆動しない．また，相互排除のロック部とバリア同期の fetch-and-add 部では，一般共有変数に対する SC 命令において，一貫性制御バスを取得できたプロセッサのみがバスを駆動し，その他はキャッシュ上でスピンする．

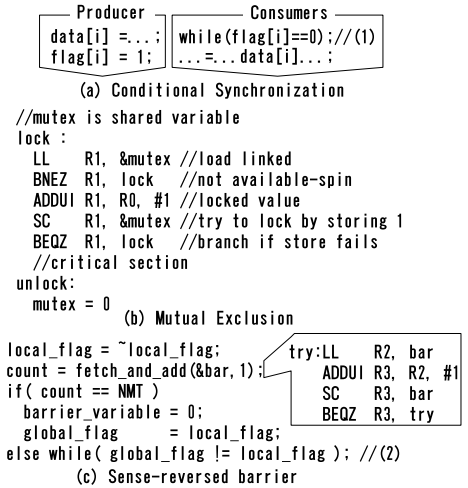


図 10　一般同期変数による同期手法

Fig. 10　Synchronization methods using general shared variables.

消費電力の見積りには Xilinx 社の ISE8.2.02i に標準で付属する XPOWER ツールを用いた．その際に，動作周波数を 33 MHz にしたタイミングシミュレーションを ModelSim SE 6.1c 上で実施し，VCD（Value Change Dump）ファイルを作成した．XPOWER ツールは VCD ファイルから算出した各信号のスイッチング確率を使用しながら消費電力を見積もる．タイミングシミュレーションでは，DOACROSS と RUNGE のデータ数を 128，BITONIC では 8 とし，RADIX に関しては，データ数を 128，桁数を 4 とした．プログラムのコンパイルには gcc-2.95 を使用した（オプションは-O3）．また，4 つのマイクロスレッドを持つ単一スレッドを想定し，各マイクロスレッドを異なる 1 台のプロセッサに割り当てた．プログラムで使用するデータとコードは，すべて，測定の開始前に組み込みメモリへ置いた．測定の開始点で 4 台のプロセッサを同時にスタートさせ，全プロセッサがプログラムを実行し終えた時点を測定の完了点とした．

6.2　実行時間の評価

図 11 に各プログラムの実行時間とその内訳を示す．横軸の TSVM は FOMP/TSVM を，SC は FOMP/SC を意味する．棒グラフは，各プログラムにおける FOMP/SC の結果を 1 とした総実行時間であり，EXE はプロセッサコアの命令実行時間を，STALL はパイプラインのストール時間を表す（各値はプロセッサにわたった平均値）．STALL は，演算の完

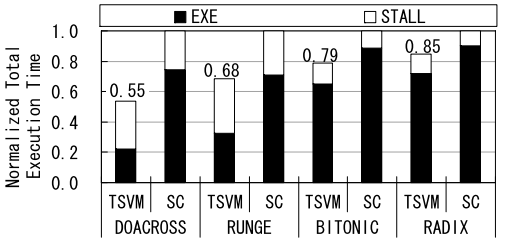


図 11　実行時間の内訳（FOMP/SC = 1）

Fig. 11　Breakdown of execution time (FOMP/SC = 1).

了待ちやキャッシュミス等の一般的な要因によるストール[18] 以外に，同期共有変数と一般変数に対する一貫性制御に起因するストール時間も含む．今回使用した従来スヌープキャッシュではラインを最新値に保つために，一貫性制御バスがキャッシュへアクセスする間（一貫性制御が実施されている間），同一ラインへアクセスしたプロセッサコアはメモリステージでストールされる（TSVM キャッシュに関する 3.4.1 項の説明と同じ理由による）．さらに，2.2 節で述べたように，並列処理の正しさを厳密に保障する理由から，メモリモデルに逐次コンシステンシ[15] を採用しており，一貫性制御の完了までプロセッサコアはメモリステージでストールされる（図 8，図 9 と同様の動作）．

FOMP/TSVM は，FOMP/SC に対して，DOACROSS で 45%，RUNGE で 32%，BITONIC で 21%，RADIX で 15%の実行時間を削減できた．これは，図 10 で示した一般変数の同期通信手法よりも，TSVM を用いた手法のオーバヘッドが小さいからである（詳細は文献 10)，20) を参照されたい）．

STALL を見ると，FOMP/TSVM は FOMP/SC に対し，DOACROSS で 1.21 倍，RUNGE で 1.23 倍，BITONIC で 1.20 倍，RADIX で 1.26 倍に増加させた．これは，FOMP/SC では，各プロセッサコアが同期待ち時に，一般共有変数上でスピンロックを行うため命令を実行し続けるのに対し，FOMP/TSVM では各プロセッサコアが TSVM キャッシュへのアクセスでストールされるからである[10]．

6.3　動的消費電力の評価

動的消費電力の実験結果を**図 12** に示す．プログラムごとに FOMP/SC の結果を 1 として正規化しており，棒グラフが動的消費電力を折れ線グラフが実行時間を意味する．実行時間は消費電力との関係を示すためにプロットしており，値は図 11 と同じである．

FOMP/TSVM は，FOMP/SC と比較して，DOACROSS では 29%，RUNGE で 15%，BITONIC で 6%，RADIX で 3%の動的消費電力を削減できた．また，全体的な傾向として，実行時間の減少にともなって消費電力も減少していることが分かる．つまり，スヌープキャッシュを TSVM キャッシュ化したことによってハードウェア量が増加したにもかかわらず，TSVM キャッシュと同期共有変数の組合せは実行時間を短縮して動的消費電力を削減できたと考えられる．

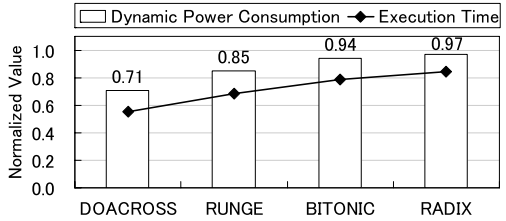


図 12　相対的な動的消費電力と実行時間（FOMP/SC = 1）
Fig. 12　Dynamic power consumption and execution time (FOMP/SC = 1).

また，6.2 節で述べたように，FOMP/TSVM は，FOMP/SC よりもストール時間を増加させる．使用したプロセッサコアは，パイプラインがストールする間，パイプラインレジスタの値が変化しない．XPOWER ツールは，電力の見積りに信号のスイッチング確率を利用するため，信号の変化が少ないストール中の消費電力を低く見積もると考えられる．よって，FOMP/TSVM でストール時間が増加したことも，動的消費電力の削減に貢献していると推測できる．ただし，FOMP における各種ストール（キャッシュミス，一貫性制御，同期待ち，および，その他のハザード等）と電力の関係に関する詳細な評価は今後の課題とする．

6.4　電力性能比の評価

FOMP/SC に対する FOMP/TSVM の電力性能比を評価するにあたり，電力比/速度向上比（$PRSR$）を以下のとおり算出する．

$$PS(i) = \frac{Pw(i)}{Sp(i)}, \quad i = SC, TSVM \tag{1}$$

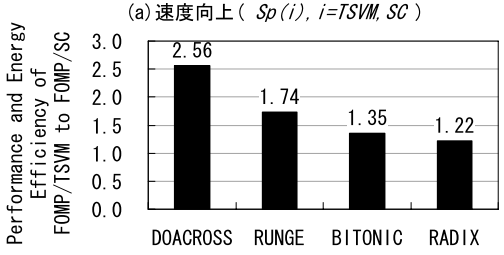
$$\begin{aligned} PRSR &= \frac{PS(SC)}{PS(TSVM)} \\ &= \frac{Pw(SC)}{Pw(TSVM)} \div \frac{Sp(SC)}{Sp(TSVM)} \end{aligned} \tag{2}$$

$Sp(i)$ は FOMP/i での逐次実行に対する並列実行の速度向上，$Pw(i)$ は FOMP/i における動的消費電力である．つまり，$PS(i)$ の値が小さいほど速度向上に要する電力が少ないことになる．$PRSR$ は図 12 に示した数値の逆数と速度向上を式 (2) に代入して算出される．

速度向上と電力比/速度向上比を**図 13** (a)，(b) に示す．FOMP/TSVM は FOMP/SC に対し，DOACROSS で 2.56 倍，RUNGE で 1.74 倍，BITONIC で 1.35 倍，RADIX で 1.22 倍の電力比/速度向上比を示した．並列処理の粒度（同期通信間の処理）は，おおむね，DOACROSS と RUNGE で数〜数十クロック，BITONIC と RADIX で数十〜数百クロック程度であり，DOACROSS や RUNGE のような比較的粒度が小さい並列処理において，TSVM キャッシュと同期共有変数の組合せはより良い電力性能比を示す．

構成	DOACROSS	RUNGE	BITONIC	RADIX
FOMP/TSVM	1.80	1.49	3.06	3.17
FOMP/SC	0.99	1.01	2.41	2.68

(a) 速度向上（ $Sp(i), i=TSVM, SC$ ）



(b) 電力比/速度向上比

図 13　FOMP/SC に対する FOMP/TSVM の電力性能比
Fig. 13　Performance and energy efficiency of FOMP/TSVM to FOMP/SC.

7. おわりに

我々は，オンチップマルチプロセッサにおいて，低オーバヘッドな同期通信をサポートするための同期共有変数 TSVM とその実現方法として TSVM キャッシュを提案してきた．

本論文では，FPGA ベースのオンチップマルチプロセッサにおいて，既存のスヌープキャッシュを TSVM キャッシュ化した際のハードウェアオーバヘッドを明らかにした．その結果，ハードウェア規模と動作速度に対して大きな影響なく導入できることが見込まれた．そして，消費電力の評価を通じ，同期共有変数と TSVM キャッシュの組合せは，ハードウェア量が増加したにもかかわらず，実行時間を短縮して動的消費電力を削減でき，電力性能比を改善した．

今後の課題として，FOMP における各種ストールと消費電力の関係を明らかにすることと，パイプライン・ストールを極力減らすための高度な機能（大容量命令ウインドウ，ノンブロッキングキャッシュ，高精度な分岐予測器，マルチスレッディング等）を持った高機能なプロセッサにおける TSVM キャッシュの評価があげられる．また，並列処理の粒度が小さいと TSVM キャッシュの効果がより大きく，近細粒度から粗粒度にわたった並列処理[24]での効果に関する評価も行う．さらに，FOMP/TSVM を用いたより大規模な応用プログラムに対する評価も今後の課題としてあげる．

参 考 文 献

1) 遠藤　祐，小泉寿男：再利用モジュールのオンライン評価を取り入れたハードウェア・ソフトウェア協調設計方式とその検証，電学論 C，Vol.124, No.11, pp.2249–2259 (2004).
2) 本田晋也，富山宏之，高田広章：システムレベル設計環境：SystemBuilder，信学論，Vol.J88-D-I, No.2, pp.163–174 (2005).
3) Pellerin, D. and Thibault, S.: *Practical FPGA Programming in C*, Prentice Hall (2005).
4) 長谷川揚平，阿部昌平，松谷宏紀，安生健一朗，粟島　亨，天野英晴：動的再構成可能プロセッサを用いた IPsec 向け暗号処理アクセラレータの設計と実装，信学論，Vol.J89-D, No.4, pp.743–754 (2006).
5) Sugawara, T., Ide, K. and Sato, T.: Dynamically Reconfigurable Processor Implemented with IPFlex's DAPDNA Technology, *IEICE Trans. Inf. & Syst.*, Vol.E87-D, No.8, pp.1997–2003 (2004).
6) Kulmala, A., Lehtoranta, O., Hamalainen, T.D. and Hannikainen, M.: Scalable MPEG-4 Encoder on FPGA Multiprocessor SOC, *EURASIP Journal on Embedded Systems*, Vol.2006, pp.1–15 (2006).
7) Ravindran, K., Satish, N., Jin, Y. and Keutzer, K.: An FPGA-Based Soft Multiprocessor System for IPv4 Packet Forwarding, *Proc. 15th International Conference on Field Programmable Logic and Applications*, pp.487–492 (2005).
8) Wang, X. and Ziavras, S.G.: A configurable multiprocessor and dynamic load balancing for parallel LU factorization, *Proc. 18th International Parallel and Distributed Processing Symposium*, pp.234–241 (2004).
9) Yamawaki, A. and Iwane, M.: Organization of Shared Memory with Synchronization on Multi-processor-on-a-chip, *Proc. 9th Int. Conf. on Parallel and Distributed Systems*, pp.83–91 (2002).
10) 山脇　彰，岩根雅彦：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，情報処理学会論文誌，Vol.47, No.2, pp.566–581 (2006).
11) Carter, L., Feo, J. and Snavely, A.: Performance and Programming Experience on the Tera MTA, *Proc. 9th SIAM Conf. on Parallel Processing for Scientific Computing* (1999).
12) Agarwal, A., Bianchini, R., Chaiken, D., Chong, F.T., Johnson, K.L., Kranz, D., Kubiatowicz, J., Lim, B., Mackenzie, K. and Yeung, D.: The MIT Alewife Machine, *Proc. IEEE*, Vol.87, No.3, pp.430–444 (1999).
13) 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).

14) 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).
15) Protic, J., Tomasevic, M. and Milutinovic, V.: *Distributed Shared Memory: Concepts and Systems*, IEEE Computer Society (1997).
16) Yamawaki, A., Chayamichi, H. and Iwane, M.: Easily Customizable Open Soft Processor Cores, *The 1st IEEE Technical Exhibition Based Conference on Robotics and Automation*, pp.79–80 (2004).
17) Kane, G. and Heinrich, J.: *MIPS RISC ARCHITECTURE*, PTR Prentice Hall (1992).
18) Hennesy, J.L. and Patterson, D.A.: *Computer Architecture: A Quantitative Approach, 3rd Edition*, Morgan Kaufmann (2003).
19) Sun: *JBus Architecture Overview Technical Whitepaper Version 1.0*, Sun microsystems (2003).
20) Yamawaki, A. and Iwane, M.: Coherence Maintenances to realize an efficient parallel processing for a Cache Memory with Synchronization on a Chip-Multiprocessor, *Proc. 8th Intl. Symp. on Parallel Architectures, Algorithms and Networks*, pp.324–333 (2005).
21) 茶屋道宏貴，岩根雅彦：バス結合共有メモリ型マルチプロセッサにおけるバリア同期機構の評価，情報処理学会論文誌，Vol.43, No.5, pp.1391–1400 (2002).
22) Lee, J.D. and Batcher, K.E.: Minimizing Communication in the Bitonic Sort, *IEEE Trans. Parallel and Distributed Systems*, Vol.11, No.5, pp.459–474 (2000).
23) 児玉祐悦，坂根広史，佐藤三久，山名早人，坂井修一，山口喜教：細粒度通信機構を用いた Radix ソートの実行，情報処理学会論文誌，Vol.38, No.9, pp.1726–1735 (1997).
24) 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).
25) ARM: *ARM developer's guide* (*ARM11 MPCORE multiprocessor semiconductor*), ARM (2001).
26) Freescale Semiconductor: *MPC8641D Integrated Host Processor Family Reference Manual Rev. 0, 07/2006*, Freescale Semiconductor (2006).

(平成 19 年 7 月 25 日受付)
(平成 20 年 1 月 8 日採録)

山脇　彰（正会員）
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．2007 年同助教．博士（工学）．計算機アーキテクチャ，システム LSI，再構成可能アナログ/ディジタル回路の研究に従事．電子情報通信学会会員．

岩根　雅彦（正会員）
1968 年京都大学工学部数理学科卒業．同年（株）東芝に入社．1980 年同社退職後，豊田工業大学を経て，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，ディジタル回路システムの研究に従事．電子情報通信学会，IEEE Computer Society 各会員．