Vol.2013-ARC-206 No.10
2013/7/31

# ハードウェアトランザクショナルメモリのアーキテクチャに依存しない評価システム

吉山 悠爾[1]　平木 敬[1]

**概要**：並列処理の普及により並行性制御はますます重要になっている一方，並行性制御の手法としてトランザクショナルメモリと呼ばれる手法が提案されている。ハードウエアトランザクショナルメモリ (HTM) はトランザクショナルメモリをハードウェアにより実現したものであり，HTM を実装したいくつかの商用プロセッサが登場している。しかしながら，HTM においては統一的なパフォーマンス評価手法が確立されているとはいえず，またそのパフォーマンスがアーキテクチャに依存しやすいため評価手法を確立しにくい。我々は FPGA 上へ単純で拡張性の高いプロセッサを実装することで，アーキテクチャに依存しない HTM のための統一的な評価環境を構築する。これにより，様々な HTM の構成法を簡単に素早く評価できるようになるだろう。

**キーワード**：トランザクショナルメモリ (TM), ハードウエアトランザクショナルメモリ (HTM), アーキテクチャ評価

# Architecture-Independent Hardware Transactional Memory Evaluation System

Yuji Yoshiyama[1]　Kei Hiraki[1]

***Abstract:*** While the concurrency control is more important to diffuse the parallel processing, transactional memory is proposed as concurrency control methods. Hardware transactional memory (HTM) is transactional memory to be implemented by hardware; some commercial processors have HTM system. However, performance evaluation methods for HTM are not established. Furthermore, it is hard because HTM performance depends on HTM architecture. We design and implement the high extensibility processor on FPGA chips, and we construct the comprehensive evaluation system for HTM. Thus, we are able to evaluate the various hardware transactional memory systems easily and quickly.

***Keywords:*** Transactional Memory(TM), Hardware Transactional Memory(HTM), Architecture Evaluation

## 1. 序論

従来，マイクロプロセッサ性能の向上は半導体技術の発展に伴うクロック周波数の向上，分岐予測を用いた投機的実行などに支えられてきた。また，ビットレベルの並列性を高めることや，スーパスケーラやアウト・オブ・オーダ実行など命令レベルの並列性を高めるマイクロアーキテクチャの採用など，シングルコア性能の向上がマイクロプロセッサの性能向上へと直結していたともいえる。しかしながら，これらの技術によるシングルコアあたりの性能向上は頭打ちを迎え，コンパイラ技術などの向上に伴うプログラム中の命令レベル並列性を高めるための技術もまた限界が表層化してきている。これらの現状に対応するため，ひとつのプロセッサあたりに複数のコアを実装するマルチコアプロセッサが広く普及している。特にひとつのチップ上に複数のコアを実装したチップマルチプロセッサは現在の商用プロセッサの多くを占めるに至っている。

このようなマルチプロセッサを採用したコンピューティ

[1] 東京大学
The University of Tokyo.

© 2013 Information Processing Society of Japan

ングシステムでは，複数のコアが主記憶装置上の記憶領域の一部を共有する場合が多い。この共有記憶領域に対するロードストア操作は，その記憶領域に含まれるオブジェクトに対して一貫性を保ちながら変更操作を行う必要がある。一貫性を保つ手法は今までにいくつか提案されている。

### 1.1 マルチコアプロセッサ上での並行性制御手法

ひとつはロックと呼ばれる並行性制御手法を用いるものである。これは共有オブジェクトに対する操作をする前にロックを獲得し，操作を完了した後にロックを解放することで，一つのプロセスが排他的にその共有オブジェクトに対する変更操作を行う手法である。しかしながら，細粒度のロックを用いれば適切なロック順序を保たなければデッドロックを引き起こす可能性があり，また粗粒度のロックを用いればオーバヘッドにより性能低下を引き起こす可能性がある。このため，プログラマはロックを用いる際に慎重な設計を要求される。

このようなロックの問題点に対して提案された並行性制御手法がトランザクショナルメモリである。トランザクショナルメモリでは，共有オブジェクトを読み込み，変更し，書き込むという一連の操作をひとつのトランザクションと見立てて実行する。この手法ではロックを用いずに並行性を制御することができ，従って原理的にはデッドロックを回避することができる。

トランザクショナルメモリはソフトウェアのみで実装することも可能であるが，ハードウェアを用いて実装することも可能である。トランザクショナルメモリは提案された当初はハードウェアにより実装されることを想定した手法であったが [1]，後の研究によりソフトウェアのみで実装可能であると提案された [2]。前者のようにソフトウェアのみで実装されたトランザクショナルメモリをソフトウェアトランザクショナルメモリ (Software Transactional Memory, STM) といい，後者のようにハードウェアを用いて実装したトランザクショナルメモリをハードウェアトランザクショナルメモリ (Hardware Transactional Memory, HTM) という。STM を用いた並行性制御手法では特殊なハードウェア機構を用いることなくトランザクショナルメモリを実現できる一方，ソフトウェアを用いて処理するために生じるオーバヘッドが大きくなるという問題点がある。一方で HTM を用いれば，それを実現するためのハードウェア機構を実装する必要がある一方で，トランザクショナルメモリによるオーバヘッドを最小限に留めることができる。

### 1.2 トランザクショナルメモリの評価

STM ならば一般のプロセッサに対して実装を行い評価を行うことができる。しかしながら HTM においてはプロセッサに特殊なハードウェアを実装する必要があり，評価を行うことが難しい。いくつかの先行研究では実際にハードウェア実装された HTM に対する評価が行われているが，HTM 自体のパフォーマンスがその構成手法に依存するためそれらを同時にかつ公平に評価することは困難である。またいくつかの先行研究ではソフトウェア実装されたシミュレータによる評価研究も行われているが，それらの結果に対する正確性についての定量的な議論は多くない。

本研究では，そのような HTM において複数の構成手法を公平に評価するための評価環境を提案する。我々の提案の基本的なアイデアは，FPGA(Field Programmable Gate Array) と呼ばれる再構成可能な半導体を用い，拡張性が高くシンプルなプロセッサコアとそれらから簡単に脱着可能なトランザクショナルメモリのモジュールを実装することにある。具体的な提案手法については後の章で述べる。

## 2. トランザクショナルメモリ

序論で触れた通り，トランザクショナルメモリの基本的なアイデアは一つの共有オブジェクトに対する一連の操作をトランザクションと見立て，それらを同時並行的に投機的実行することである。このような手法は楽観的な並行性制御手法であるとも呼ばれるが，これは一連の操作が失敗することを許し，また失敗した場合には再実行可能であることによる。一方でロックに基づく並行性制御手法は悲観的であるとも呼ばれ，操作の失敗を許さないことに由来する。本章ではトランザクショナルメモリについて概説する。

### 2.1 トランザクショナルメモリの実行手順

はじめにトランザクショナルメモリの基本的な実行手順について説明する。図 1 はトランザクショナルメモリの実行手順を図示したものである。この図では左から右へ時間的な変化を表しており ($t$ 軸)，3 つの独立したプロセス $P_1$，$P_2$，$P_3$ が並列に実行されている。基本的にプロセスはメモリ上の共有オブジェクトを読み込み，何かしらの操作を行った後にメモリ上へ書き戻すという操作を行う。プロセス $P_1$ では，A という状態にある共有オブジェクトをはじめの R のタイミングで読み出し，操作を行った後に B という状態になった共有オブジェクトを次の W のタイミングでメモリ上へ書き戻している。この読み出しから書き戻しまでの一連の処理をトランザクション (transaction) と呼び，トランザクションを終了する際に行う書き出しの操作をコミット (commit) と呼ぶ。コミットが成功するためには，トランザクションのはじめに共有オブジェクトを読みだしてから，コミットするまでの間に他のプロセスが同じ共有オブジェクトに対してコミット操作を行っていないことが必要である。

次に複数のプロセスが同時に実行されている際の実行手順を，図 1 におけるプロセス $P_2$，$P_3$ を例に述べる。まずプロセス $P_2$ が状態 B にある共有オブジェクトを R のタイミングで読み出し，次にプロセス $P_3$ が状態 B にある共有

© 2013 Information Processing Society of Japan

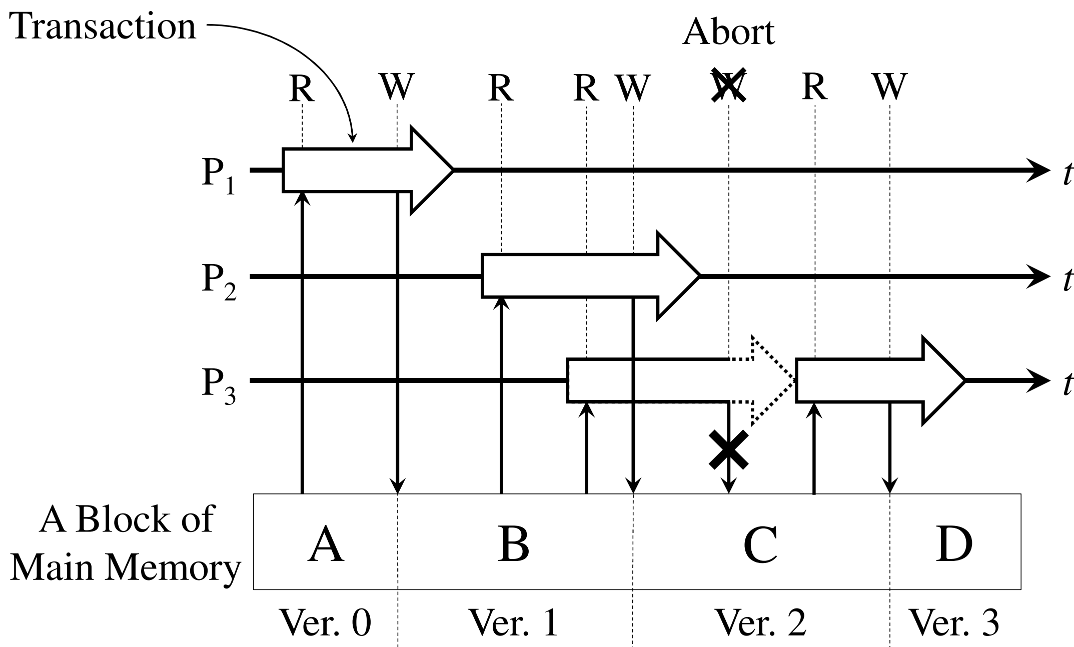


**図 1** トランザクショナルメモリの概要

オブジェクトをさらに次の R のタイミングで読み出す。さらにプロセス $P_2$ は共有オブジェクトを C という状態にした上で W のタイミングでメモリ上に書き戻す。これによりプロセス $P_2$ はコミットが行われたため，プロセス $P_2$ におけるトランザクションは成功する。さらに次の W のタイミングでプロセス $P_3$ が変更を施した共有オブジェクトを次の W のタイミングでメモリ上に書き戻そうとするが，B の状態にある共有オブジェクトを読みだしてからこのタイミングまでにプロセス $P_2$ がコミットを成功させているため失敗する。コミットが失敗するとこのトランザクションにおける変更処理はすべて破棄される。これをアボート (abort) と呼ぶ。さらにトランザクションがアボートすると，プロセスはトランザクションを初めからやり直す。これ一連のやり直しの処理をロールバック (rollback) と呼ぶ。プロセス $P_3$ はトランザクションが失敗したため，アボート，ロールバック処理を行い，またトランザクションを開始する。今度は R のタイミングで状態 C にある共有オブジェクトを読み出し，状態 D へと変更した上で次の W のタイミングでメモリ上に書き戻す。今度はコミットまでに他のプロセスによるコミット操作が行われなかったため，トランザクションは成功する。

### 2.2 トランザクショナルメモリの競合管理

前節で述べたように，あるプロセスが実行するトランザクションのコミットが成功するためには他のプロセスがコミットを成功させていないことが必要である。すなわち，一つのトランザクションは一つの不可分な処理として行われる必要がある。このような性質を原子性 (atomicity) という。また，並列に実行されるトランザクションは他のトランザクションに影響を与えず，また影響を受けない。このような性質を独立性 (isolation) という。

先述の 2 つの性質を保ちながら，かつ共有オブジェクトに対する一貫性 (consistency) を保証する必要がある。すなわち，並列に実行される他のトランザクショナルに実行されるメモリ操作の間で競合する操作を管理しなければならない。これを保証するための一般的な手法のひとつは，共有オブジェクトに対してメタデータを付与することである。

メタデータの表現方法の一つは共有オブジェクトに対してバージョンを付与するものである。すなわちトランザクションが書き込むを行うタイミングで読み込み時のバージョンと一致しているか否かを確認し，一致しているならばデータを書き込むと同時にバージョンを新しいものへと更新する。不一致ならばトランザクションはアボートする。

このバージョンに基づく競合管理手法は STM，HTM のどちらにおいても実現可能な手法のひとつである。一方で HTM に議論の焦点を絞れば，キャッシュラインに読み出しビット (read bit)，書き戻しビット (write bit) というメタデータを付与することもできる。この場合のメタデータは，キャッシュライン上のキャッシュタグとして実装される。この 2 つのビットデータを用いることで，キャッシュラインに対する一貫性を破壊するような操作を検出できる。

## 3. 関連研究

Moore らは"LogTM: Log-based Transactional Memory"[5] において，はじめに先行研究で提案されたいくつかのトランザクショナルメモリデザインを分類している。この分類はバージョン管理手法と競合検出手法の 2 つ

の観点で行われている。

第一にバージョン管理手法 (version management) については eager version management と lazy version management と呼ばれる 2 種類の手法に分けられている。Eager version management では書き込みが行われる際に，まず古いオブジェクトを別の領域に退避させ，その後に書き込みを行う。対して lazy version management では書き込みが行われる際に，新しいオブジェクトを別の領域に書き込む。

Eager version management ではコミットを行うためには古いオブジェクトが退避している領域を破棄するだけでよいが，ロールバック処理の際に退避している領域から古いオブジェクトを書き戻す必要がある。Lazy version management では逆にロールバック処理は領域を移るだけでよいが，コミットの際には新しいオブジェクトを書き戻す必要がある。これらはコミット回数とアボート回数とのトレードオフの関係にあるといえる。

第二に競合検出手法 (confliction detecton) については eager confliction detecton と lazy confliction detecton と呼ばれる 2 種類の手法に分けられている。Eager confliction detecton と lazy confliction detecton の違いは，どのタイミングで競合検出を行うかにある。Eager confliction detecton ではトランザクションに含まれるすべてのメモリアクセス命令に対して競合を検出する。対して lazy confliction detecton ではコミット時のメモリアクセス命令についてのみ競合を検出する。

Lazy confliction detecton ならば競合検出によるオーバヘッドを最小限に留めることができる一方，トランザクションがアボートした際にはオーバヘッドが大きくなる。一方で eager confliction detecton ならば，競合が発生した時点から可能な限り素早く検出できるため，アボートが起こる際でもオーバヘッドを小さくすることができる。

### 3.1 いくつかの商業的なデザインについて

商業的なプロセッサアーキテクチャのデザインではハードウェアによるトランザクショナルメモリのサポートを謳っているものも存在する。

ひとつめは Sun Microsystems 社 (現 Oracle 社) の Rock というコードネームで呼ばれたプロセッサである [7]。Rock は 2008 年の ISSCC にて技術的な発表がなされたが，ハードウェアトランザクショナルメモリ以外にもスカウトスレッドなど意欲的にいくつかの先進的なアーキテクチャを導入したプロセッサデザインであった。

次に IBM 社の BlueGene/Q を紹介する [8]。この BlueGene/Q というプロセッサは PowerPC A2 をベースアーキテクチャにもつプロセッサで 1 ノードあたり 16 計算コアが実装されている。BlueGene/Q では 2 次キャッシュに embedded DRAM という技術を利用し 32MB という大きな容量を実装することを可能にした。その 2 次キャッシュは Multi-Versioned Cache と呼ばれバージョン情報をもつキャシュとなっており，これによりハードウェアトランザクショナルメモリの機構を実現している。

最後に第 4 世代 Core i シリーズプロセッサである Haswell アーキテクチャについて紹介する。このプロセッサでは第 3 世代の Core i シリーズである Ivy Bridge からさらにいくつかのマイクロアーキテクチャレベルの改良が加えられている。またこのプロセッサには Transaction Synchronization Extensions(TSX)[11] と呼ばれる拡張命令セットが実装されている。この TSX がサポートする命令を大別すると Hardware Lock Elision(HLE) と Restricted Transactional Memory(RTM) という 2 種類に別れる。HLE は従来のロックベースの並行アプリケーションを簡単にトランザクショナルメモリを利用したアプリケーションへと変更できるようにする手法である。一方，RTM は新規にトランザクショナルメモリを用いた並行アプリケーションを開発するために用いられることを想定した手法である。

## 4. 提案手法

さて，序論でも述べたとおり HTM の評価についてはソフトウェアによるシミュレーションベースが主流であり，また実際にハードウェア実装されたシステムに対する評価は前章で触れたいくつかの商業的なデザインなど少数である。ソフトウェアによるシミュレーションには Martin らによるマルチプロセッサシミュレータである GEMS[4] などが用いられることが多い。また性能を評価するためのベンチマークアプリケーションとしては GEMS に付属するマイクロベンチマークや Minh らによる STAMP[6] が用いられることが多い。

一方で現在行われている多くの評価結果を公平に比較することは難しい。ハードウェア実装された HTM システムは，HTM 以外の部分においても多様なアーキテクチャによってデザインされており，単純にトランザクショナルメモリとしてのアーキテクチャ評価を行えているとはいえない。またソフトウェアによるシミュレーション研究においても，実際に実装された環境が存在しなければそのシミュレーション結果に対する正確性を担保することは難しい。

### 4.1 概要

従って我々はハードウェアとして実装されていること，そして HTM 以外の部分におけるアーキテクチャからの独立性が高いという 2 点を特徴とする評価システムを提案する。これにより，この評価システムへ実装された HTM については高い公平性をもって評価結果を比較することができる。また，各モジュールを単純な構成にし，かつモジュール間の独立性を高めることで，HTM のための機構を簡単に実装できるように配慮した。

このシステムは FPGA と呼ばれる再構成可能な半導体上

© 2013 Information Processing Society of Japan

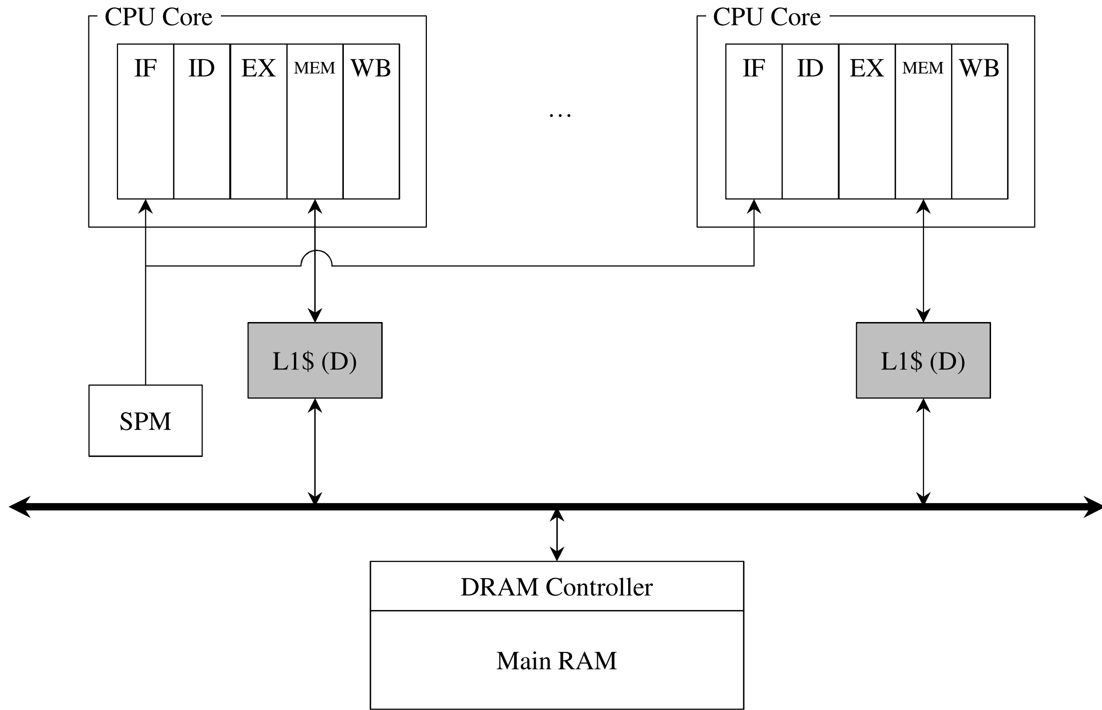


**図 2** 提案するマイクロアーキテクチャ

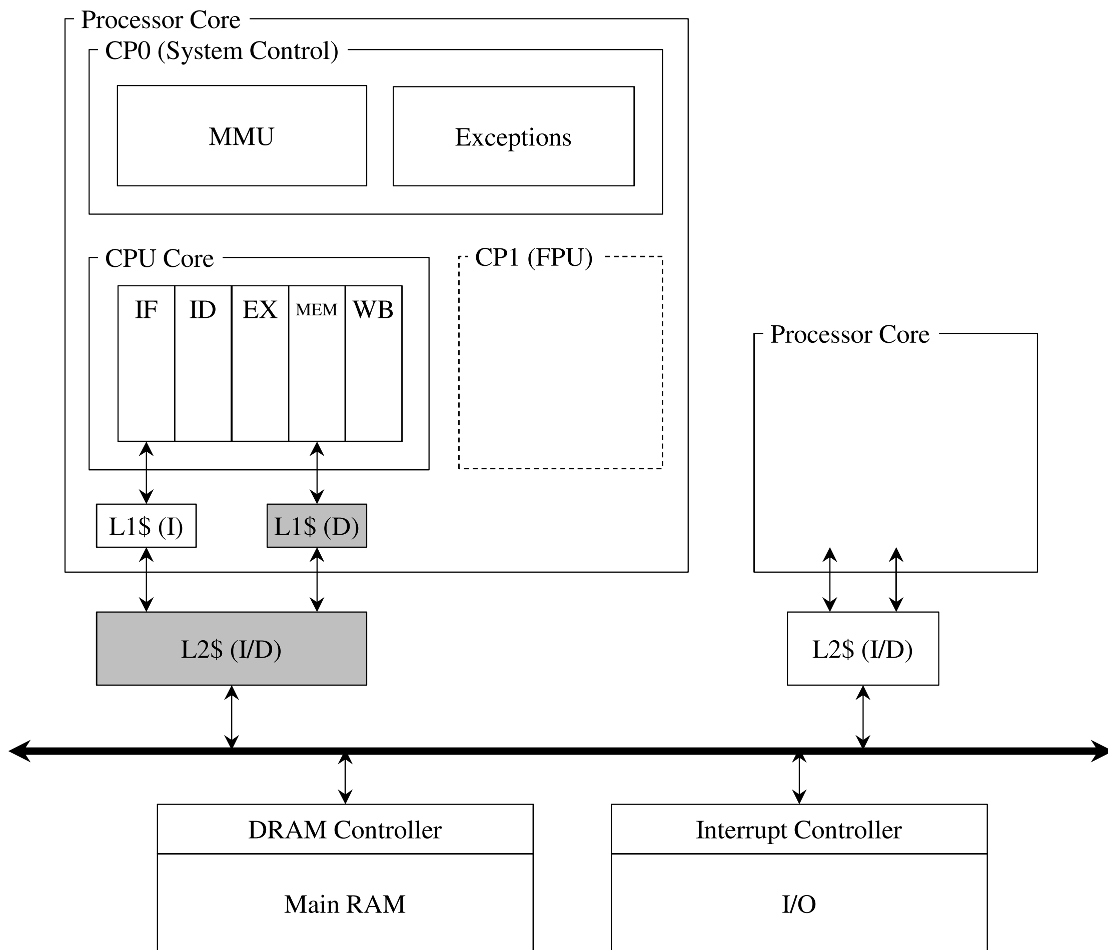


**図 3** MIPS R4000 のマイクロアーキテクチャ

© 2013 Information Processing Society of Japan

に実現している。そのため実際にハードウェアを実装するために高いコストを払うことなく，市販されているFPGAチップや基盤を用いることができる。ただし，FPGAチップや基盤ごとの物理的な差については考慮する必要がある。

### 4.2 アーキテクチャに依存しないこと

「アーキテクチャに依存しない」ということはどういうことであるかを中心にして我々の提案するシステムの構成について議論する。プロセッサアーキテクチャには，「どのような命令群を実行できるか」ということを定めた命令セットアーキテクチャ (Instruction Set Architecture, ISA) と,「命令をどのように実行するか」ということを定めたマイクロアーキテクチャがある。

まずHTMはISAには依存しないことについて議論する。HTMの実行は命令群において最低限のマルチプロセッササポートが存在すれば実現可能であるからである。ここでISAが満たすべき最低限のマルチプロセッササポートとは原子性をもつメモリ操作命令をいう。このような命令にはいくつか種類があるが，Test-And-Set(TAS)命令やCompare-And-Swap(CAS)命令，Load-Linked/Store-Conditional(LL/SC)命令などがある。これらはトランザクショナルメモリだけでなく従来のロックに基づく手法など並行性制御のために広く用いられている。

トランザクショナルメモリの性能はマイクロアーキテクチャに依存するところが大きい。従って今回提案するシステムにおいてはプロセッサはなるべく単純な構造をもつようにし，さらに拡張性を高めることで様々なマイクロアーキテクチャを追加実装することができるようにする。それでは，具体的にどのようなプロセッサを実装すべきであろうか。基本的な指針として，現在一般に広く普及しているプロセッサの構造とFPGAにおける物理的制約を中心に議論を進める。

現在広く利用されているプロセッサの多くはどのようなマイクロアーキテクチャを実装しているだろうか。我々はx86，SPARC，Power，Alpha，MIPSなど様々なマイクロアーキテクチャを精査し，次のような機能を現代的なプロセッサがサポートすべき最低限のマイクロアーキテクチャとした。

- 命令パイプライン
- キャッシュメモリ機構
- コモンバス形式のインターコネクション

ここに挙げたもの以外では，分岐予測機構やアウトオブオーダ実行をサポートする機構などは広く普及しているといえる。しかしそれらは上に挙げた機構を実装したプロセッサに対して追加的に実装できることから，最低限という性質を満たさないと判断した。

またFPGAの物理的制約を考慮すれば，シングルコアあたりの規模を大きくすると同時に実装できるコアの数が減少する。従ってなるべくプロセッサコアの規模を小さくする必要がある。例えばCISC型のプロセッサは命令デコーダが複雑であるなど構造が単純とはいえず今回の目的に反している。逆にRISC型ならば構造を単純化しやすく今回の目的に合致するといえる。

以上の議論を踏まえた上で実装すべきISA及びマイクロアーキテクチャについて議論する。ISAについては，原子性をもつメモリ操作命令を実装していればよいのだが，これらを満たすアーキテクチャはSPARC，POWER，MIPSなど様々な種類があり，また独自のアーキテクチャを定義することも可能である。これらの事実を考慮したとき，我々はベースとなるISAとしてMIPSを選択した。なぜならば，RISC型のプロセッサの中では最も単純で基本的な命令セットのひとつであり，かつ今までに多くのコンパイラやツール群が開発されてきた経緯があるからである。マイクロアーキテクチャについては図2のように先述の3つの機構を含み，かつ各パイプラインステージをモジュール化したものを提案する。単純な構造をもつRISC型のマルチコアプロセッサの実例としてMIPS R4000[9],[10]というプロセッサのマイクロアーキテクチャを図3に示す。我々の提案するプロセッサも命令セットとしてMIPSをベースとしたことからレジスタファイルの構成などはMIPS R4000に基づいた設計になっている。

| Tag | V | R | W | Data |
| --- | --- | --- | --- | --- |
| | | | | |
| | | | | |
| | | | | |
| ⋮ | | | | |
| | | | | |
| | | | | |

**図 4** キャッシュメモリの構成例

### 4.3 記憶装置

我々の提案するシステムはHTMの評価を行うものであるから，記憶装置の構成は最もクリティカルに性能に直結するといえる。従って1次キャッシュメモリを設けるという条件以外については制約を加えず，様々な構成を採用できるようにする。

ゆえに様々な構成において評価を行うことができるが，一例として図4のようなHTMを実行できるキャッシュメモリの構成を提示する。この例ではlazy version management，eager confliction detecton を用い，また読み出しビット，書き戻しビットを用いた構成とする。あるプロセスにおけるトランザクションがコミット要求を出した場合，Vビット (valid bit) を確認しキャッシュラインが有効

なものかを判断する。有効なキャッシュラインであれば，まず他のキャッシュに同一のキャシュラインが存在するならば無効化するように通知をする。その通知が完了した後，書き戻しビットが立っているならばコミットすべきデータを主記憶装置上へと書き込む。なおこの一連の動作の間は共有バスのアクセス権限を保持する。また，キャッシュラインに対するすべてのロードストア命令を実行する際に V ビットを確認し，キャッシュラインが無効ならばトランザクションをアボートする。この際に主記憶装置上から値を再度読み込むことでロールバックを行う。

### 4.4 評価手法

FPGA の物理的制約から，システム全体を動作させるためのクロック周波数を向上させることが難しい場合がある。従ってパフォーマンスを評価する場合にはパフォーマンスカウンタなどを用いて，実時間ではなく実行サイクル数で評価する。また，アボート回数やコミット要求数に対する成功コミット数の比などを用いて公平性の評価を行うことができる。

異なる FPGA チップ或いは周辺回路を含む FPGA 基板を用いると，それらの評価結果を公平に比較することはできない。しかしながら，同一基板上での評価は公平に比較することができ，FPGA 上ではハードウェア定義言語で定義された回路を構成することが可能であるため，必要な比較対象となるシステムをそれぞれ同一基板上で評価することができる。

### 4.5 考察

プロセッサコアをハードウェア定義言語である Verilog によって実装し，Altera 社の Quartus II 13.0 を用いて論理合成を行った。Altera 社の FPGA チップである Cyclone II(EP2C35F672C6N, LE 数 33216) をターゲットとしたところ，コアあたりの LE 数は 2021 となった。また，Altera 社の Stratix IV(EP4SE820) と呼ばれる FPGA チップでは LE 数が約 82 万であり，キャッシュメモリなどの記憶装置や周辺機構などにもよるが，HTM のスケーラビリティの評価を幅広く行えるといえる。

## 5. 結論

本研究ではトランザクショナルメモリの評価上の難しさを解消することを目的とし，HTM を FPGA と呼ばれる再構成可能な半導体上にハードウェアとして実現する評価システムを提案した。さらに我々の提案ではアーキテクチャに依存しないシステムを構成することで，得られる評価結果を比較しやすいものとした。定性的な議論には留まったが，アーキテクチャに依存しない評価システムを用いることで，従来よりも HTM の評価を公平に比較しやすくなるといえた。

### 参考文献

[1] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: architectural support for lock-free data structures. In *Proceedings of the 20th annual international symposium on computer architecture (ISCA'93)*, pages 289-300, San Diego, California, USA, 1993.

[2] Nir Shavit and Dan Touitou. Software transactional memory. In *Proceedings of the 14th annual ACM symposium on Principles of distributed computing (PODC'95)*, pages 204-213, Ottowa, Ontario, Canada, 1995.

[3] Lance Hammond, Vicky Wong, Mike Chen, Brian D. Carlstrom, John D. Davis, Ben Hertzberg, Manohar K. Prabhu, Honggo Wijaya, Christos Kozyrakis, and Kunle Olukotun. Transactional Memory Coherence and Consistency. In *Proceedings of the 31st annual international symposium on Computer architecture (ISCA'04)*, pages 102-113, Washington, DC, USA, 2004.

[4] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, David A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. In *ACM SIGARCH Computer Architecture News - Special issue: dasCMP'05*, pages 92-99, New York, NY, USA, November 2005.

[5] Kevin E. Moore and Jayaram Bobba and Michelle J. Moravan and Mark D. Hill and David A. Wood. LogTM: Log-based Transactional Memory. In *12th Annual International Symposium on High Performance Computer Architecture (HPCA-12)*, pages 254-265, Austin, TX, USA, February 2006.

[6] Chi Cao Minh, JaeWoong Chung, Christos Kozyrakis and Kunle Olukotun. STAMP: Stanford Transactional Applications for Multi-Processing. In *Proceedings of The IEEE International Symposium on Workload Characterization (IISWC'08)*, pages 35-46, Seattle, WA, USA, September 2008.

[7] Shailender Chaudhry, Robert Cypher, Magnus Ekman, Martin Karlsson, Anders Landin, Sherman Yip Sun, Håkan Zeffer and Marc Tremblay. Rock: A High-Performance Sparc CMT Processor. In *IEEE Micro 29(2)*, pages 6-16, Los Alamitos, CA, USA, March 2009.

[8] Amy Wang, Matthew Gaudet, Peng Wu, Jose Nelson Amaral, Martin Ohmacht, Christopher Barton, Raul Silvera and Maged Michael. Evaluation of Blue Gene/Q Hardware Support for Transactional Memories. In *the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*, pages 127-136, Minneapolis, MN, USA, September 2012.

[9] Dominic Sweetman. See MIPS Run Linux Second Edition. Morgan Kaufmann Publishers, Burlington, MA, USA. 2007.

[10] Joe Heinrich. MIPS R4000 Microprocessor User's Manual Second Edition. MIPS Technologies, Inc., 1994. `http://groups.csail.mit.edu/cag/raw/documents/R4400_Uman_book_Ed2.pdf`

[11] Intel Corporation. Intel(R) Architecture Instruction Set Extensions Programming Reference, Chapter 8. Intel Corporation., Santa Clara, CA, USA. February 2012. `http://software.intel.com/sites/default/files/m/3/2/1/0/b/41417-319433-012.pdf`

© 2013 Information Processing Society of Japan