情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-SLDM-144 No.32
Vol.2010-EMB-16 No.32
Vol.2010-MBL-53 No.32
Vol.2010-UBI-25 No.32
2010/3/27

組込システムにおけるソフトウェアトランザクションメモリの検討

田村祐典[†1]　中本幸一[†1]　山田晋平[†1]

近年、プロセッサのマルチコア化が進む中で新たな並行処理同期法としてトランザクションメモリという技術が注目されるようになってきている．トランザクションメモリとはデータベースの分野で使用されるトランザクションをメモリのデータ管理に応用したものである．トランザクションメモリは従来の並行処理における共有データの同期制御法であるロックベースの排他制御の代替手法とすることが研究される．従来の同期処理手法である排他制御は危険領域を大きく取るか細かく取るかでトレードオフがある．大きく取る場合はプログラムは見やすくなるがスループットが低下する．反対に細かく取ればスループットは増加するがプログラムが見難くなり、予期せぬデッドロックの可能性も高くなる．トランザクションメモリは，一貫性が損なわれない範囲で従来危険領域とされる処理においてのスレッドの並行実行を許す．そのため，排他制御の問題の解決法として考えられている．プロセッサのマルチコア化は将来の組込システムの分野にも適応可能であると考えられ，汎用コンピュータの場合と同様にトランザクションメモリを使用することで利益が得られると考えられる．本研究はこのような観点の下，組込システムにおいて，トランザクションメモリの実装方式を検討するものである．

Deliberation of applying software transactional memory for embedded system

Yusuke Tamura,[†1] Yukikazu Nakamoto[†1]
and Shimpei Yamada[†1]

Recently, as processor has become multi-core, transactional memory has been getting attention as a new memory synchronization method. The transactional memory is a memory management technology based on transaction used in field of database. It has been researched for expecting that it will be an alternative to a lock-based exclusive control in the traditional memory synchronization. The traditional exclusive control has trade-off between coarse-grained locking or fine-grained locking. The coarse-grained locking becomes easy to see the program, but throughput (concurrency) decreased. The fine-grained locking, increase throughput, but it becomes difficult to see the program and prone to programming errors. The transactional memory provides a solution to this problem because it allow multiple task to run concurrently in critical section as far as the consistency of data is not lost. In the field of embedded system, multi-core processer can be used too, so embedded system can benefit from transactional memory. Under this perspective, this study focus on applying transactional memory for embedded system, and design transactional memory for implemention into embedded operating system.

1. はじめに

近年，プロセッサのマルチコア化が進む中で新たな並行処理同期法としてトランザクションメモリという技術が注目されるようになってきている 4)．トランザクションメモリとはデータベースの分野で使用されるトランザクションのデータ管理法をメモリ管理に応用したものである．トランザクションは並行処理に対して一貫性を持たせることを目的とする技術であり，その特徴は一貫性が損なわれた場合に，トランザクション内で行った更新処理をロールバック操作により，更新を無効にできることである．このトランザクションをメモリ管理に応用し，従来の並行処理における共有資源の同期制御法である排他制御の代替手法として研究されている．

従来の並行処理同期制御法であるロックベースの排他制御は危険領域（共有資源にアクセスする操作列） を大きく取るか，細かく取るかによってトレードオフがある．危険領域を大きく取る場合，プログラムの内容把握が容易になるが並行性が低下する．反対に危険領域を細かく取れば，並行性は増加するがプログラムが見難くなり，予期せぬデッドロックの可能性も高くなる．この排他制御の問題の解決法として考えられているのがトランザクションメモリである．排他制御は危険領域における操作に対して一貫性を保証するために，危険領域をただ一つのスレッドしか実行できないようにしているのに対し，トランザクションメモリは危険領域を同時に複数のスレッドが実行することを許す．トランザクションメモリでは

†1 兵庫県立大学大学院応用情報科学研究科
Graduate School of Applied Infomatics, University of Hyogo

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.32
Vol.2010-EMB-16 No.32
Vol.2010-MBL-53 No.32
Vol.2010-UBI-25 No.32
2010/3/27

従来，危険領域とされた領域で一貫性を損なうような操作があれば，その領域内で行われた処理を無効とすることができるからである．

プロセッサのマルチコア化は将来の組込システムの分野にも適用可能であると考えられ，汎用コンピュータの場合と同様にトランザクションメモリを使用することで利益が得られると考えられる．本研究はこのような観点の下，組込システム上にトランザクションメモリの機能を実装することを目標とした．

トランザクションメモリは，数多くの実装方式が研究されている．本稿では，目標を達成するための段階として，これらの調査を行い，組込システムに適した機能の設計・評価を行うものである．

2. トランザクションメモリ

2.1 トランザクション

データベースの分野でトランザクションは，共有データの一貫性を保持する技術として使用されている．トランザクションとは，操作の列のことであり，操作列は論理時間的に一点で処理される．トランザクションはデータベースなどの共有データを，ある一貫した状態から他の一貫した状態に遷移させるために使用されるものである 2)．

トランザクションはコミット，アボート，ロールバックという機能を持つことでデータの一貫性を保つことが出来る．トランザクションが正常な（データの一貫性を保った）状態のままシステムを終了する操作をコミット（Commit） といい，データの一貫性が損なわれた（以降，コンフリクトと呼ぶ）場合に，更新処理を全て取り消す操作をアボート（Abort）という．トランザクションがアボートした場合やトランザクションを中断する場合は，トランザクションの処理によって更新されたデータ状態は初期状態に戻されなければならない．トランザクション実行前の初期状態に戻すことをロールバック（Roll Back） という．

これらの機能を実現するための原理であるが，各スレッドが実行するトランザクションは，データを更新前の状態と更新後の状態に分けて持つことで実現する．こうすることで，もしトランザクション内の処理においてコンフリクトを起すような操作があった場合，共有データの状態を更新処理を行う前の状態に戻す手段を得る．トランザクション内の一連の処理にコンフリクトがなかった場合，更新は有効となるので，更新処理後のデータを確定させる．

これらの処理を**図 1** に示す．

これらの機能を，メモリの共有データの一貫性管理に応用したものがトランザクションメモリである．

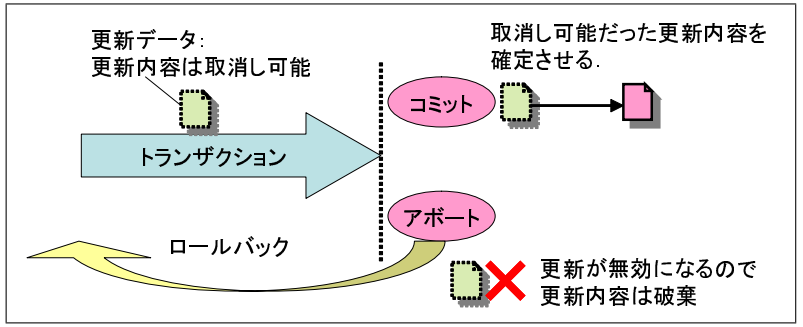

図 1 トランザクションの機能
Fig. 1 function of transaction

2.2 HTM と STM

トランザクションメモリの構築法の大きな分類として，その機能をハードウェアで実現させるのか，ソフトウェアで実現させるのかという分類がある．

HTM（Hardware Transactional Memory）は，トランザクションメモリのキーとなる技術であるコンフリクト検査やバージョン管理をハードウェアで行う構築法のことで，具体的には，トランザクション専用キャッシュなどを利用する 3)，4)．コンフリクト検査やバージョン管理の処理はオーバヘッドが大きいので，ハードウェアでこれらを行う HTM は，これらをソフトウェアで行う STM よりも処理性能が高くなる．しかしハードウェアを用いるので，そのハードウェアの容量により，並行して動作できるトランザクションの数やトランザクション内で使用できる共有データの数やサイズに制限がかかる．

STM（Software Transactional Memory）は，トランザクションメモリの機能を全てソフトウェアで実現させる手法である 5)．STM は，ソフトウェアでバージョン管理やコンフリクト検査を行うので，ソフトウェアの設定次第で HTM が受けたような制限は変えられる．また，ハードウェアに依存しないので，様々なマシンに移植できるというメリットもある．しかしバージョン管理やコンフリクト検査などのオーバヘッドの大きな処理をソフトウェアで行うので，HTM に比べ，オーバヘッドが大きい．このことから，理論上は HTM で受けたような制限は受けないものとされるが，処理のオーバヘッドが大きいことから，メモリ容量や CPU 性能により，実際は制限を受けることとなる．

本研究では，STM を採用している．その理由としては，HTM のように，組込システムにトランザクションメモリ専用のハードウェアを搭載させることは，コスト面から現実的で

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.32
Vol.2010-EMB-16 No.32
Vol.2010-MBL-53 No.32
Vol.2010-UBI-25 No.32
2010/3/27

はないと考えられるので，ソフトウェアで実現させることを本研究では選択した．ソフトウェアで実現することで，ハードウェアに依存しないので，このトランザクションメモリの機能を移植できる機器が増えるというメリットが挙げられる．

3. STM の分類

STM には，いくつかの機能を設計する上で，その手法に分類がある 3)．それらのうち，トランザクションメモリで主要な機能である，コンフリクト発見法とバージョン管理について述べる．

3.1 コンフリクト発見法

トランザクションメモリは，もし，トランザクション実行中にコンフリクトが起きた場合，アボートを行ってデータの一貫性を保つために，コンフリクトが起きたことを発見しなければならない．コンフリクトの発見の仕方には，ペシミスティックコンフリクト発見法とオプティミスティックコンフリクト発見法の二つの分類がある．

ペシミスティックコンフリクト発見法は，トランザクション実行中に読取りや書込みがある度にコンフリクトが起きていないか検査をするコンフリクト発見法である．この方式のメリットは，もしトランザクション実行中でコンフリクトの原因となる操作があった場合に，すぐにコンフリクトを発見でき，その後の無駄な操作（コンフリクトすることが決まっているので） を排除できることである．この方式のデメリットは，検査回数が多くなってしまうことと，順序依存のあるコンフリクトの場合，コードが前に進めない状態になってしまう可能性があり，この問題を対処しなければならないことである．

オプティミスティックコンフリクト発見法は，トランザクション終了時（コミット/アボートをする直前） にコンフリクトが起きていないか検査をするコンフリクト発見法である．この方式のメリットは，検査の回数が一回で済むことと，順序依存のあるコンフリクトについて対処法を考えなくて済むことである．この方式のデメリットは，もしトランザクション実行中にコンフリクトの原因となる操作がなされたとしても，そのコンフリクトは，トランザクション終了直前まで発見されないので，コンフリクトの原因となる操作以降の無駄な操作が行われることである．

3.2 バージョン管理

トランザクションメモリは，もし，トランザクション内でコンフリクトが発生すると共有データの一貫性を維持するために，トランザクション内で行った処理をロールバックにより無効にしなければならない．ロールバックによって共有データの状態はトランザクション開始前の状態に戻るわけであるが，このロールバックをするためには，システムはトランザクション開始前のデータの状態とトランザクション内で変更したデータの状態を分けて管理しておく必要がある．バージョン管理とはこの共有データの状態（バージョン） を管理することである．共有データのバージョンの管理法にはアンドゥログ方式とライトバッファ方式という二つの方式がある．

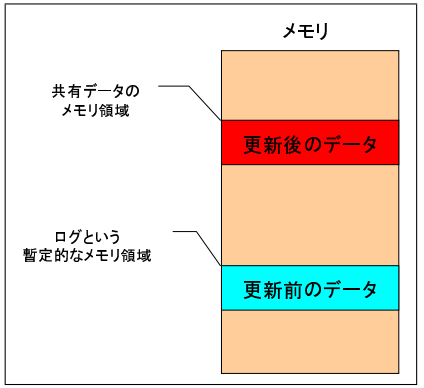

図 2 アンドゥログ方式
Fig. 2 principle of UndoLog

アンドゥログ方式は，トランザクション内で書込み（共有データの更新処理） があった場合，**図 2** に示すように，更新前の値をログと呼ばれるメモリの仮領域に保持させておき，共有データが存在するメモリ領域に直接更新後の値を書込むバージョン管理方式である．この方式では，更新処理を元々対象があった共有メモリ領域に書込むので，一貫性の保持のために，あるトランザクションが更新したデータはコミットするまで他のトランザクションからは見えない状態にする必要がある．この方式のメリットは，直接対象となる共有データのメモリ領域を更新するため，コミット時に共有データへコミットした内容を反映させるためのオーバヘッドがかからないことである．また, コミット時の処理はログのメモリ領域の解放だけの操作で済むことである．この方式のデメリットは，更新データの元の値をログ用のメモリ領域に保管させているので，トランザクションがアボートしてロールバックする場合，ログ用のメモリ領域に書かれている元の値を更新を行った共有データのメモリ領域に戻すオーバヘッドがかかることである．また，トランザクションがある共有データに書込みを行った場合，一貫性保持のため，その共有データを他のトランザクションが読取らせないよ

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.32
Vol.2010-EMB-16 No.32
Vol.2010-MBL-53 No.32
Vol.2010-UBI-25 No.32
2010/3/27

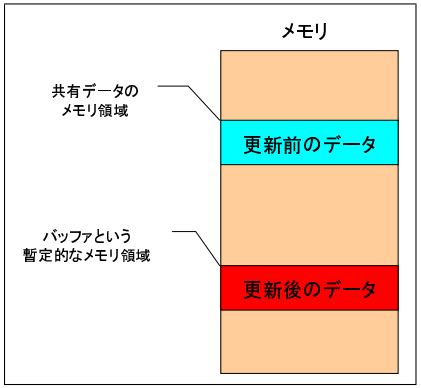

図 3 ライトバッファ方式
Fig. 3 principle of WriteBuffer

うにする必要があり，データの所有権を握らなければならない（排他制御を行わなければならない）．書込みを行ったトランザクションが長期間そのデータの所有権を保持してしまう可能性がある．

ライトバッファ方式は，トランザクション内で書込み（共有データの更新処理）があった場合，**図 3** に示すように，直接共有データのメモリ領域を更新するのではなく，書込みを行う共有データに対応する暫定的な仮のメモリ領域（バッファと呼ぶ）を作成し，そのバッファへ更新する値を格納する．こうすることでバージョン管理を行う．

この方式のメリットは，共有データの更新を仮領域に書込みをしているため，実際に更新が行われてはいないので，アボート時に共有データをロールバックするためのオーバヘッドがかからないことがある．また，トランザクション内での更新が仮領域になされることで，他のトランザクションがその書込まれたデータに対して読取りを行う場合に，待たされることが無い．優先度の低い処理を行うトランザクションが，先にある共有データに書込みを行い，長期間そのデータの所有権を開放しないような場合に優先度の高い処理が待たされずに済む．この方式のデメリットは，データの更新を暫定的な仮のメモリ領域に対して行うので，トランザクションをコミットさせる場合，更新データを実際の共有データのメモリ領域に反映させるオーバヘッドがかかることである．

4. 機 能 仕 様

本研究で設計した STM の仕様を述べる．

- `tk_tx_set_up()`
  この関数でトランザクションメモリの使用準備を行う．システム内でトランザクションメモリで使用するためのデータの作成・初期化を行う．
- `tk_tx_var_regist(void *`*pointer_to_shared_data*`)`
  この関数でトランザクションの対象とする（トランザクション内で読取り，書込みを行う）共有データを登録する．本研究では，使用する共有データを明示的に指定することで，トランザクションメモリの動作を軽量化する工夫をするため，このような関数を実行する必要がある．文献 3) のトランザクションメモリにおいてもこのような登録を必要としているが，登録する共有データは，プログラム上の静的なデータのみである．本研究で設計したトランザクションメモリでは，動的に生成した（メモリ領域を割り当てた）データも扱うことができる．
- `TXBEGIN( unsigned int` *transaction_ID* `)`
  このマクロ関数でトランザクションを開始する．引数には，開始されたトランザクションの識別子が代入される．この ID を使用して，どのトランザクションに対して読取りや書込みを行ったかを記録していく．
- `TXEND( unsigned int` *transaction_ID* `)`
  このマクロ関数でトランザクションを終了する．TXBEGIN で使用したトランザクション識別子を引数として取る．もし，TXBEGIN から TXEND の間の操作でコンフリクトが発生した場合，この間の共有データへの（TXREAD と TXWRITE を使用した）操作は無効となり，TXBEGIN の部分から操作が再試行される．
- `TXREAD(unsigned int` *transaction_ID*`,` *data type*`,` *var_name_to*`,` *var_name_from*`)`
  このマクロ関数でトランザクション内 (TXBEGIN から TXEND の間) での共有変数の参照を行う．第一引数に TXBEGIN で指定した unsigned int 型のトランザクション識別子を取り，第二引数に参照を行う共有データのデータ型を，第三引数に参照した値を格納する変数名を，第四引数に参照する共有データの変数名を取る．
- `TXWRITE( unsigned int` *transaction_ID*`,` *data type*`,` *var_name*`,` *value* `)`
  このマクロ関数でトランザクション内での共有変数の更新を行う．第一引数に TXBEGIN で指定した unsigned int 型のトランザクション識別子を，第二引数に更新を行う

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.32
Vol.2010-EMB-16 No.32
Vol.2010-MBL-53 No.32
Vol.2010-UBI-25 No.32
2010/3/27

共有データのデータ型を，第三引数に更新を行うデータを，第四引数に更新する値を引数として取る．

5. 設　　計

5.1 設計方針

本研究で設計した STM の設計手法の分類について示す．

コンフリクト発見法の分類は，オプティミスティックコンフリクト発見法にあたる．これを選択した理由としては，コンフリクト検査の回数が少なくて済むということと，順序依存のあるコンフリクトについて考えずに済むということが挙げられる．設計した STM では，コンフリクト検査をまとめて行うことで検査の高速化を実現する．

バージョン管理の分類は，ライトバッファ方式にあたる．これを理由としては，コミット時のオーバヘッドは大きくなるが，更新する内容を仮領域に書込むことで，優先度の低い処理を行うトランザクションによって優先度の高い処理を行うトランザクションが待たされずに済むということが挙げられる．

5.2 データ構造

本研究で設計したトランザクションメモリで使用しているデータ構造を示す．

・トランザクション構造体

始めにトランザクション構造体を示す．トランザクション構造体は，トランザクションが開始されたときに生成されるもので，各トランザクションが個別に一つ持つ．各トランザクションの状態を保持するためのものである．

トランザクション構造体は，メンバとして，トランザクション識別子，トランザクションが実行中なのかどうか（コードの状態が TXBEGIN と TXEND の間にあるか）を表すフラグ，トランザクション実行中に参照（読取り）した共有データの履歴（以下，Read セットと呼ぶ），トランザクション実行中に更新（書込み） した共有データの履歴（以下，Write セットと呼ぶ），トランザクション中の更新処理の内容を保持しておくため構造体（共有データのアドレス，仮領域のアドレス，共有データのサイズ），トランザクション開始時に既に実行中である他のトランザクションの ID のリスト，トランザクションが他のトランザクションの処理に関わっている数を表すカウンタがある．

・トランザクション構造体用ハッシュテーブル

トランザクション構造体用ハッシュテーブルは，各トランザクション構造体を一括して保管するハッシュテーブルである．トランザクション識別子をキーとし，キーで指定された識別子を持つトランザクション構造体を取り出す．このハッシュテーブルで使用するセルは，tk_tx_set_up でまとめて作成し，未使用リストに保持させる．セルを使用する場合に，この未使用リストからセルを取り出し，使い終われば未使用リストに返す．

・Read セット，Write セット

トランザクションメモリでは，複数のトランザクションが並行実行された場合，各トランザクションでの読み書きの相互作用によってコンフリクトを起こしたかどうかを検査する必要があり，そのために各トランザクションでの読み書きに対して履歴を取る必要がある．これは，トランザクション構造体のメンバである，Read セット，Write セットを使用して履歴を残す．

この Read セット，Write セットは本研究では unsigned int 型の変数で表す．この変数をビット列（本研究の環境では 32 個のビット列）として扱い，各ビットの列は各共有データと対応付くようにしている．初期値は 0 とする．ある共有データを読取りたい場合，Read セットの共有データに対応したビットを立て，書込みの場合は，同様に Write セットのビットを立てる．こうしてトランザクション内での共有データへの読み書きの履歴を取る．

読み書きの履歴にビット列を使用する理由は，トランザクションメモリの機能の中で最もオーバヘッドのかかると言われるコンフリクト検査を軽量化するためである．

・シフトビット用ハッシュテーブル

シフトビット用ハッシュテーブルとは，Read セット，Write セットのビット列と各共有データの対応付けを行うためのものである．共有データのアドレスをキーとし，それに対応する一意な整数を得る．ハッシュテーブルによって得られる整数は，Read セット，Write セットの最下位ビットから数えて何番目のビットを指すのかを表すものであり，共有データに対応付いた一意なものである．この整数を使用して，共有データの読取り，書込みがあった場合，Read セット，Write セットの共有データに対応したビットを立てる．

5.3 コンフリクト検査アルゴリズム

本研究のトランザクションメモリは，トランザクションの終了時点（TXEND が呼ばれた時点）で，並行実行していた他のトランザクションとの間でコンフリクトが起こっていないかを検査する．コンフリクトを起こさないことを保証するためのアルゴリズムを示す（文献 1)）．

コンフリクトが起きていないということを表す規則を次のように定義する．同時並行にト

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.32
Vol.2010-EMB-16 No.32
Vol.2010-MBL-53 No.32
Vol.2010-UBI-25 No.32
2010/3/27

ランザクションの実行が重なった場合を考え，先に終了処理を行うトランザクション（先行トランザクション）を T_i とし，後から終了処理を行うトランザクション（後行トランザクション） を T_j とする．その場合の規則を表 1 に示す．

表 1 コンフリクトを起さないための規則
Table 1 the rule of not causing conflict

T_i	T_j	規則
読取り	書込み	1．T_i は T_j が書込んだデータ項目を読取ってはならない．
書込み	読取り	2．T_j は T_i が書込んだデータ項目を読取ってはならない．
書込み	書込み	3．T_i は T_j が書込んだデータ項目に書き込んではならない．
		さらに T_j は T_i が書込んだデータ項目に書込んではならない．

次にコンフリクト検査の動作を示すために，トランザクションを開始してから正常終了（コミット） するまでの動作を三つに分ける．動作の段階は追跡段階，検査段階，確定段階の三つであり，この順で行われる．

追跡段階

追跡段階とは，トランザクション実行中に行われた読取り，書込みを追跡し，Read セット，Write セットに履歴を残していく段階のことである．TXBEGIN から TXEND 間の処理にあたる．

検査段階

検査段階とは，トランザクションの実行終了時に他のトランザクションに対してコンフリクトがあるのかを確認する段階のことである．コンフリクトの検査が成功すれば，トランザクションはコミットできる．失敗すれば，検査をしているトランザクションがアボートされる．

確定段階

確定段階とは，トランザクション内で行われた更新の内容をバッファ領域から実領域に反映させる段階のことである．検査段階が成功し，コミットしたときにこの段階に移行する．

ここで，**図 4** に示すように「検査段階と確定段階をただ一つのトランザクションだけが実行できる」ものとする．

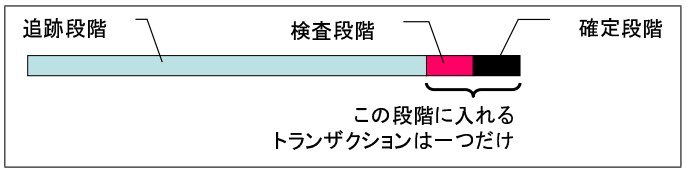

図 4 トランザクションの動作の状態
Fig. 4 operating state of the transaction

トランザクション内での更新内容を実際に書込むのは確定段階であるので，確定段階に入れるトランザクションをただ一つだけとしているこの場合，規則 3 は自動的に満たされる．また，後行トランザクションの更新内容が実際に反映されるのは，先行トランザクションが終了してからなので，先行トランザクションの全ての読取りは，後行トランザクションの書込みの前に行われ，規則 1 も自動的に満たされる．

トランザクションの検査は，規則 2 を満たすために行う．検査は，それを行うトランザクションが追跡段階にいる間に読取った値が，他の先行トランザクションによって書き換えられていないかを調べるものである．つまり，検査を行う後行トランザクションの Read セットの要素が，先行トランザクションの Write セットの要素と重複する要素を持たないかを検査する．検査の対象となるトランザクションは，追跡段階にいる間に読取った値を書き換える可能性があるものである．つまり，検査に入る後行トランザクションの開始時に，既に開始されて追跡段階にあり，かつ後行トランザクションが検査段階に入る時に，既に確定段階を終えているトランザクションが対象となる．例として，**図 5** に示すように，検査に入るトランザクションを T_j とし，先行トランザクションを T_1，T_2，T_3 とし，後行トランザクション *active* がある場合を考える． この図 5 の場合，T_j の検査の対象となるトランザクションは T_2 と T_3 である．

本研究のトランザクションメモリは，読取りと書込みの履歴をビット列として記録しているので，要素の重複を一つ一つ検査せず，アンド演算を行うことで，まとめて要素の重複を検査できるようにしている．

6. おわりに

本研究では，トランザクションメモリの機能を組込システム上で実現させることを目標として，今回は設計までを行った．現在，組込オペレーティングシステムの T-Kernel 上に今回設計した機能を実装中である．今後の課題として，組込オペレーティングシステム上への

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.32
Vol.2010-EMB-16 No.32
Vol.2010-MBL-53 No.32
Vol.2010-UBI-25 No.32
2010/3/27

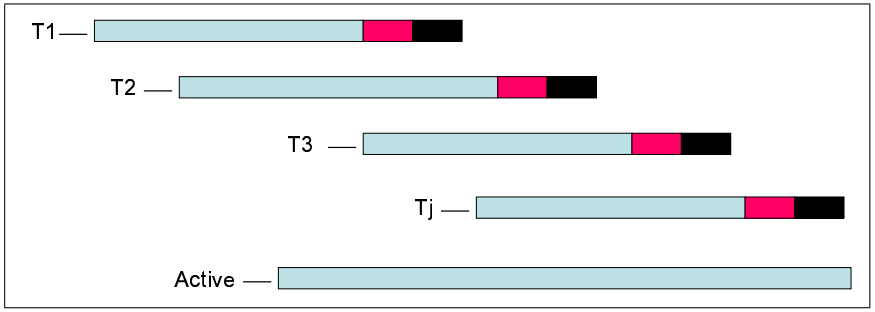

図 5 検査対象となるトランザクション
Fig. 5 transaction of subject to inspection

実装を完了させ，その機能の処理のオーバヘッドを測ることがある．また，本研究で述べた，コンフリクト発見法やバージョン管理の分類をそれぞれ実装して比較することも課題として挙げられる．

参 考 文 献

1) Dollimore, J., Kindberg, T. and Coulouris, G.: *Distributed Systems: Concepts and Design*, Addison Wesley, 4th edition (2005.
2) Gray, J. and Reuter, A.: *Transaction processing : concepts and techniques*, Morgan Kaufmann Publishers (1993).
3) Mankin, J., Kaeli, D. and Ardini, J.: Software transactional memory for multicore embedded systems, *SIGPLAN Notices*, Vol.44, No.7, pp.90–98 (2009).
4) Rossbach, C.J., Hofmann, O.S., Porter, D.E., Ramadan, H.E., Aditya, B. and Witchel, E.: TxLinux: using and managing hardware transactional memory in an operating system, *Proc. 21st ACM SIGOPS symposium on Operating systems principles*, pp.87–102 (2007).
5) Shavit, N. and Touitou, D.: Software transactional memory, *Proc. of the fourteenth annual ACM symposium on Principles of distributed computing*, pp.204–213 (1995).

© 2010 Information Processing Society of Japan