情報処理学会論文誌　コンピューティングシステム　Vol. 4　No. 3　133–146 (May 2011)

中断可能な優先度継承キューイングスピンロックとそのハードウェア実装

一場利幸[†1]　松原豊[†2]
本田晋也[†2]　高田広章[†1,†2]

マルチコアプロセッサを用いたリアルタイムシステム向けの排他制御アルゴリズムとして，数多くのキューイングスピンロックアルゴリズムが提案されている．これらの多くは，単一のロックを取得する状況を想定しているため，複数のロックを同時に取得する場合には，コア数に対するスケーラビリティとリアルタイム性を両立させることができないという問題がある．本論文では，スピンロックが満たすべき要件を示し，この要件を満たす複数のロックを同時に取得する中断可能な優先度継承キューイングスピンロックアルゴリズムを提案する．さらに，提案アルゴリズムをハードウェアで実装することで高速に実行できる手法について述べる．ロック取得要求から解放までの時間と割込み応答性を実機で測定し，提案アルゴリズムが要件を満たすかどうか評価した．ハードウェア実装はソフトウェア実装に比べて，最大 3.11 倍高速化した．スピンロックというプリミティブな排他制御を 3 倍高速化できたことは，高い応答性が求められるシステムに対する有用性が高いと考えられる．

Preemptable Priority Inheritance Queueing Spin Lock and its Hardware Implementation

Toshiyuki Ichiba,[†1] Yutaka Matsubara,[†2]
Shinya Honda[†2] and Hiroaki Takada[†1,†2]

Many queueing spin lock algorithms for exclusive accesses on multicore real-time systems have been proposed. Most of the algorithms are only for a single spin lock. Therefore they can not satisfy both of real-time property and scalability for increasing the number of cores in the nested spin locks. In this paper, we propose a preemptive priority inheritance queueing spin locks algorithm for nested spin locks. Moreover the design of a queueing spin lock hardware for supporting the algorithm is described. As the evaluation of the algorithm, acquiring and releasing time of nested locks and interrupt latency are measured with a FPGA evaluation board. In the evaluation, the hardware implementation achieved 3.11 times speedup against the software implementation. It is useful in a system that requests a high response.

1. はじめに

近年，大規模化・複雑化が進む組み込みシステムにおいても，マルチコアシステムの重要性が増している．その背景には，消費電力の増大を抑えつつ処理性能の向上を図るためには，クロック周波数を上げるよりも，コア数を増やしたほうが有利であるという状況がある．特に，複数のコアを 1 つの LSI 上に集積したマルチコアプロセッサは，処理性能面からも消費電力面からも利点が大きく，広範な組み込みシステムへの適用が期待される．

組み込みシステムのソフトウェア開発においては，リアルタイム OS（以下，RTOS）が用いられることが多い．そのため，マルチコアを用いて組み込みシステムを構築する際には，マルチコア上でのシステム開発をサポートする RTOS（マルチコア向け RTOS）が必要になる．

マルチコア向け RTOS は，異なるコア上で動作するタスク間の通信や同期機構を提供するため，システムコール発行時に RTOS が管理する共有データ（以下，管理データ）を排他制御する必要がある[1)]．管理データの排他制御を実現するには，クリティカルセクションに関連づけられるオブジェクト（ロックと呼ぶ）を取得したコアのみが，排他区間の処理を実行できるようにする手法が用いられる．クリティカルセクションの出口ではロックを解放し，他のコアがクリティカルセクションの処理を実行できるようにする．一般に RTOS 内部のクリティカルセクションは短いため，ロックを取得できたかどうかを繰り返しチェックするスピンロックと呼ばれる方式が用いられる．すなわち，マルチコア向け RTOS がアプリケーションに提供する通信・同期機構は，RTOS 内部で用いるスピンロックを利用して実現されている[⋆1]．

組み込みシステムの多くは，最悪実行時間が予測できること，高速な割込み応答が可能であること，といったリアルタイム性を保証することが重要となる．そのため，RTOS にも

†1 名古屋大学大学院情報科学研究科
Graduate School of Information Science, Nagoya University
†2 名古屋大学大学院情報科学研究科附属組込みシステム研究センター
Center for Embedded Computing Systems, Nagoya University

⋆1 以降，特に記述のない場合，RTOS 内部で用いるスピンロックを，単にスピンロックと呼ぶ．

© 2011 Information Processing Society of Japan

同様のリアルタイム性が要求される．前述のように，マルチコア向けのRTOSでは，システムコールの実行時にRTOS内部でスピンロックを取得するため，スピンロック取得の最悪実行時間を抑えないと，システムコールの最悪実行時間が抑えられない．

マルチコア向けRTOSのリアルタイム性を実現するためのスピンロック方式に関して，様々な研究が行われており，単一のロックを取得する場合について，コア数に対するスケーラビリティとリアルタイム性を両立させた，中断可能なキューイングスピンロックと呼ばれるアルゴリズムが提案されている[2]．

我々がこれまでに開発したマルチコア向けRTOSであるTOPPERS/FDMPカーネル[1]（以下，FDMPカーネル）では，多くのシステムコールが2個のロックを同時に取得する必要がある[3]．しかしながら，2個ないしそれ以上のロックを取得する場合については，コア数に対するスケーラビリティを確保しつつリアルタイム性を満たすスピンロックアルゴリズムは提案されていない．

本論文では，RTOS内部で2つのロックを取得する場合について，コア数に対するスケーラビリティとリアルタイム性を両立させた，中断可能な優先度継承キューイングスピンロックアルゴリズムを提案する．提案手法は，エンジン制御などの最悪実行時間の予測性や高速な割込み応答性が求められるハードリアルタイムシステムに有効な手法である．アルゴリズムの一部をハードウェア化することにより，アルゴリズムの実現にCAS命令やLL/SC命令を必要としないため，様々なプロセッサで利用可能である．

本論文の構成は次のとおりである．2章では，まず，スピンロックが満たすべき要件を示す．これまでに提案されているアルゴリズムをあげ，すべての要件を満たすことができないことを述べる．3章では，すべての要件を満たすためのアルゴリズムについて述べる．4章では，提案アルゴリズムの実装方法について述べる．5章では，実装したアルゴリズムについて，すべての要件を満たしているかを評価する．6章では，関連研究について述べ，7章でまとめと今後の課題について述べる．

2. RTOS向けスピンロック

2.1 要　　件

リアルタイムシステムにおいては，ある処理の実行時間や応答時間に上限があることが求められる．マルチコアプロセッサシステムにおいては，さらに，コア数に対するスケーラビリティを確保することが重要である．スピンロックについては，ロック取得・解放処理の時間に上限があり，かつ，その上限がスケーラビリティを持つことが求められる．文献3)では，マルチコア対応RTOSが満たすべき要件をあげており，これを基に，2つの性質を両立させるために，スピンロックが満たすべき要件を以下のように定めた．

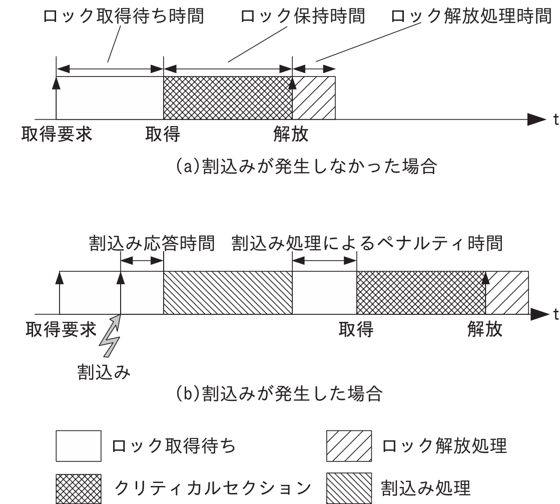

図1　3つの要件に対応する時間
Fig. 1　Times that correspond to three requreiments.

(1)　ロック取得待ち時間とロック解放処理時間の最悪実行時間がコア数のリニアオーダで抑えられる．

(2)　クリティカルセクション以外では，割込みを受け付けることができ，割込み応答時間がコア数に依存しない．

(3)　割込み処理によるペナルティ時間がコア数に依存しない．

(1)に関して，まず，ロックがすでにあるコアに保持されている場合は，解放されるまで他のコアはロックを取得することができない．ロック取得待ち時間は，ロック取得で競合するコア数によって変動するのは避けられないが，その最悪実行時間が定まることがリアルタイム性を満たすために必要である．さらにコア数に対するスケーラビリティを確保するためには，ロック取得待ち時間の変動をコア数のリニアオーダで抑える必要がある．また，アルゴリズムによってはロック取得待ち時間だけでなくロック解放処理時間も競合するコア数によって変動するので，取得と解放両方の最悪実行時間がコア数のリニアオーダで抑えられることが求められる．**図1**(a)に，ロック取得待ち時間とロック解放処理時間を示す．

© 2011 Information Processing Society of Japan

(2) に関して，多くの場合，ロック保持時間に比べると割込み処理の時間は長いため，ロック保持中の処理であるクリティカルセクションは割込み禁止にする．クリティカルセクションに加えて，ロック取得待ちの間も割込みを禁止すると，競合するコアが多い場合にロック取得待ち時間とともに割込み禁止時間も長くなり，割込み応答性が低下する．ここで割込み応答とは，割込みが発生してから割込み処理が実行されるまでの時間で，図 1 (b) で割込み応答時間とした部分である．割込み処理は高い応答性が求められるため，ロック取得待ちの間は割込み処理を受け付けることが望ましい．また，割込み応答時間がコア数に依存するとシステムのスケーラビリティが著しく損なわれてしまう．

(3) について，まず，割込みを受け付けることによりロック取得待ち時間が増える．割込みによって増加するのは，割込み処理に遷移してから復帰するまでの処理時間だけではない．割込みを受け付けることでロックを取得できる順番が変化し，割込み処理から復帰した後のロック取得するまでの待ち時間が増加する場合がある．この増加時間を，割込み処理によるペナルティ時間と呼ぶことにする．図 1 (b) で，割込み処理によるペナルティ時間とした部分が割込み処理から復帰した後に増えた待ち時間である．割込み処理によるペナルティ時間がコア数に依存すると，割込みが発生した場合のロックの取得待ちと解放処理の最悪実行時間がコア数のリニアオーダに抑えることができない．したがって，割込み処理によるペナルティ時間がコア数に依存しないことが求められる．

2.2　シングルロックのロックアルゴリズム

本論文では，単一のロックを取得することをシングルロックと呼ぶ．シングルロックの最も単純なスピンロックとして，TAS スピンロックがある．これは，ロック変数のチェックと書き込みをアトミックに行える命令を利用し，ロックが取得できるまでチェックを繰り返すというものである．このアルゴリズムでは，ロックを取得できる順番はランダムに決まるため，要件 (1) を満たせない．要件 (2) を満たすように実装するのは容易である[1)]．TAS スピンロックでは，ロック取得順がランダムなため，割込み処理によるペナルティ時間はなく，要件 (3) を満たす．

TAS スピンロックのようにロック取得の順番がランダムに定まる場合は，ロック取得待ち時間の上限を抑えることができない．そのため，ロックの取得順をキューで管理するキューイングスピンロックアルゴリズムが提案されている．単純なキューイングスピンロックアルゴリズムである MCS ロック[4)] は，ロック取得待ちの間に割込み処理を受け付けることができないため，要件 (2) を満たせない．

ロック取得待ち中に割込み処理を受け付けることができるキューイングスピンロックアルゴリズムとして，2 種類の中断可能なキューイングスピンロックアルゴリズムが提案されている[2)]．なお，要件 (3) については，アルゴリズムによって異なる．文献 2) のアルゴリズム 1 は，割込み処理から復帰後にキューの最後尾に並ぶため，ロック取得までの待ち時間がコア数に依存する．したがって，要件 (3) を満たさない．もう一方のアルゴリズム 2 は割込み処理から復帰後もキューの位置は変わらないため要件 (3) を満たす．

2.3　ネストロックの必要性

複数のロックをネストして取得することをネストロックと呼ぶ．

マルチコアシステムにおいては，1 つのロックで排他制御を行う共有資源の単位（ロック単位と呼ぶ）を適切に設定する必要がある[1)]．異なるロック単位に属する共有資源を同時にアクセスする場合には，対応するロックをすべて取得しなければならない．

RTOS 内部で用いるロック単位について，すべての管理データをまとめて 1 つのロック（ジャイアントロック）にしてしまうと，コア内に閉じた処理であってもロックの競合が発生し，他のコアの処理を阻害してしまう．このように，ジャイアントロックでは，マルチコアの利点をいかせないため，最低でもコアごとに別々のロックを設定する必要がある．

我々が開発している FDMP カーネルでは，プロセッサごとに 2 種類のロックを用意し，システムコールでは，最大 2 つのロックを同時に取得できれば，コア数に対するスケーラビリティとリアルタイム性を両立可能である[1),3)]．そのため，本論文では，2 つのロックを取得するネストロックのアルゴリズムを対象とする．

2.4　ネストロックのロックアルゴリズム

コア数を N とすると，シングルロックにおいてロック取得までの時間が $O(N)$ のアルゴリズムであっても，2 つのロックをネストして取得すると，ロック取得までの時間が $O(N^2)$ となってしまう．この問題について，$O(N)$ の Totally FIFO アプローチが提案されており[5)]，次の手順をとっている．1 段目のロック取得前にタイムスタンプを取得しておき，これを 1 段目と 2 段目のロック取得時の優先度とする．2 段目のロックは優先度順キューイングスピンロックであるが，タイムスタンプを優先度としているため，全体として FIFO 順となる．よって，Totally FIFO アプローチは要件 (1) を満たす．しかし，割込みについて言及がなく，そのままでは割込みを受け付けることができないため，要件 (2) を満たさない．

2.5　優先度逆転

Totally FIFO アプローチを割込みが受け付けられるように拡張するのは容易であるが，そのように拡張したアルゴリズムでは，優先度逆転が発生してしまい，割込み処理によるペナルティ時間がコア数に依存してしまうため，要件 (3) を満たさない．

© 2011 Information Processing Society of Japan

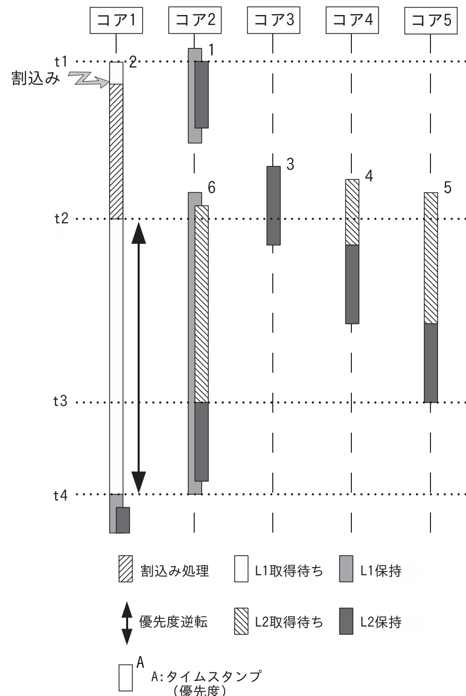

図 2　割込みを受付け可能な Totally FIFO アプローチで発生する優先度逆転の例
Fig. 2　An example of priority inversion by An enhanced Totally FIFO approach.

優先度逆転の例を**図 2** に示す．ロック L1 と L2 があり，割込み処理を受け付けるよう拡張した Totally FIFO アプローチでコア 1 とコア 2 が L1，L2 の順にネストロックを取得し，コア 3，コア 4，コア 5 が L2 のシングルロックを取得する場合を考える．ここでは，優先度として用いるタイムスタンプは 1 から始まり，数が小さいほどロック取得の優先度が高いものとする．t1 においてコア 1 が優先度 2 で L1 を取得しようとしたときには，コア 2 がすでに優先度 1 で L1 を保持している．そのため，コア 1 は L1 の取得待ちになる．その後，コア 1 に割込みが発生し，コア 1 は割込み処理を実行する．コア 1 が割込み処理を実行している間に，コア 2 は L2，L1 を解放する．t2 でコア 1 が割込み処理から復帰する前に，L2 について，コア 3 が優先度 3 で，コア 4 が優先度 4 で，コア 5 が優先度 5 で取得要求を出す．次に，コア 2 が優先度 6 で L1，L2 を順に取得しようとすると，コア 2 は L2 を取得する際にコア 3，コア 4，コア 5 に待たされる．そして，t2 でコア 1 が割込み処理から復帰して，L1 を再び取得しようとする．Totally FIFO アプローチでは，コア 1 のタイムスタンプが最も古いため他のコアより高い優先度を持つが，コア 2 が L1 を保持しているので，コア 1 は他の 4 つのコアに待たされてしまう．このように，高い優先度のコアが低い優先度のコアに待たされることを優先度逆転と呼ぶ．さらに，コア数が増え，そのコアがコア 2 より先に L2 を取得すると，優先度逆転の区間が長くなる．コア 1 が割込み処理から復帰したときに発生する優先度逆転の区間はコア数に比例して長くなるため，要件 (3) を満たすことができない．

3. PPIQL アルゴリズム

Totally FIFO アプローチをベースに，2 段のネストロックについて 3 つの要件を満たすよう拡張した PPIQL（Preemptive Priority Inheritance Queueing spin Lock）アルゴリズムを提案する．

3.1 アルゴリズム

一般に，優先度逆転を回避するための手法として優先度継承プロトコルと優先度上限プロトコルが知られている[6]．一般には優先度上限プロトコルの方が安全であるとされているが，次に述べる理由により，優先度継承プロトコルを用いることとした．

優先度上限プロトコルでは，あらかじめ上限となる優先度（上限優先度）を求める必要があるが，PPIQL アルゴリズムでは，タスクに固有の優先度が与えられないため上限優先度が定まらない．これは，ロック取得要求の順番でロックを取得するために，ロック取得要求時のタイムスタンプを優先度として用いるためである．一方，あらかじめ上限優先度を求めずに，特別な値（0 など）を上限優先度として用いる手法も考えられる．この手法では，たとえば，タスク T1 がロック L1，L2 の順に，タスク T2 がロック L3，L2 の順にロックを取得し，タスク T3 はロック L2 のみを取得する場合，タスク T3 は通常の優先度で L2 の取得を試行するのに対し，タスク T1 とタスク T2 は上限優先度で L2 を取得することになる．タスク T1 とタスク T2 は交互に L2 の取得を繰り返せるため，タスク T3 が L2 を取得する最悪実行時間が定まらない[5]．したがって，この手法では要件 (1) を満たせない．

PPIQL アルゴリズムは，Totally FIFO アプローチを割込み処理の受付け可能に拡張し，さらに優先度継承処理を行うものである．いい換えると，ロック取得待ちの間に割込みが入らない場合は，優先度継承処理を除いて Totally FIFO アプローチと同じである．アルゴリズムを**図 3** に示す．

© 2011 Information Processing Society of Japan

```
1:  ロック取得の優先度（タイムスタンプ）を取得する
2: acquireL1:
3:  1 段目のロック取得待ちキューに優先度に従いならぶ
4:   1 段目のロック取得待ちとなる
5:  1 段目のロック取得待ちの間，a. を繰り返す
6:    a. 割込みが発生した場合
7:      1 段目のロック取得待ちを解除し，待ちキューから外す
8:      割込み処理を実行する
9:      割込み処理から復帰したら acquireL1 に戻る
10:  1 段目のロックを取得する
11:   1 段目のロック取得待ちを解除し，待ちキューから外す
12:
13:  1 段目のロックによるクリティカルセクションの実行
14:
15:  2 段目のロック取得待ちキューに優先度に従いならぶ
16:   2 段目のロック取得待ちとなる
17:  2 段目のロック取得待ちの間，b. と c. を繰り返す
18:    b. 優先度継承処理
19:      1 段目のロック取得待ちの最高優先度を探す
20:      ロック取得の優先度を，探索した優先度に設定する
21:    c. 割込みが発生した場合
22:      1 段目のロックを解放する
23:      2 段目のロック取得待ちを解除し，待ちキューから外す
24:      割込み処理を実行する
25:      割込み処理から復帰したら acquireL1 に戻る
26:  2 段目のロックを取得する
27:   2 段目のロック取得待ちを解除し，待ちキューから外す
28:
29:  1 段目，2 段目のロックによるクリティカルセクションを実行
30:
31:  2 段目のロックを解放する
32:  1 段目のロックを解放する
```

図 3　PPIQL アルゴリズム
Fig. 3　PPIQL algorithm.

1 段目，2 段目にかかわらず，ロック取得待ちの間に割込みが発生した場合は，ロック取得待ちを解除してから割込み処理を実行し，他のコアがロックを取得できるようにする（6, 21 行目）．割込み処理から復帰した後はロック取得処理をやり直す．1 段目のロックについては，ロック取得待ち状態であることを解除してロック取得処理をやり直すだけでよい．2 段目のロックの場合は，1 段目のロックを保持しているため，ロック取得状態の解除だけでなく，1 段目のロックも解放する必要がある（22 行目）．これは，2 段目のロック取得待ちの間に割込み処理を実行する場合，1 段目のロックを解放してから割込み処理を実行しないと，他のコアが 1 段目のロックを取得する最悪実行時間が長くなってしまうためである．

1 段目のロックを解放し，ロック取得をやり直す場合は，1 段目のロックにより排他されるクリティカルセクションの実行（13 行目）を複数回行うことになる．複数回実行するかどうかは予測が難しいため，13 行目で実行できる処理は繰り返し実行しても問題ない処理に限られる．そのため，繰り返し実行すると問題のある処理は 2 つのロックを取得してから実行する必要がある．FDMP カーネルでは，1 段目のロックにより排他されるクリティカルセクションは，2 段目のロックの判定処理や RTOS が管理するデータの状態による場合分けなど繰り返し実行しても問題ない処理のみで記述できている．アプリケーションが PPIQL アルゴリズムを利用する際には，1 段目のロックにより排他されるクリティカルセクションが繰り返し実行しても問題ないことが必要であり，そうでない場合は PPIQL アルゴリズムを用いることができない．

2 段目のロック取得については，優先度継承処理が必要となる．優先度継承処理は，1 段目のロックについて自コアの設定した優先度より高い優先度のコアがロック取得待ちになっていないかをチェックする．もし，より高い優先度のコアが待っている場合は，その優先度を継承し，2 段目のロック取得の優先度とする．これにより 2 段目のロックを取得する優先度が高くなり，優先度逆転を低減することができる．

3.2　優先度継承の例

優先度継承により，図 2 のシナリオは**図 4** のようになる．コア 2 が優先度 6 で L2 の取得を要求するとき，すでにコア 3 が L2 を保持しているため，コア 2 は L2 を取得できない．コア 2 は再度 L2 の取得を試みる前に，コア 2 より高い優先度のコアが L1 の取得待ちになっていないかチェックを行う．この優先度のチェックは，L2 のロック取得に失敗し，再度取得を試みる前に毎回行う．t2 より後にチェックを行うと，コア 2 より優先度の高いコア 1 が取得待ちをしているため，コア 2 はコア 1 の優先度 2 を継承し，L2 を取得する優先度として設定する．t3 でコア 3 がロック L2 を解放すると，コア 2 はコア 4 とコア 5 より優先度が高いため L2 を取得する．t4 でコア 2 が L1 を解放するとコア 1 が L1 を取得する．t4 以降，コア 1 が t5 まで L2 を取得できないのは避けられない．このように，優先度継承を導入することによってコア 1 が待たされる低優先度のコアを 3 つに抑えることできる．

3.3　ソフトウェア実装

優先度順キューイングスピンロックアルゴリズムに優先度継承を導入したソフトウェア

© 2011 Information Processing Society of Japan

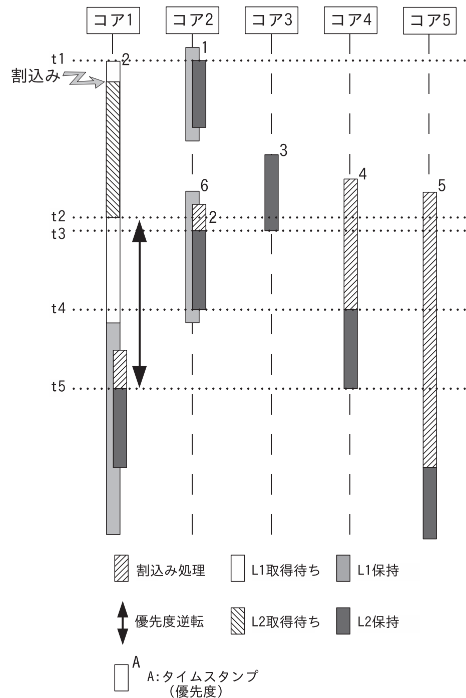

図 4　PPIQL の優先度継承の例
Fig. 4　An example of PPIQL.

ルゴリズムがすでに提案されている[7)]．しかし，このアルゴリズムは割込み処理を受け付けることができないため，割込み処理を受け付けるよう拡張すれば，PPIQL のソフトウェア実装が可能である．割込み処理を受け付ける拡張については，シングルロックでコア数に対するスケーラビリティとリアルタイム性を両立させた，中断可能なキューイングスピンロックアルゴリズム[2)] が参考になる．

PPIQL をソフトウェア実装するためには，CAS 命令や LL/SC 命令が必要であり，TAS 命令では実装できない．LL/SC 命令を用いて PPIQL をソフトウェア実装した結果，2 段目のロック取得についてデータ定義などを除き 170 行程度の規模となった．文献 2) で提案されている 2 つのアルゴリズムが 50 行から 80 行程度の規模であるため，ロック取得待ち時間とロック解放処理時間が増加すると考えられる．PPIQL のソフトウェア実装では，コードサイズの増加による処理時間の増加だけでなく，アルゴリズムの複雑化による処理時間の増加も懸念される．PPIQL ではロック取得順は優先度順でなければならず，そのためにはキューのノードを走査する必要がある．また，優先度継承処理でもノードの走査が必要である．これらの処理には，コア数に依存した時間がかかるため，PPIQL アルゴリズムのソフトウェア実装では，ロック取得待ち時間とロック解放処理時間が長くなると予想される．

4. PPIQL アルゴリズムのハードウェア実装

本論文ではハードウェアを用いた PPIQL アルゴリズムの実現方法を提案する．

本章では，まず，ハードウェア実装の利点と欠点について述べた後，PPIQL アルゴリズムを実現するためのハードウェアについて述べ，これを利用したロック取得と解放処理のドライバについて述べる．

4.1　ハードウェア実装

組み込み向けプロセッサには，CAS 命令や LL/SC 命令を持たず，TAS 命令のみを持つものも少なからず存在している．このようなプロセッサで PPIQL を用いるためには，CAS 命令や LL/SC 命令を用意するか，提案ハードウェアを用いる必要がある．ここで，CAS 命令や LL/SC 命令を実現するためにはバスを複雑化する必要があるため，提案ハードウェアを用いることに有用性があると考えられる．また，専用ハードウェアにより排他制御を実現しているプロセッサにおいても，専用ハードウェアは TAS 命令相当の機能であるため（TAS ハードウェアと呼ぶ），PPIQL のソフトウェア実装を実現することができない．TAS ハードウェアを用意しているプロセッサにおいては，PPIQL のハードウェア実装を用いることで，より優れた排他制御手法を用いることができる．

TAS ハードウェアを用いている例として，SH7265（ルネサスエレクトロニクス社）では 32 個持つ．また，MC9S12XD（フリースケール社）ではメインの S12X とコプロセッサの XGATE 間のために 8 個，OMAP4430（テキサスインスツルメンツ社）では Cortex-A9, Cortex-M3，DSP 間のために 32 個の TAS ハードウェアを持つ．このように，ヘテロジニアスなマルチコアシステムでも TAS ハードウェアを用いていることが多い．

PPIQL をハードウェア実装し，アプリケーションから提案ハードウェアを直接用いたい場合には，アプリケーションからハードウェアをアクセス可能とするため，カーネル空間で実行する必要がある．一般的に RTOS は保護機能を持たず，アプリケーションをカーネル空間で実行するため，問題とならない．

システムで必要となるロックの数によって，必要となるハードウェアの数も変化する．

© 2011 Information Processing Society of Japan

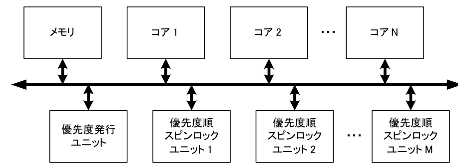

図 5　提案ハードウェアを利用したシステムの例
Fig. 5　An example system with proposed hardware.

RTOS 内部で用いる場合には，RTOS 設計時に必要なロックの個数が決まるため，この数を用意すればよい．前述のように，我々が開発している FDMP カーネルでは，コア数 × 2 個用意すればよい．

一方，アプリケーションが提案ハードウェアを用いる場合には，アプリケーションによって必要となるロック数が変化するため，ハードウェアが提供すべきロックの数を予測しにくい．そのため，アプリケーションが提案ハードウェアを利用するシステムを構築する際には，十分な数のハードウェアを用意しておく必要がある．

4.2　ハードウェアの概要

提案ハードウェアは，優先度発行ユニットと優先度順スピンロックユニットで構成される．優先度発行ユニットはシステムに 1 つ必要で，優先度順スピンロックユニットはロック個数と同数だけ必要である．**図 5** に提案ハードウェアを用いてロックを実現するシステムの例を示す．2 種類のユニットを用いてロックを取得する手順は次のようになる．ロックを取得するコアは，優先度発行ユニットから優先度を取得し，優先度順スピンロックユニットに設定する．この優先度はロックを取得する順序を決めるもので，優先度順スピンロックユニットは設定された優先度の中で最も優先度の高いコアにロックを与える．

4.2.1　優　先　度

優先度は，優先度発行ユニットと優先度順スピンロックユニットの両方で利用する値である．優先度発行ユニットから読み出した優先度を優先度順スピンロックユニットに設定する．優先度には，有効値と無効値があり，優先度発行ユニットからは有効値のみを読み出す．優先度順スピンロックユニットには，有効値もしくは無効値を設定することができる．無効値が設定されているコアはロック取得を要求していないことを表す．有効値は，値の小さいものほど高い優先度を持つことを表す．

4.2.2　優先度発行ユニット

優先度発行ユニットは，ロックを取得しようとしているコアに対し，ロック取得の優先度を与える．優先度はユニット内部のカウンタにより保持されており，値を読み出すたびにインクリメントされる．値が小さいほど高い優先度のため，値を先に読み出したコアが高い優先度を持つ．優先度発行ユニットにより，FIFO 順にロックを取得するという原則が与えられる．

複数のコアが同時に優先度の読み出しを行った際は，バスによる調停が行われる．読み出しは単一命令で実現できることが多いため，この場合は複数のコアはそれぞれ異なる値を読み出すことが保証される．バスの調停方式は複数あるが，他のコアにバスを占有され，バスを使用できないコアが発生する可能性のある方式では，要件 (1) を満たせないため，ラウンドロビンなどの方式を用いる必要がある．

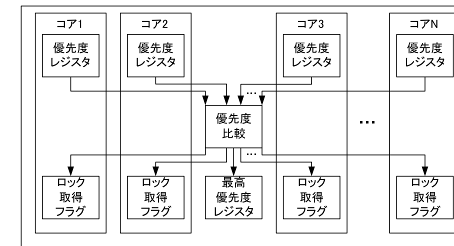

図 6　優先度順スピンロックユニット
Fig. 6　Priority ordering spin lock unit.

4.2.3　優先度順スピンロックユニット

優先度順スピンロックユニットは，ロック取得可能なコアを決定するユニットである．内部には，最高優先度レジスタと，コアごとに優先度レジスタとロック取得フラグの組を持つ．優先度順スピンロックユニットを**図 6** に示す．

優先度順スピンロックユニットの状態遷移を**図 7** に示す．状態はアンロック状態とロック状態の 2 状態を持ち，リセット時にはアンロック状態となる．アンロック状態のときは，有効値を保持している優先度レジスタの値を比較し，最も高い優先度を持つ優先度レジスタに対応するロック取得フラグをセットし，ロック状態に遷移する．すべての優先度レジスタの値が無効値である場合はアンロック状態のまま遷移しない．ロック状態のときは，セットされているロック取得フラグに対応する優先度レジスタに無効値が書き込まれると，ロック取得フラグをクリアして，アンロック状態に遷移する．最高優先度レジスタには，状態に関係なく，有効値の中で最高の優先度が設定される．

優先度の比較について，優先度がオーバフローすると単純な大小比較ではうまくいかな

© 2011 Information Processing Society of Japan

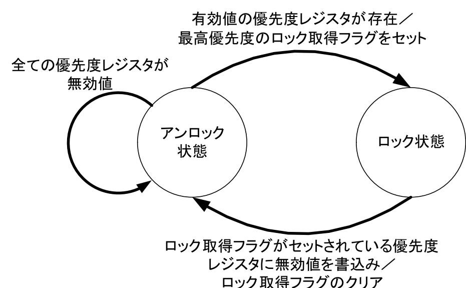

図 7　優先度順スピンロックユニットの状態遷移図
Fig. 7　State transition diagram of the priority ordering spin lock unit.

い．そのため，比較には ICTOH [8] と同じアルゴリズムを用いている．

ロックを取得したいコアは，まず，優先度の有効値を優先度レジスタに設定する．ロック取得フラグをチェックし，セットされていれば，ロック取得に成功する．ロック取得フラグがセットされていないときは，ロック取得に成功するまでロック取得フラグをチェックする．クリティカルセクション実行後は，優先度レジスタに無効値を書き込むことでロック解放を行う．

優先度比較器は，1 つの比較器を用いてコア数分の優先度を比較するのではなく，複数の比較器を組み合わせて実現している．コア数が多くなったときは，比較器の段数が深くなるため，結果を得るためのサイクル数が増えるか，動作クロックを遅くする必要がある．そのため，コア数が多くなるとスピンロック取得待ち時間が長くなる．

4.3 ドライバ

以下に述べるドライバの処理は，アプリケーションが直接利用することも可能であり，カーネル空間で実行されることを想定している．

4.3.1 ロック取得・解放

提案ハードウェアを利用したロック取得・解放のルーチンを**図 8** に示す．図 8 では，すべての処理をまとめて記述している．19，20 行目は 1 段目のロックによるクリティカルセクション（CS_1 とする）で，35 行目が 1 段目と 2 段目のロックによるクリティカルセクション（CS_12 とする）である．`coreid` はコアの識別子を，`lockid1` と `lockid2` は優先度順スピンロックユニットの識別子をそれぞれ表す．`lockid1` は 1 段目のロックに対応し，`lockid2` は 2 段目のロックに対応する．`pri` 配列は各コアがロックを取得する優先度を保持する．優先度には無効値（`INVALID`，たとえば 0）と有効値があり，無効値はそのコアが

```
 1:  /* 割込み禁止 */
 2:  disable_interrupt();
 3: retry:
 4:  /* 初回のみ優先度を取得する */
 5:  if( pri[coreid] == INVALID ) {
 6:    pri[coreid] = get_priority();
 7:  }
 8:  spinning[coreid] = true;
 9:  /* 1 段目のロック取得 */
10:  enter_spinlock(lockid1, coreid, pri[coreid]);
11:  while( try_spinlock(lockid1, coreid) ) {
12:    enable_interrupt();
13:    /* ここで割込み処理に移る */
14:    disable_interrupt();
15:    if (spinning[coreid] == false) {
16:     goto retry;
17:    }
18:  }
19:  /* ここから 1 段目のロックによるクリティカルセクション
20:     ここの処理は，割込みにより繰り返し実行される場合がある */
21:  /* 2 段目のロック取得 */
22:  enter_spinlock(lockid2, coreid, pri[coreid]);
23:  while( try_spinlock(lockid2, coreid) ) {
24:    enable_interrupt();
25:    /* ここで割込み処理に移る */
26:    disable_interrupt();
27:    if (spinning[coreid] == false) {
28:     goto retry;
29:    }
30:    high_pri = get_high_pri_on_spinlock(lockid1);
31:    if( high_pri < pri[coreid] ){
32:      enter_spinlock(lockid2, coreid, high_pri);
33:    }
34:  }
35:  /* 1 段目，2 段目のロックによるクリティカルセクション */
36:  /* ロック解放 */
37:  release_spinlock( lockid2, coreid );
38:  release_spinlock( lockid1, coreid );
39:  spinning[coreid] = false;
40:  pri[coreid] = INVALID;
41:  enable_interrupt();
```

図 8　ロック取得・解放ルーチン
Fig. 8　Routine of acquiring and releasing locks.

© 2011 Information Processing Society of Japan

```
1:  /* 無効値を設定する */
2:  suspend_spinlock( lockid2, coreid );
3:  suspend_spinlock( lockid1, coreid );
4:  spinning[coreid] = false;
```

図 9　割込み処理入口ルーチン
Fig. 9　Interrupt service routine.

ロック取得に関する処理を実行していないことを表す．spinning 配列は各コアが割込み処理を実行したかどうかを保持する．

ロック取得を要求したら，まず，優先度を取得する．pri 配列には無効値で初期化されているため，優先度発行ユニットから get_priority() 関数により優先度を読み出す（6 行目）．読み出した優先度は enter_spinlock() 関数により優先度レジスタに設定する（10 行目）．優先度順スピンロックユニットは，有効値の優先度を比較し，最高優先度のコアに対応するロック取得フラグをセットする．ロックが取得できたかどうかは，ロック取得フラグをチェックする try_spinlock() 関数で行う（11 行目）．try_spinlock() 関数はロック取得できない場合に真が返ってくる．偽が返ってきてロックが取得できていれば，クリティカルセクションの実行に移る．ロックを取得できず，かつ，割込みが発生した場合，割込み処理を行うことで割込み応答性を向上させる．割込み応答性に関しては後述する．

2 段目のロックについても同様の処理を行う．ただし，Totally FIFO アプローチと同様に 1 段目の優先度で 2 段目のロックを取得する．また，優先度継承するため，2 段目のロック取得待ち時に 1 段目のロックの最高優先度レジスタを get_high_pri_on_spinlock() 関数により読み出す（30 行目）．読み出した優先度がそのコアが設定した優先度より高い場合は，優先度を継承し 2 段目のロック取得を行う．

ロックの解放は，release_spinlock() 関数で優先度順スピンロックユニットに対し無効値を設定し（37，38 行目），各配列に無効値を設定する（39，40 行目）．

4.3.2　ロック取得待ちの中断

ロック取得待ち中の割込み応答性を向上させるために，ロックが取得できなかった場合は割込み処理を行う．具体的には，try_spinlock() 関数でロック取得できていないことが分かった場合，いったん，割込みを許可する（12，24 行目）．割込みが発生していた場合はここで割込み処理にジャンプする．割込み処理では，**図 9** に示す処理を実行した後，実際の割込み処理を行う．図 9 の処理では，suspend_spinlock() 関数により優先度レジスタに無効値が書き込まれるため，優先度順スピンロックユニットによる優先度比較の結果，割込み処理を実行するコアにロックが渡されることがない．割込み処理の実行後は，get_priority() 関数を呼ばずに，割込み処理前と同じ優先度を優先度レジスタに設定する．

5．評　　価

5.1　評価項目

実機を用いた実験により，提案するアルゴリズムの性能評価を行う．比較には，MCS ロックと優先度順キューイングスピンロックアルゴリズムである Markatos ロック[9] を用いた Totally FIFO アプローチ（TF），提案ハードウェアを用いたシングルロックを単純にネストロックにしたもの（simple），Totally FIFO を中断可能にしたもの（TF/P），PPIQL をソフトウェア実装したもの（PPIQL(SW)）を用いる．simple の基本となるシングルロックは中断可能なキューイングスピンロックであるため，割込み処理を受け付けることができる．TF/P は，PPIQL の優先度継承の処理がないものである．RTOS 内部で用いるプリミティブなロックについて，評価対象としたアルゴリズム以外に割込み応答とコア数のスケーラビリティについて言及している手法は，我々が知る限り存在しない．

それぞれのアルゴリズムについて，3 つの要件を満たすかという観点で評価を行う．要件 (3) については，優先度逆転が発生するケースについても評価を行う．また，提案ハードウェアのサイズの比較評価を行う．

5.2　評価環境

提案ハードウェアを VHDL で記述し，実装デバイスには StratixII EP2S180 を用い，Altera 社の QuartusII version 8.1 を用いて合成を行った．優先度は 16 ビットとした．コアには周波数 50 MHz の NiosII プロセッサを用いた．NiosII プロセッサの命令キャッシュは 4 KB で，データキャッシュはない．この構成の NiosII プロセッサを 8 個合成したもので評価を行う．優先度発行ユニットを接続するバスの調停はラウンドロビンで行われ，優先度比較は 8 コアでも 1 サイクルで終了することを確認した．また，PPIQL(SW) の実装のため，LL/SC 命令を NiosII にユーザ命令として追加している．

5.3　評価方法

ロック取得要求から解放までの処理時間がコア 1 で長くなるように，コア 1 とその他のコアで処理を変え，コア 1 の計測結果により各アルゴリズムの比較評価を行う．コア 1 はロック L1，L2 を取得するネストロックルーチンを実行する．その他のコアは，コア 1 と同じネストロックルーチンをコア 1 より先行して実行した後，ロック L2 を取得するシングルロックルーチンを 8 回繰り返す．以上の処理を 1 つの単位とし，1,000,000 回繰り返し実

© 2011 Information Processing Society of Japan

行する．このようにすると，コア1が1段目のロックを取得要求したとき，他のコアが先行して1段目のロックを取得する．さらに，1段目のロックが解放された後も，コア1が2段目のロックを取得要求したとき，他のコアが2段目のロック取得を繰り返しているため，競合しやすい．クリティカルセクションは，ネストロックについてCS_1とCS_12の2つ，L2によるシングルロックによるもの（CS_2とする）で，計3つがある．3つのクリティカルセクションの処理内容はすべて空ループである．CS_1とCS_12は，実行時間が約18 μs であり，CS_2の実行時間は約34 μs である．割込みはタイマによって10 ms ごとに発生し，割込み処理の実行時間は約19 μs である．

リアルタイムシステムでは最悪実行時間によりアルゴリズムの性能評価を行うのが望ましいが，マルチコアシステムではバス衝突などの予測が困難な事象により真の最悪値を計測するのが困難である．そのため，計測した結果を確率分布にし，実行の終わる確率が99.999%以上となった時間を比較に用いる．

5.4 評価結果

5.4.1 ロック取得待ち時間とロック解放処理時間

要件 (1) について，ネストロック全体の取得要求から解放までに要した時間を計測することにより評価する．そのため，計測結果にはCS_1とCS_12の実行時間が含まれる．1段目，2段目のロックについて，ロック取得待ち時間とロック解放処理時間を計測するのが理想であるが，計測処理は状況によって実行時間が変化するため，全体の実行時間を計測した．

ロック取得要求から解放までの処理時間について，コア数を1から8まで変えた結果を**図10**に示す．図10はロック取得待ち中に割込みが発生しなかった場合である．図10でCSとしているのは，ロック取得待ちやロック解放処理にかかる時間，バス衝突の影響などがないとしたときの最悪実行時間である．たとえば，4コアの場合，他の3つのコアが先行してロックを取得する場合が最悪であり，自コアのクリティカルセクション実行時間を含めた $4 \times 2 \times 18\,\mu\mathrm{s} = 144\,\mu\mathrm{s}$ が最悪実行時間となる．

simpleは $O(N^2)$ になってしまうため，コア数が増えたときの処理時間が他のアルゴリズムに比べて大きい．TFは処理時間が短いが，割込みを受け付けることができない．TF/PとPPIQL(SW)は，TFを拡張したため，TFより処理時間が増えている．CSとの差分で比較すると，PPIQL(HW)はPPIQL(SW)の52.9%の実行時間となり，1.89倍高速化した．

simple以外のアルゴリズムは，コア数のリニアオーダになるはずだが，ややリニアオーダより悪くなっている．これは，コア数の増加によりデータアクセスについてバス衝突の可能性が高くなっているためと考えられる．バス衝突はハードウェア構成により発生するものであるため，スピンロックアルゴリズムとしては要件 (1) を満たしているとする．

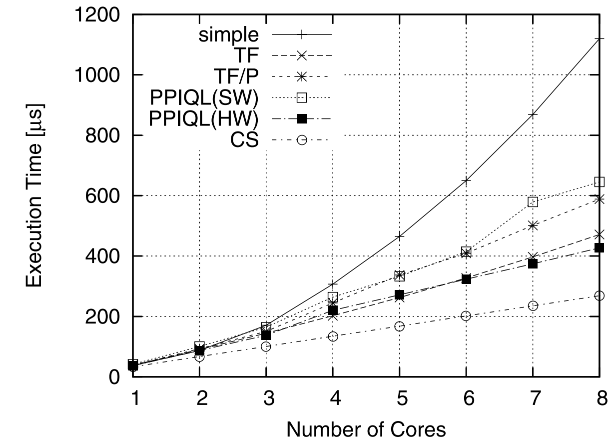

図10　ロック取得要求から解放までの実行時間（割込みなし）

Fig. 10 Execution times from acquring locks to releasing locks without interrupt.

5.4.2 割込み応答時間

要件 (2) については，ネストロック全体の取得要求から解放までの間に発生した割込みに対する応答時間により評価する．具体的には，5.4.1項と同様に各コアがロック取得要求とロック解放を繰り返し，その間に発生した割込み応答時間を計測する．

割込みに対する応答時間について，コア数を1から8まで変えた結果を**図11**に示す．TFはロック取得待ち中に割込み処理を実行することができないため，コア数の増加に応じてロック取得待ち時間とロック解放処理時間が長くなるほど割込み応答時間が悪くなる．クリティカルセクションを実行している18 μs の間以外は割込み処理を実行可能なため，TF以外のアルゴリズムは高速に割込みを受け付けていることが分かる．また，割込み応答時間がコア数に依存しておらず，要件 (2) を満たしている．

5.4.3 割込み処理によるペナルティ時間

要件 (3) については，ロック取得待ち中に割込みが1回だけ発生した場合のネストロック全体の取得要求から解放までに要した時間を計測し，割込みが入らなかった場合（図10）と比較し評価する．

ロック取得待ち中に割込みが1回だけ発生した場合の，ロック取得要求から解放までの処理時間について，コア数を2から8まで変えた結果を**図12**に示す．割込み処理を実行し

© 2011 Information Processing Society of Japan

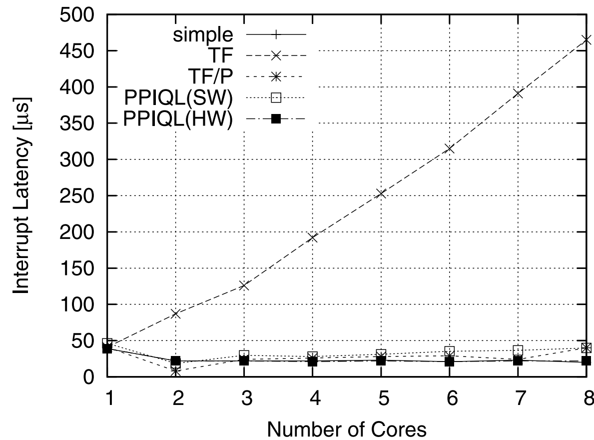

図 11　割込み応答時間
Fig. 11　Interrupt latency.

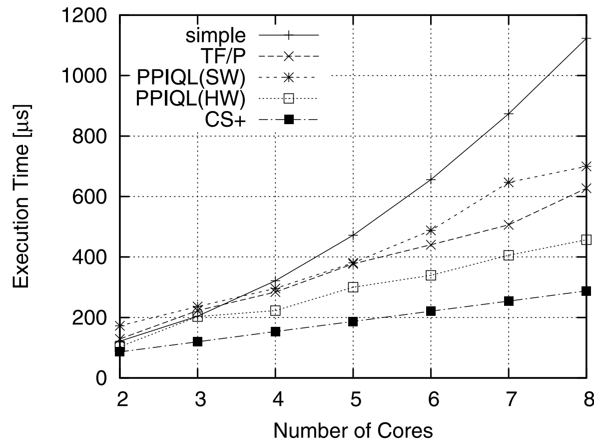

図 12　ロック取得要求から解放までの実行時間（割込みあり）
Fig. 12　Execution times from acquring locks to releasing locks with interrupt.

た分だけ図 10 より実行時間が長くなっているが，傾向は図 10 と同じである．ただし，TF は割込みを受け付けられないため図 12 にはない．クリティカルセクションの実行時間と割込み処理の実行時間を加えた結果を CS+で表しており，4 コアでは $144 + 19 = 163\,\mu$s となる．

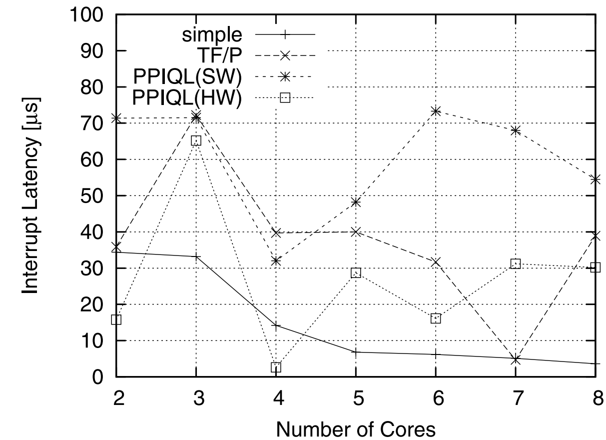

図 13　割込み処理実行時間と割込み処理によるペナルティ時間
Fig. 13　Times of an interrupt service and a penalty for an interrupt service.

図 10 と図 12 の差分は**図 13** のようになる．この差分は，割込み処理実行時間と割込み処理によるペナルティ時間である．コア数と実行時間の変動について規則性が見受けられないため，すべてのアルゴリズムでコア数に依存していないといえる．したがって，この計測結果から TF 以外のアルゴリズムが要件 (3) を満たしていると考えられる．なお，多くのアルゴリズムで 3 コアのときの時間が長くなっているが，これは，前述のように今回の測定が 99.999% 以上となった時間を比較に用いているため，真の最悪値となっておらず，偶然に 3 コアの結果が他のコア数と比較して悪くなったためと考えられる．

5.4.4　優先度継承

図 10 では，TF/P と PPIQL(SW) に大きな違いがない．これは測定結果に優先度逆転が発生したケースが少ないためだと考えられる．優先度継承の効果を測定するため，次のように優先度逆転が発生しやすい状況にした．

- 割込みが発生する間隔を 10 ms から 1 ms に変更．
- コア 2 は，L1 と L2 のネストロックルーチン（コア 1 と同じルーチン）を 8 回繰り返す．
- コア 1 とコア 2 以外のコアは，L1 と L2 のネストロックルーチンを 1 回実行した後，L2 のシングルロックルーチンを 16 回繰り返す．

このような状況でコア数を 2 から 8 まで変えたとき，ロック取得待ち中に割込みが発生し

© 2011 Information Processing Society of Japan

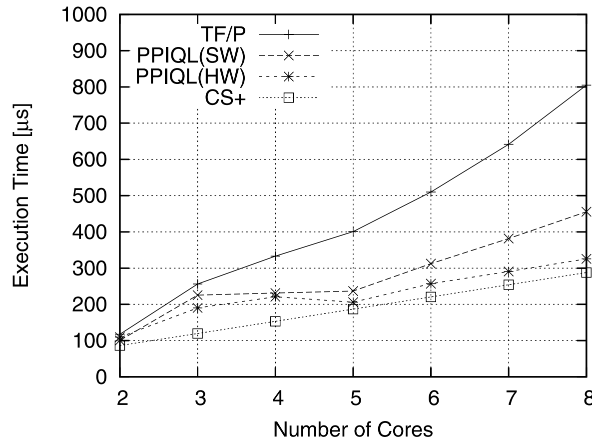

図 14　優先度継承の効果
Fig. 14　Effect of priority inheritance.

た場合のロック取得要求から解放までの処理時間を**図 14** に示す．前述したように，優先度継承の効果が現れるのはコア 1 以外に 3 つ以上（計 4 つ以上）のコアが存在するときである．優先度逆転の最大時間は，コア数が増えると長くなるため，コア数が増えると優先度継承の効果が大きくなる．5 コア以上の場合について，CS との差分で比較すると，PPIQL(HW) は PPIQL(SW) の 32.2% の実行時間となり，3.11 倍高速化した．

優先度逆転は，その発生確率が非常に低いと考えられるが，優先度逆転が発生する可能性のある TF/P は要件 (3) を満たすことができない．ネストロックで要件 (3) を満たすには PPIQL アルゴリズムを利用する必要がある．

5.4.5　ハードウェアサイズ

優先度順スピンロックユニットを 4 コア，8 コアそれぞれについてサイズを計測した．優先度順スピンロックユニットは，コアの数だけ優先度レジスタとロック取得フラグが必要である．16 ビットの優先度レジスタについて，4 コアと 8 コアの実装を行った結果，**表 1** のようになった．比較のために，Mutex 回路，1 チャネルの 32 ビットタイマ，4 コアと 8 コアのシステム全体のハードウェアサイズを計測した．優先度比較はコア間でそれぞれ行っているため，コア間の組合せが増える．具体的には 4 コア時には 6（$= {}_4C_2$）個だった比較器が，8 コア時には 28（$= {}_8C_2$）個になった．提案ハードウェアは，優先度順スピンロックユニットと優先度発行ユニットの組合せでシステム全体に占める割合が最大 2% 程度であり，

表 1　ハードウェアのサイズ
Table 1　Hardware size.

	Mutex	タイマ	優先度発行ユニット	優先度順スピンロックユニット		システム全体	
				4 コア	8 コア	4 コア	8 コア
ALUT 数	47	144	34	201	754	18,297	38,129
logic register 数	33	120	16	85	182	9,805	19,285

表 2　要件の充足
Table 2　Satisfaction of the requirements.

	simple	TF	TF/P	PPIQL(SW)	PPIQL(HW)
要件 (1)	×	◯	◯	◯	◯
要件 (2)	◯	×	◯	◯	◯
要件 (3)	◯	-	×	◯	◯

許容できると考えられる．

5.5　考　　察

以上の評価より，各アルゴリズムの要件の充足について**表 2** に示す．simple はロック取得待ち時間が $O(N^2)$ になってしまうため要件 (1) を満たさないが，要件 (2) と要件 (3) を満たす．TF は要件 (1) を満たすが，割込みを受け付けることができないため，要件 (2) を満たさず，要件 (3) について評価ができない．TF/P は，TF をベースに割込み受付け可能な拡張をしたことで，要件 (1) に加えて要件 (2) も満たすが，優先度逆転により要件 (3) を満たさない．提案アルゴリズムである，PPIQL(SW) と PPIQL(HW) は，TF/P に優先度継承処理を加えたことで要件 (3) を含むすべての要件を満たす．

PPIQL(SW) と PPIQL(HW) を比較すると，PPIQL(HW) では最大 3.11 倍程度の性能向上を得られるが，ハードウェアコストが増加するというデメリットもある．PPIQL を実装するためには CAS 命令や LL/SC 命令が必要なため，これらの命令が用意されていないシステムでは，ハードウェア実装が有効であると考えられる．一方，これらの命令が用意されているシステムにおいて，ソフトウェア実装による性能が要求を満たしている場合では，ソフトウェア実装を用いた方がハードウェアコストを抑えられる．このように，PPIQL の実装方法を選択する際には，ハードウェアコストと性能向上のトレードオフを考慮する必要がある．

© 2011 Information Processing Society of Japan

6. 関連研究

リアルタイム性を考慮した排他制御に関する研究は，2 種類に分けることができる．1 つは RTOS 内部で用いるプリミティブなロック（RTOS 向けロック）であり，もう 1 つはプリミティブなロックを用いて実現するロック（アプリケーション向けロック）である．アプリケーション向けロックは，RTOS の機能としてアプリケーションに提供されるのが一般的である．

RTOS 向けロックのソフトウェアアルゴリズムとしては，中断可能なキューイングスピンロックアルゴリズム[2)]，Totally FIFO アプローチ[5)]，優先度継承スピンロックアルゴリズム[7)] などが提案されている．これらのアルゴリズムは，前述したように 3 つの要件を満たすことができない．RTOS の内部のクリティカルセクションは一般に短く上限時間が定まっているため，RTOS 向けロックアルゴリズムはスピンによりロックの取得を待つ．

RTOS 向けロックのハードウェアサポートによる実現方法が，Saglam らによって提案されている[13)]．このアルゴリズムは，ロックを取得できなかったときにロック状況を繰り返しチェックするのではなく，ロックを解放するコアが次にロックを取得するコアに対して割込みを発生させる機構をハードウェアで実現することにより，ロック取得待ちのバスアクセスを低減させている．ロック取得の順序は FIFO を選択可能なため，ロック取得までの時間に上限があるが，割込み応答性については考慮していないため，要件 (2) を満たさない．

アプリケーション向けロックのソフトウェアアルゴリズムとしては，優先度継承プロトコルをマルチコアシステム向けに拡張した MPCP[10),11)] や FMLP など[12)] の手法が提案されている．これらの手法は，アプリケーションの比較的長いクリティカルセクションを実現するために用いられる．そのため，ロックの取得をスピンで待つのではなく，タスクの状態を待ち状態として待つことで，プロセッサの利用効率を向上させている．タスクの状態を待ち状態とするには，タスクの管理データを操作する必要があるため，RTOS 内部のクリティカルセクションを実現する必要がある．また，FMLP は，クリティカルセクションが短い場合にはスピンで，長い場合はタスクの待ち状態でロックの取得を待つ．しかしながら，スピン時のスピンロックの手法については，OS が提供するスピンロックを使用することを前提している．すなわち，これらのロックを実現するには，RTOS 向けのロックが内部で必要となり，本論文で対象としている RTOS 向けロックとして用いることは難しいと考えられる．これら関連研究との詳細な比較は今後の課題とする．

7. おわりに

本論文では，マルチコアシステムで必須となるコア間排他制御を実現するスピンロックを，リアルタイムシステムで用いる際に求められる要件を満たす，中断可能な優先度継承キューイングスピンロックアルゴリズムを提案した．さらに，提案アルゴリズムの一部をハードウェア化することにより，最大 3.11 倍高速化することを示した．スピンロックというプリミティブな排他制御を 3 倍高速化できたことは，高い応答性が求められるシステムに対する有用性が高いと考えられる．提案ハードウェアは CAS 命令や LL/SC 命令などを用いずにスピンロックを実現できるため，様々なマルチコアシステムで利用できる．また，提案ハードウェアの利用により実現できるアルゴリズムは FIFO 順，優先度順，優先度継承ありなしなどの様々な組合せが可能であり，ソフトウェアの実装を変更するだけで，システムに適したアルゴリズムを選択することができる．

今後の課題として，3 つ以上のネストロックがあげられる．我々が開発している RTOS である，FDMP カーネルでは，2 つのネストロックでよい．しかしながら，FDMP カーネルを発展させた，TOPPERS/FMP カーネルでは，タスクマイグレーションをサポートした関係で，一部の機能の実現のため，3 つのロックを同時に取得する必要がある．そのため，今後は 3 つ以上のネストロックについても提案手法が有効であるか検証する必要がある．

謝辞　本研究は，一部，科研費（21700026）の助成を受けたものである．

参 考 文 献

1) 本田晋也，高田広章：ITRON 仕様 OS の機能分散マルチプロセッサ拡張，電子情報通信学会論文誌 D，Vol.91, No.4, pp.934–944 (2008).
2) 高田広章，坂村　健：中断可能なキューイングスピンロックアルゴリズム，電子情報通信学会論文誌 D-I，Vol.J78-D-I, No.8, pp.661–669 (1995).
3) 高田広章，坂村　健：マルチプロセッサリアルタイムカーネルのスケーラビリティを重視した実現，電子情報通信学会技術研究報告，CPSY，コンピュータシステム，Vol.95, No.603, pp.1–6 (1996).
4) Mellor-Crummey, J.M. and Scott, M.L.: Algorithms for scalable synchronization on shared-memory multiprocessors, *ACM Trans. Comput. Syst.*, Vol.9, No.1, pp.21–65 (1991).
5) Takada, H. and Sakamura, K.: Real-Time Scalability of Nested Spin Locks, *International Workshop on Real-Time Computing Systems and Applications*, pp.160–167 (1995).

© 2011 Information Processing Society of Japan

6) Sha, L., Rajkumar, R. and Lehoczky, J.: Priority Inheritance Protocols: An Approach to Real-Time Synchronization, *IEEE Trans. Comput.*, Vol.39, pp.1175–1185 (1990).
7) 王　才棟，高田広章，坂村　健：優先度継承スピンロックアルゴリズムとその評価，情報処理学会論文誌，Vol.38, No.11, pp.2262–2273 (1997).
8) Carlini, A. and Buttazzo, G.C.: An efficient time representation for real-time embedded systems, *SAC '03: Proc. 2003 ACM symposium on Applied computing*, pp.705–712, New York, NY, USA, ACM (2003).
9) Markatos, E.P.: Multiprocessor Synchronization Primitive with Priorities, *Proc. IEEE Workshop Real-Time Operating Systems and Software* (1991).
10) Rajkumar, L.S.R. and Lehoczky, J.: Real-time synchronization protocols for multiprocessors, *Proc. 9th IEEE International Real-Time Systems Symposium*, pp.259–269 (1988).
11) Lakshmanan, K., de Niz, D. and Rajkumar, R.: Coordinated Task Scheduling, Allocation and Synchronization on Multiprocessors, *Real-Time Systems Symposium, IEEE International*, Vol.0, pp.469–478 (2009).
12) Brandenburg, B.B. and Anderson, J.H.: An Implementation of the PCP, SRP, D-PCP, M-PCP, and FMLP Real-Time Synchronization Protocols in LITMUS^{RT}, *Proc. 2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.185–194, Washington, DC, USA, IEEE Computer Society (2008).
13) Saglam, B.E. and Mooney, V.J. III: System-on-a-Chip Processor Synchronization Support in Hardware, *Design, Automation and Test in Europe Conference and Exhibition*, pp.633–641 (2001).

(平成 22 年 10 月 1 日受付)
(平成 23 年 1 月 19 日採録)

一場　利幸（学生会員）

2009 年名古屋大学大学院情報科学研究科博士前期課程修了．現在，同博士後期課程在学．マルチコアシステムのリアルタイム性，リアルタイム OS の研究に従事．

松原　豊

名古屋大学大学院情報科学研究科附属組込みシステム研究センター研究員．2006 年名古屋大学大学院情報科学研究科博士前期課程修了．2009 年同博士後期課程単位取得満期退学．2009 年 4 月より現職．リアルタイム OS，リアルタイムスケジューリング理論に関する研究に従事．博士（情報科学）．

本田　晋也（正会員）

2002 年豊橋技術科学大学大学院情報工学専攻修士課程修了．2005 年同大学院電子・情報工学専攻博士課程修了．2005 年名古屋大学情報連携基盤センター名古屋大学組込ソフトウェア技術者人材養成プログラム産学官連携研究員．2006 年名古屋大学大学院情報科学研究科附属組込みシステム研究センター助教．現在同准教授．リアルタイム OS，ソフトウェア・ハードウェアコデザインの研究に従事．博士（工学）．2002 年度情報処理学会論文賞受賞．電子情報通信学会，日本ソフトウェア科学会各会員．

高田　広章（正会員）

名古屋大学大学院情報科学研究科情報システム学専攻教授．1988 年東京大学大学院理学系研究科情報科学専攻修士課程修了．同専攻助手，豊橋技術科学大学情報工学系助教授等を経て，2003 年より現職．2006 年より大学院情報科学研究科附属組込みシステム研究センター長を兼務．リアルタイム OS，リアルタイムスケジューリング理論，組込みシステム開発技術等の研究に従事．オープンソースの ITRON 仕様 OS 等を開発する TOPPERS プロジェクトを主宰．博士（理学）．IEEE，ACM，電子情報通信学会，日本ソフトウェア科学会各会員．

© 2011 Information Processing Society of Japan