密結合マルチプロセッサにおける排他制御処理オーバヘッドの解析的一評価手法†

岩崎正明†† 高本良史†† 吉住誠一†††

本論文では，密結合マルチプロセッサ (TCMP) システムの排他制御処理に起因するダイナミック・ステップ数増加を定量解析する一手法を提案する．現在．メインフレーム系の TCMP システムは，オンライン・トランザクション処理 (OLTP) 分野で実用化が進んでいる．従来，OLTP システムの性能評価は，主に待ち行列理論を用いて行われてきており，共有資源の競合による待ち時間増加に比べれば，ダイナミック・ステップ数増加の影響は小さいと考えられてきた．しかし，プロセッサ台数が4台程度以上になると，排他制御処理に要するステップ数増加が無視できない場合がある．特に，オペレーティング・システムや OLTP システムの排他制御ルーチンの設計に際しては，ダイナミック・ステップ数増加の定量的な評価が必要となる．本論文では，ユニプロセッサ動作時の共有資源占有率から，任意のプロセッサ台数の TCMP における排他制御処理オーバヘッドを算出する手法を提案する．

1. まえがき

近年，大規模データベースをオンラインで検索・更新するトランザクション・サービスの進展に伴い，そのトラフィックは増加の一途を辿っている．このトラフィック増加に見合った性能を達成するために，現在オンライン・トランザクション処理 (OLTP) 分野で数多く実用に適されているシステムとして，単一のオペレーティング・システムの制御下で，主メモリを共有した複数のプロセッサが並行して動作する密結合マルチプロセッサ (TCMP) がある[1)~3)]．

TCMP システムは，プロセッサ台数に応じてリニアに性能が向上することが理想である．しかし，現実のシステムでは共有資源の競合などを始めとする種々の要因[4)~12)]により，効率の低下が問題となる場合が多い．

一方，OLTP システムの性能評価は，従来，主に待ち行列理論を用いて行われてきた[7)~9)]．待ち行列理論を拡張したマルチプロセッサ・システムのモデル化も試みられているが，これらの解析では，共有資源の排他制御処理オーバヘッド，すなわちロック競合に伴うダイナミック・ステップ数の増加は無視しうると仮定しているものが多い．

しかしながら，オペレーティング・システムや OLTP システムの排他制御ルーチンの設計に際しては，排他制御処理に伴うダイナミック・ステップ数増加の定量的な評価が必要となる．特にプロセッサ台数が4台程度以上になると，排他制御処理に要するステップ数増加が無視できなくなり，その評価手段が必要となる．

本論文は，ユニプロセッサ動作時の共有資源占有率から，任意のプロセッサ台数の TCMP における排他制御処理オーバヘッドを算出する手法を提案する．

2. 定性的分析

TCMP システムの性能がプロセッサ台数に比例してリニアに向上しない原因は大きく3つに分類される．第1は，共有資源の待ち時間増加によるプロセッサ稼働率の低下である[7),8)]．第2は，個々のプロセッサのハードウェア面での性能低下である[4),12)]．第3は，排他制御処理に伴うダイナミック・ステップ数の増加である．

本論文では，排他制御処理に伴うダイナミック・ステップ数増加について論じる．以下本章では，ダイナミック・ステップ数増加の原因を定性的に分析し，オーバヘッドの削減を図った2つの排他制御方式について述べる．

2.1 一般的な排他制御方式

現在，OLTP システムの主流を占めるメインフレーム系の TCMP システムの多くは，共有資源の排他制御を Wait・Post 方式[2),3)]により実現している．Wait・Post 方式は，共有資源がタスク1により占有中であった場合に，この共有資源の占有に失敗したタスク2を Wait 状態にする Wait 処理，および，タス

† An Analysis Method for Mutual Exclusion Overhead of Tightly-Coupled Multiprocessor by MASAAKI IWASAKI, YOSHIFUMI TAKAMOTO (8th Department, Central Research Laboratory, Hitachi, Ltd.) and SEIICHI YOSHIZUMI (3rd Department, System Development & Research Laboratory, Hitachi, Ltd.).

†† (株)日立製作所中央研究所第8部
††† (株)日立製作所システム開発研究所第3部

ク1が共有資源を解放する際にタスク2のWait状態を解除するPost処理によって，共有資源占有使用権のタスク間排他制御を行う*.

各タスクはロック取得に失敗**するとWait処理ルーチンへ制御を渡す．Wait処理ルーチンはロック取得に失敗したタスクをWait状態にし，ディスパッチャを呼び出す．ディスパッチャは，Wait状態となったタスク（Waitタスク）を実行していたプロセッサに，Ready状態の別タスクを割り当てる．

一方，各タスクは共有資源を解放する直前にWaitタスクの有無を調べ，Waitタスクがあればpost処理ルーチンへ制御を渡す．Post処理ルーチンはWaitタスクをReady状態に戻す．Ready状態に戻されたタスクは，ディスパッチャによってプロセッサを割り当てられ，共有資源の占有使用権を得て処理を再開する．

以下，Post処理を起動するタスクをPost発行タスク，Post処理によりWait状態を解除されるWaitタスクをPost受理タスクと呼ぶ．

2.2 排他制御方式の問題点

上述のWait・Post処理に伴うダイナミック・ステップ数増加には2つの要因がある．第1の要因は，OLTPシステムを構成するユーザ・アプリケーションやミドルソフトウェア（データベース管理システムなど）内の共有資源の競合により，Wait・Post処理回数が増加することである．第2の要因は，このWait・Post処理に伴うコンテクスト・スイッチ回数増加により，オペレーティング・システム内の資源であるタスク・キュー（ディスパッチング・キュー，プロセス・テーブルなどとも呼ばれる）のロック，いわゆるディスパッチャ・ロックの競合が増加することである．

TCMPシステムの場合，複数のプロセッサ上で動作しているタスクが，同時に共有資源へのアクセスを試みる．このため，プロセッサ台数増に伴って共有資源のビジー率が高くなる．トランザクション当たりの共有資源使用率 α を，

$$\alpha=\frac{\text{共有資源をロックして走行するステップ数}}{\text{トランザクション当たりの平均走行ステップ数}}$$

と定義すると，ユニプロセッサ時の共有資源のビジー率 β は，この共有資源使用率にほぼ等しい．これに対し，TCMPシステムの共有資源のビジー率 β は，共有資源使用率 α にプロセッサ数 P を乗じた値に近くなる*.

このためプロセッサ台数が増加するとロック競合に伴う排他制御処理オーバヘッドが無視できなくなる．特に，コンテクスト・スイッチ時，ディスパッチャはタスク制御情報等を集中的に管理するタスク・キューをロックしなければならない．このため上述のWait・Post処理に伴うコンテクスト・スイッチ回数増加は，複数のプロセッサ上でディスパッチャが同時に動作する機会を増加させ，ディスパッチャ・ロックの競合を誘発し，ダイナミック・ステップ数を増加させる．

2.3 オーバヘッド削減手法

上述の排他制御処理に伴うオーバヘッドを削減するには，共有資源の占有率を小さくしたり，Post処理やWait処理の単価（以下，1回の処理当たりの平均走行ステップ数を単価と呼ぶ）を小さくすることは当然であるが，TCMPシステムの場合にはこれ以外に次の3項目を考慮する必要がある．

（1） コンテクスト・スイッチ回数の削減
（2） 競合発生時の共有資源ビジー率上昇回避
（3） リトライ回数の削減

まず，上記(1)項のコンテクスト・スイッチ回数削減では，Post処理後のコンテクスト・スイッチ有無が問題となる．Wait処理を行った後，通常のオペレーティング・システムはディスパッチ処理を行い，プロセッサをアイドル状態にしないように，別タスクをプロセッサに割り当てる．これに対し，Post処理後の制御には2通りの方式が考えられる．ディスパッチ処理を行わず，Post発行タスクの実行を継続する方式と，ディスパッチ処理を行い，Post発行タスクの実行を中断し，別タスク（例えばPost受理タスク）にコンテクスト・スイッチする方式である．

マルチプロセッサの場合，Post発行タスクの実行を中断しなくても，他のプロセッサ上でディスパッチャが動作する契機（例えばI/O命令の発行によるタスク実行の中断）があれば，そのプロセッサでPost受理タスクをディスパッチすることができる．前節に述べたように，単一のタスク・キューを複数のプロセッサで共有するTCMPでは，プロセッサ数が増加

---

* 正確には，タスク2がタスク1にPostされるとは限らない．一般に，複数のタスクが同一の共有資源の解放を待つ場合，これらのタスクはキューイングされ，順番にPostされる．

** 以下，共有資源の占有使用権を得ることをロックを取る，ロック権を取得する等と表現する．また，ロックを取ろうとして，他のタスクが占有使用中であったためにロックが取れなかったことを，ロック取得失敗等と表現する．

* 厳密には，現実のシステムの場合，上式の分母に共有資源の待ち時間を含める必要がある．議論を進める上での前提条件や詳細な定量的解析は3章に述べる．

した場合，ディスパッチャ・ロックの競合がボトルネックとなる場合が多い．この点を考慮すると，Post 処理後はコンテクスト・スイッチを行わない方が好ましい．

次に，上記(2)項の競合発生時のロックビジー率上昇回避と(3)項のリトライ回数の削減について述べる．

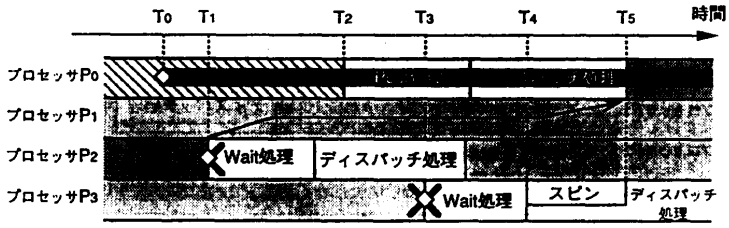

図 1 FIFO 方式
Fig. 1 Mutual exclusion by FirstIn-FirstOut.

ユニプロセッサの場合，Post 受理タスクが Post 処理後に最初にディスパッチされるタスクであれば，この Post 受理タスクは必ず共有資源を確保できる．すなわち，Post 処理後，直ちに Post 受理タスクへコンテクスト・スイッチすれば，Post 受理タスク以外のタスクが，Post 受理タスクより先に共有資源を使用するという状況は発生しない*．

これに対し，マルチプロセッサの場合，Post 発行タスクから Post 受理タスクへコンテクスト・スイッチしても，Post 受理タスクが共有資源を確保できるとは限らない．Post 受理タスクが，実際にプロセッサにディスパッチされて，Running 状態に遷移するまでの間（すなわち，Post されてから Running 状態になるまでの間）に，他のプロセッサ上で実行中の別タスク（仮にタスクXとする）から共有資源の使用要求が発行される．

タスク X から共有資源の使用要求が発行された場合，次の2通りの制御が考えられる．

(a) Post 受理タスクより先にタスク X に共有資源を使用させる
(b) タスクXを Wait させる

タスクXに共有資源を使用させる(a)の方式では，Post 受理タスクは共有資源の使用要求を再発行（リトライ）する必要がある．このリトライの後，Post 受理タスクは再び Wait する可能性があり，これはリトライ回数の増加を招く．

一方，タスク X を Wait させる(b)の方式では，Post 受理タスクは Post 発行タスクから共有資源の占有権を確実に受理でき，リトライ回数の増加は発生しない．しかし，Post 受理タスクが Post されてから Running 状態になるまでの間，タスクXが共有資源を使用できなくなる．Post 処理の発生時，すなわち，ロック競合の発生時に共有資源のビジー率が実効的に上昇することになる．これを低減する方法は，Post 発行タスクから Post 受理タスクにコンテクスト・スイッチさせ，タスクXが共有資源を使用できない時間を短縮することである*．

2.4 FIFO 方式と QuickRelease 方式

次章では，前述のオーバヘッド削減手法を考慮した2通りの排他制御方式，FIFO 方式と QuickRelease 方式について定量的に解析する．ここでは，上記(1)～(3)の項目と FIFO 方式，QuickRelease 方式の関係を説明する．

FIFO 方式**は，リトライ回数増加防止を優先する排他制御方式であり，前節に述べた(b)の方式に対応して，Post 処理後ただちに Post 受理タスクにコンテクスト・スイッチする．**図1**に FIFO 方式による排他制御動作の一例を示す．時刻 T_0 から T_2 までは，プロセッサ P_0 上のタスクがロックを取得している．時刻 T_1 にプロセッサ P_2 上のタスクがロック取得に失敗する．プロセッサ P_0 では，時刻 T_2 から Wait タスクへの Post 処理を開始し，時刻 T_5 にディスパッチ処理が完了して，Post 受理タスクが Running 状態となる．

FIFO 方式の利点は，Post 受理タスクが確実にロックを取得でき，リトライ回数の増加が発生しない点である．一方，欠点は，図1に示すように時刻 T_2 から T_5 の間，Post 受理タスク以外のロック取得が禁止されるため，プロセッサ P_3 上のタスクのように，ロック取得に失敗するケースが発生する点（同図の時刻 T_3）である．また，ディスパッチ処理回数の増加により，時刻 T_4 から T_5 のプロセッサ P_3 に見られるようなディスパッチャ・ロックの競合によるスピン・ステップ数増加が生じる．

* 厳密には，割込み処理の影響で Post 受理タスクより先に共有資源を使用する別のタスクがディスパッチされる可能性があるが，ここではその影響は無視する．

* このほかにも，他のプロセッサに割り込んで Post 受理タスクを優先的にディスパッチさせる方法も考えられる．しかし，割込み先のプロセッサの決定方法やプロセッサ間通信のオーバヘッドを検討すると実現上問題が多い．

** FIFO の名称は，Post 受理タスクが他のタスクに追い越されることなく，共有資源の使用要求の発行順に FirstIn/FirstOut で使用権が与えられることを示す．

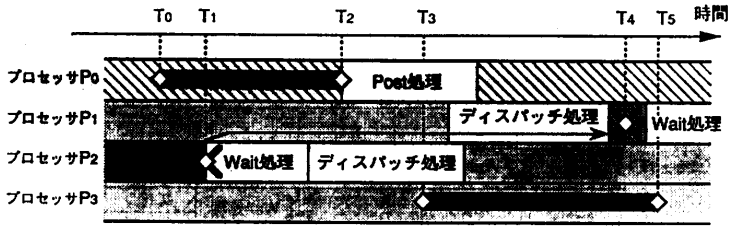

図 2 QuickRelease 方式
Fig. 2 Mutual exclusion by QuickRelease.

QuickRelease 方式は，コンテクスト・スイッチ回数の削減と競合発生時の共有資源ビジー率上昇回避を優先する排他制御方式であり，前節に述べた(a)の方式に対応する．**図2** に QuickRelease 方式による排他制御動作の一例を示す．時刻 T_0 から T_2 までは，プロセッサ P_0 上のタスクがロックを取得している．時刻 T_1 にプロセッサ P_2 上のタスクがロック取得に失敗する．プロセッサ P_0 では，Post 発行タスクがまず共有資源を解放（時刻 T_2）し*，Wait タスクへの Post 処理完了後は，Post 発行タスクの実行を継続する．Post 受理タスクが Running 状態となるまでの間，プロセッサ P_3 上のタスクのように Post 受理タスクを追い越してロックを取得できる（時刻 T_3）．

QuickRelease 方式の利点は，Post 処理後にディスパッチ処理を必要としない点，および Post 受理タスクを追い越してロックを取得できる**点である．一方，欠点は，ディスパッチされた Post 受理タスクがリトライを行った場合，図2に示す例のように，再びロック取得に失敗するケース（時刻 T_4）が発生する点である．

3. 定量的分析

ここでは，前節に述べた FIFO 方式と QuickRelease 方式について，排他制御処理によるダイナミック・ステップ数増加を定式化し，さらにその解法について述べる．

3.1 モデル化の前提条件

解析を行うために以下の前提条件を置く．

(a) システム全体で単一のタスク・キューを共有する．また，ディスパッチャ等のカーネル・ルーチンは，どのプロセッサ上でも実行できる対称型密結合マルチプロセッサとする．タスク・キューの排他制御はスピンロック方式[1]（ビジーウェイト等とも呼ばれる）によって行う．カーネル・ルーチン実行中は割込みが禁止される．

(b) 全プロセッサの稼働率が 100% となるに十分な Ready 状態のタスクが常時存在する．各タスクはどのプロセッサ上でも実行でき，プロセッサ間負荷バランスが質的にも均等で，共有資源の占有使用要求は各プロセッサから同じ割合で発行される．

(c) ハードウェア面の性能低下要因との関連は無視できる．すなわち，命令ごとの実行サイクル数の違い，あるいはバッファ一致制御やシリアライゼーション動作によるパイプラインの乱れに起因した命令実行サイクル数の増加等は影響しない．

(d) ロック占有タスクは優先的にディスパッチされる．タスクがロックを保持したままディスパッチされない現象は，その発生確率が低く無視できる．

(e) ロック占有中の割込み受付けは，その発生確率が低く無視できる．ロック占有中に割込みを受け付けると実質的にロック占有時間が伸びるが，この影響は無視できる．

実システムでは，これらの前提条件は必ずしも成立しない．特に上記(b)項のように Ready 状態タスクが常時存在し，プロセッサ・アイドルがないと仮定すると，実システムよりも共有資源の使用率が高くなる．実システムでは，共有資源の使用率が1に近づくと，多くのタスクがこの共有資源の空き待ちとなり，プロセッサ・アイドルが増加する．このため，排他制御処理に要するステップ数増加よりも，資源待ち時間増加の影響が大きくなる．

以下の検討では，プロセッサ台数，共有資源使用率，排他制御処理単価などの要因とダイナミック・ステップ数との関係を定量的に把握することを目的とし，資源待ち時間増加の影響がないシステムを仮定する．すなわち，共有資源の使用率が高くなってもプロセッサ・アイドルが増加しないようにチューニングを施した密結合マルチプロセッサ・システムを前提とする．

---

* QuickRelease の名称は，共有資源の解放を優先することを表す．
** プロセッサ P_3 上のタスクのロック解放タイミング T_3 が T_4 より早ければ，Post 受理タスクのリトライは発生しない．T_3 が T_4 より早ければ，Post 受理タスクを追い越してロックを取得できることがメリットとなる．

3.2 解析式の導出

オーバヘッド増加のメカニズムは，ロック制御方式によって異なる．ここでは，前述の FIFO，Quick-Release の各方式について解析モデルを作成する．なお，以下ユニプロセッサを略して UP，マルチプロセッサを略して MP と記す．

まず，UP/MP ダイナミック・ステップ数比 E_P を次のように定義する．

$$E_P=\frac{T_1}{T_P} \qquad (1)$$

ここで，T_1 は UP 時のトランザクション当たりの平均走行ステップ数，T_P は MP 時のトランザクション当たりの平均走行ステップ数である．添字 P はプロセッサ台数を表す．リトライ処理オーバヘッド（リトライ処理による走行ステップ数の増加分）を δ_{LK}，ディスパッチ処理オーバヘッド（ディスパッチ処理による走行ステップ数の増加分）を δ_{OS} とすると，T_P は次式で表される．

$$T_P=T_1+\delta_{LK}+\delta_{OS} \qquad (2)$$

リトライ処理オーバヘッド δ_{LK} は，Post 処理，Wait 処理に要するステップ数をそれぞれ ε_{POST}，ε_{WAIT} とすると次式で表される．

$$\delta_{LK}=\Delta(\varepsilon_{POST}+\varepsilon_{WAIT}) \qquad (3)$$

ここで Δ はトランザクション当たりのロック取得失敗回数を表す．このロック取得失敗回数 Δ は，FIFO 方式と QuickRelease 方式とでそれぞれ以下のように異なる．

FIFO 方式の場合，図1に示すように，1回ロック取得に失敗したタスクが Post 受理後には必ずロックを取得できるように制御する．よって，ロック取得失敗率を β_{LK}，トランザクション当たりのロック取得回数を N_{LK} とすると，トランザクション当たりのロック取得失敗回数 Δ は次式で表される．

$$\Delta=\beta_{LK}N_{LK} \qquad (4a)$$

一方，QuickRelease 方式の場合，図2に示すように，Post 発行後，Post 受理タスクがディスパッチされるまでの間に Post 受理タスク以外のタスクが共有資源をロックできるように制御する．このため FIFO 方式と異なり，ロック取得に失敗したタスクが Post 受理後に必ずロック取得に成功するとは限らない．トランザクション当たり平均 N_{LK} 回ロックを取得する場合，1回目のトライで失敗するのが $\beta_{LK}N_{LK}$ 回，成功するのが $(1-\beta_{LK})N_{LK}$ 回となる．失敗した $\beta_{LK}N_{LK}$ 回については2回目のリトライを行うが，失敗するのが $\beta_{LK}\beta_{LK}N_{LK}$ 回，成功するのが $(1-\beta_{LK})\beta_{LK}N_{LK}$ 回となる．以下，同様に3回目のリトライ，4回目のリトライ…と繰り返すと，ロック取得失敗の累積回数は，次式のとおりとなる．

$$\beta_{LK}N_{LK}+\beta_{LK}^2N_{LK}+\beta_{LK}^3N_{LK}+\cdots$$
$$=N_{LK}\sum_{k=1}^{\infty}\beta_{LK}^k=\frac{\beta_{LK}}{1-\beta_{LK}}N_{LK} \qquad (5)$$

この累積回数が，すなわちトランザクション当たりのリトライ回数 Δ であるから，Δ は次式で表される．

$$\Delta=\frac{\beta_{LK}}{1-\beta_{LK}}N_{LK} \qquad (4b)$$

ロック取得失敗率 β_{LK} は，自プロセッサがロック取得要求を発行したとき，他のプロセッサがロックを占有している確率であるから，プロセッサ稼働率 100% を仮定しているので，次式で表される．

$$\beta_{LK}=(P-1)\alpha_{LK} \qquad (6)$$

ここで，P はプロセッサ台数を，α_{LK} はトランザクション当たりのロック使用率を表す．α_{LK} については本節の最後で定式化する．

次に，ディスパッチ処理オーバヘッド δ_{OS} は，ユニプロセッサ時のコンテクスト・スイッチ回数を N_{OS}，ロック競合によるコンテクスト・スイッチ回数の増加分を Γ，ディスパッチャ・ロックの取得失敗率を β_{OS}，ディスパッチ処理に要するステップ数を λ_{OS} とすると次式で表すことができる．

$$\delta_{OS}=\Gamma\lambda_{OS}+\frac{\beta_{OS}}{1-\beta_{OS}}(\Gamma+N_{OS})\lambda_{SPIN} \qquad (7)$$

この式の右辺の第1項は，コンテクスト・スイッチ回数の増加分と1回のディスパッチ処理ステップ数の積である．第2項はディスパッチャ・ロックの競合によるスピン・オーバヘッドを表す．

ここで，λ_{SPIN} は「1回のスピン処理に要するステップ数」である．「1回のスピン処理に要するステップ数」は下記スピンループにおける外側ループの走行ステップ数と定義する．

```
SpinLoop1   Load             R5,#Locked
SpinLoop2   Load             R4,Lockword
            Compare          R4,#Free
            BranchNotEqual   SpinLoop2
            Load             R4,#Free
            CompareAndSwap   R4,R5,Lockword
            BranchNotZero    SpinLoop1
```

このスピンループは，内側ループで共有資源の空きを待ち，外側ループで共有資源の確保を試みる．第3図は内側ループを○印で表し，外側ループの CompareAndSwap 命令によるロック取得失敗を×印で表

している．この外側ループの走行ステップ数を λ_{SPIN} と定義する．**図3**から明らかなように，外側ループの走行ステップ数は，最初の1回目を除いて λ_{OS} に等しい．厳密にはこの1回目の走行ステップ数はディスパッチャ・ロックのビジー率に依存するが，以下の議論では λ_{SPIN} は λ_{OS} に等しいと近似する．

この近似を行うとディスパッチ処理オーバヘッド δ_{OS} は次式で表せる．

$$\delta_{OS}=\Gamma\lambda_{OS}+\frac{\beta_{OS}}{1-\beta_{OS}}(\Gamma+N_{OS})\lambda_{OS}$$

$$=\frac{\lambda_{OS}(\Gamma+\beta_{OS}N_{OS})}{1-\beta_{OS}} \quad (7)$$

FIFO 方式の場合，Wait 処理，Post 処理後にそれぞれコンテクスト・スイッチを行うので，ロック競合によるコンテクスト・スイッチ回数の増加分 Γ は，式 (4a) のトランザクション当たりのリトライ回数 Δ を用いて次式で表せる．

$$\Gamma=2\Delta=2\beta_{LK}N_{LK} \quad (8a)$$

一方，QuickRelease 方式の場合，Post 処理後にコンテクスト・スイッチを行わないので，Γ は式 (4b) の Δ を用いて次式で表せる．

$$\Gamma=\Delta=\frac{\beta_{LK}}{1-\beta_{LK}}N_{LK} \quad (8b)$$

なお，式(7)に現れるディスパッチャ・ロックの取得失敗率 β_{OS} は，β_{LK} と同様に次式で表される．

$$\beta_{OS}=(P-1)\alpha_{OS} \quad (9)$$

最後にトランザクション当たりの共有資源の使用率 α_{LK} とディスパッチャ・ロックの使用率 α_{OS} を定式化する．プロセッサ稼働率 100% を仮定しているので，トランザクション当たりの共有資源使用率 α を，

$$\alpha=\frac{\text{共有資源をロックして走行するステップ数}}{\text{トランザクション当たりの平均走行ステップ数}}$$

と定義すると，FIFO 方式におけるトランザクション当たりの共有資源使用率 α_{LK} は次式で表せる．

$$\alpha_{LK}=\frac{L_{LK}N_{LK}+\beta_{LK}N_{LK}\left(\varepsilon_{POST}+\lambda_{OS}+\frac{\beta_{OS}}{1-\beta_{OS}}\lambda_{OS}\right)}{T_P}$$

すなわち，

$$\alpha_{LK}=\frac{L_{LK}N_{LK}+\beta_{LK}N_{LK}\left(\varepsilon_{POST}+\frac{\lambda_{OS}}{1-\beta_{OS}}\right)}{T_P} \quad (10a)$$

この式の分母 T_P は，式(2)に示した MP 時のトランザクション当たり平均走行ステップ数である．分子の第1項はトランザクション当たりのロック取得回数と1回のロック保持ステップ数の積である．分子の第2項は Post 受理タスクへコンテクスト・スイッチするまでの間，共有資源が占有状態になることを反映している．

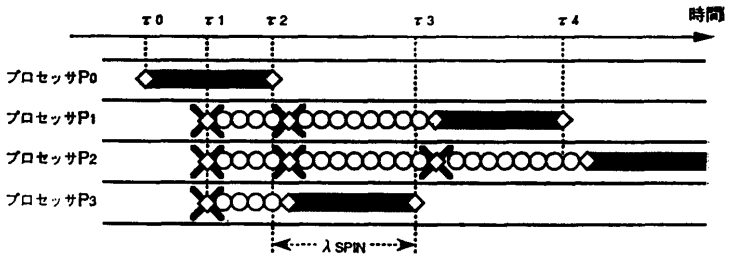

図3 スピンロック方式
Fig. 3 Spin overhead.

一方，QuickRelease 方式の場合は，Post 処理開始前に共有資源を解放するので，トランザクション当たりの共有資源の使用率 α_{LK} は次式で表せる．

$$\alpha_{LK}=\frac{L_{LK}N_{LK}}{T_P} \quad (10b)$$

ディスパッチャ・ロックの使用率 α_{OS} は，ユニプロセッサ時のディスパッチャ・ロック取得回数 N_{OS} とコンテクスト・スイッチ回数増加分 Γ の和が，MP 時トランザクション当たりのディスパッチャ・ロック取得回数となるので，次式のとおりとなる．

$$\alpha_{OS}=\frac{\lambda_{OS}(\Gamma+N_{OS})}{T_P} \quad (11)$$

3.3 解　　法

本節では3.2節で導いた一連の式の解法について説明する．式(2)から式(11)に現れるパラメータのうち，以下の8個は OLTP システムの設計値，あるいはユニプロセッサ上で OLTP システムを稼働させた場合の実測データを用いる．

P ：プロセッサ台数
T_1 ：UP 時のトランザクション当たり平均ステップ数
N_{LK} ：トランザクション当たりロック取得回数
L_{LK} ：1回のロック保持ステップ数
ε_{POST} ：1回の Post 処理ステップ数
ε_{WAIT} ：1回の Wait 処理ステップ数
N_{OS} ：UP 時のコンテクスト・スイッチ回数
λ_{OS} ：1回のコンテクスト・スイッチのステップ数

目的は，上記の8パラメータを用いて，下記の9変

数の値を求めることである.

T_P : MP時のトランザクション当たり平均ステップ数

δ_{LK} : トランザクション当たりリトライ処理オーバヘッド

δ_{OS} : トランザクション当たりディスパッチ処理オーバヘッド

Δ : トランザクション当たり Wait/Post 処理回数

β_{LK} : ロック取得失敗率

α_{LK} : トランザクション当たりのロック占有率

Γ : コンテクスト・スイッチ回数の増分

β_{OS} : ディスパッチャ・ロック取得失敗率

α_{OS} : トランザクション当たりのディスパッチャ・ロック占有率

QuickRelease 方式については, 上記9変数のうち, 1変数を残して消去することができる. 式(2)から式(11)を数式処理システム (例えば *Mathematica*[13]等) を用いて整理すると, T_P についての4次方程式が得られる.

T_P の値を算出するには, FIFO方式, QuickRelease 方式ともに,

$$0<P\cdot\alpha_{OS}<1$$

$$0<P\cdot\alpha_{LK}<1$$

を満たす α_{OS} と α_{LK} を初期値として, 2変数の差分法による反復計算で求めることができる.

4. 数値計算例

ここでは, 前章で導いた式を用いて算出した計算結果の例を2つ示す. 4.1 節の例は本解析手法と実測データとの比較を含む. 4.2 節の例は 10 台以上のプロセッサを有する対称型 TCMP の性能予測である.

4.1 実測データとの比較

図4にメインフレーム上のオンライン・システムを模したベンチマークの評価結果を示す. グラフの横軸はプロセッサ台数, 縦軸は UP/MP ダイナミック・ステップ数比 E_P とプロセッサ台数の積である.

ケース1のベンチマークは, FIFO 方式を用いており, UP 時のトランザクション当たりロック使用率 α_{LK} が約 12% に設定されている*. このベンチマークは, グラフに示すようにダイナミック・ステップ数が大幅に増加する. 計算ではプロセッサ稼働率 100% を仮定しているため, プロセッサ台数が4台の場合, 計算値の方が実測値よりステップ数増加が大きくなっている. 4台の場合の実測データでは, プロセッサ・アイドルが大きくなり, ダイナミック・ステップ数の増加は頭打ちとなっている.

ケース2のベンチマークは, QuickRelease 方式を用いており, UP 時のトランザクション当たりロック使用率 α_{LK} が約 5% に設定されている. このベンチマークについては, 実測値と計算値の差は約 5.5% である.

4.2 大規模 TCMP の場合

図5, 図6 にプロセッサ台数が 10 台以上の対称型 TCMP システムを想定した計算結果を示す. この計算結果から, QuickRelease/FIFO のいずれの方式でも, プロセッサ台数を増加した場合には, システムの分割運用など共有資源の競合を回避する工夫が必要となることが予想される.

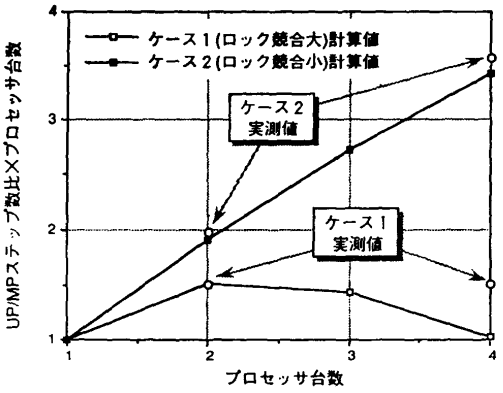

図 4 計算値と実測値の比較
Fig. 4 Comparison of calculated value and measured value.

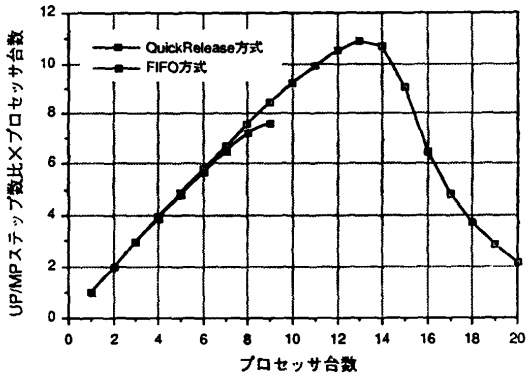

図 5 プロセッサ台数-UP/MP ステップ数比
Fig. 5 Processor number vs. performance curve.

* UP 時のトランザクション当たりロック使用率 α_{LK} は, 次式で表される.

$$\alpha_{LK}=N_{LK}L_{LK}/T_1$$

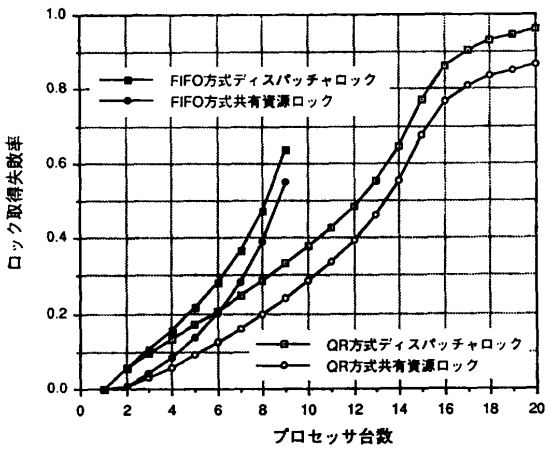

図 6 プロセッサ台数-ロック取得失敗率
Fig. 6 Processor number vs. lock fail rate curve.

なお，FIFO 方式の 10 台以上の部分は差分法による反復計算で解が収束しなかった．図6のグラフを見ると，ロック取得失敗率βが，プロセッサ数9台の場合で 70% に達している．このため，実際には 10 台以上のプロセッサで構成した TCMP システムは，プロセッサ稼働率が大幅に低下することが予想される．

5. むすび

密結合マルチプロセッサ・システムにおける排他制御処理オーバヘッドの定量的評価を目的として，共有資源の占有確率とダイナミック・ステップ数増加の関係を明らかにした．また，これを定式化して解析モデルを作成した．これにより，密結合マルチプロセッサ・システムにおけるダイナミック・ステップ数増加のメカニズムを把握することができ，オペレーティング・システムや OLTP システムの排他制御ルーチンの設計に際し，迅速な性能評価・予測が可能となった．

謝辞 本研究を推進するにあたりご協力をいただいた日立製作所ソフトウェア工場の皆様，日立ソフトウェア・エンジニアリングの皆様に感謝いたします．

参考文献

1) Hwang, K. and Briggs, F. A.: *Computer Architecture and Parallel Processing*, McGraw-Hill (1985).
2) Holley, L. H., Parmelee, R. P., Salisbury, C. A. and Saul, D. N.: VM/370 Asymmetric Multiprocessing, *IBM Syst. J.*, Vol. 18, No. 1, pp. 47-70 (1979).
3) Tetzlaff, W. H. and Buco, W. M.: VM/370 Attached Processor and Multiprocessor Performance Study, *IBM Syst. J.*, Vol. 23, No. 4, pp. 375-385 (1984).
4) Beck, B., Kasten, B. and Thakkar, S.: VLSI Assist for a Multiprocessor, *Proc. 2nd ASPLOS*, pp. 10-20 (1987).
5) Goodman, J. R.: Coherency for Multiprocessor Virtual Address Caches, *Proc. 2nd ASPLOS*, pp. 72-81 (1987).
6) Rashid, R., Tevanian, A., Young, M., Golub, D., Baron, R., Black, D., Bolosky, W. and Chew, J.: Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor, *Proc. 2nd ASPLOS*, pp. 31-39 (1987).
7) Nelson, R., Towsley, D. and Tantawi, A. N.: Performance Analysis of Parallel Processing Systems, *IEEE Trans. Softw. Eng.*, Vol. 14, No. 4, pp. 532-540 (1988).
8) Balbo, G. and Bruell, S. C.: Combining Queueing Networks and Generalized Stochastic PetriNets for the Solution of Complex Models of System Behavior, *IEEE Trans. Comput.*, Vol. 37, No. 10, pp. 1251-1268 (1988).
9) Heidelberger, P. and Lakshmi, M. S.: A Performance Comparison of Multimicro and Mainframe Database Architectures, *IEEE Trans. Softw. Eng.*, Vol. 14, No. 4, pp. 522-540 (1988).
10) チャニントーン，渡辺，中西，手塚：平均並列度を用いた平行処理システムの近似解析法，電子情報通信学会論文誌，Vol. J 71-D, No. 9, pp. 1623-1631 (1988).
11) 下條，宮原：分散処理システムにおけるプロセスの割当て問題に対する近似解法，電子情報通信学会論文誌，Vol. J 71-D, No. 9, pp. 1199-1206 (1988).
12) 長坂，黒川，栗山，和田：密結合マルチプロセッサ記憶階層性能評価手法，情報処理学会研究報告，Vol. 90, No. 7, 90-ARC-80 (1990).
13) Wolfram, S.: *Mathematica*, Addison-Wesley (1988).

付　　録

下のリストは T_P の算出に用いた2変数の差分法による反復計算プログラムを示す．$f(x, y)=0$ の解を求める場合，第1引数に関数 $f(x, y)$ を，第2引数に初期値 (x_0, y_0) を，第3引数に初期値 (x_1, y_1) を，第4引数に計算精度を，第5引数に計算の打ち切り回数を与えて関数 DIFF-2D を呼び出す．

```
(defun DIFF-2D (F V0 V1 E N)
  (let ((X0 (first  V0))
        (Y0 (second V0))
        (X1 (first  V1))
        (Y1 (second V1)))
    (do* ((I N (1- I)) (X2) (Y2))
         ((or (and (< I 0)
                   (CALC-ERROR
                     'cannot-solve-the-equation NIL))
              (and (< (abs (/ (- X1 X0) X0)) E)
                   (< (abs (/ (- Y1 Y0) Y0)) E)))
          (list X1 Y1))
      (progn (setf X2 (DIFF-2D-NEXT-X X0 X1 F Y0 Y1))
             (setf Y2 (DIFF-2D-NEXT-Y Y0 Y1 F X0 X1))
             (setf X0 X1 Y0 Y1)
             (setf X1 X2 Y1 Y2)))))

(defun DIFF-2D-NEXT-X (X0 X1 F Y0 Y1)
  (- X1
     (* (funcall F X1 Y1)
        (/ (- X1 X0)
           (- (funcall F X1 Y1) (funcall F X0 Y0))))))

(defun DIFF-2D-NEXT-Y (Y0 Y1 F X0 X1)
  (- Y1
     (* (funcall F X1 Y1)
        (/ (- Y1 Y0)
           (- (funcall F X1 Y1) (funcall F X0 Y0))))))

(defun TEST-FUN (X Y)
  "sample function for DIFF-2D."
  (+ (sin (* 2.0 X X X))
     (* 5.0 X Y Y)
     (cos (* -7.0 Y)) 3.0))
```

(平成元年5月15日受付)
(平成2年9月11日採録)

岩嵜　正明（正会員）

1958年生．1981年九州工業大学電子工学科卒業．1983年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．同年(株)日立製作所中央研究所入所．以来，並列推論マシン，OLTP システム，マルチプロセッサシステム等のシステム・ソフトウェアの研究に従事．電子情報通信学会，日本ソフトウェア科学会各会員．

高本　良史

昭和39年生．昭和57年愛媛県立松山工業高校情報技術科卒業．同年(株)日立製作所中央研究所入所．以来，データベースシステム，マルチプロセッサ，コンピュータアーキテクチャの研究に従事．

吉住　誠一（正会員）

昭和43年東京大学理学部数学科卒業．昭和46年同大学院修士課程修了．同年(株)日立製作所中央研究所入所．以来，OS，性能評価，コンピュータアーキテクチャの研究に従事．現在(株)日立製作所システム開発研究所第3部長．ソフトウェア科学会，IEEE，ACM 各会員．