4A-4

マルチコアアプリケーション間通信方式の設計と評価

元濱努† 片山吉章† 松本利夫† 山本治‡
†三菱電機 (株) ‡(株) ルネサステクノロジ

1 はじめに

近年，組込みソフトウェアの機能や性能要求が増大しており，高性能化と低費消費電力化を同時に実現するマルチコアが注目されている．マルチコアアプリケーションの開発ではコスト削減のために，シングルコア上で動作していた従来のソフトウェア資産を再利用することが考えられる．しかし，従来のソフトウェアは，並列処理を考慮した設計がされていないため，マルチコア用に最適化する必要があり，開発にコストがかかってしまう．

本稿では，従来のソフトウェアを再利用する枠組みを提供するマルチコアアプリケーション通信方式の設計とその基本性能評価について述べる．

2 設計

2.1 想定環境

マルチコア上で動作するソフトウェアの形態として，複数の CPU コアが同等な立場で処理を行う SMP（Symmetric Multiple Processor）と各々の CPU コアに機能を固定して処理を行う AMP（Asymmetric Multiple Processor）がある．SMP はタスクがどの CPU コアで処理されるかは実行するまで予測不可能なため，リアルタイム性を保証することが困難である．そのため，タスクの実行順序の予測が容易である AMP が組込み機器で採用されることが多い．以下，本稿では AMP で開発することを前提にして述べる．

2.2 課題

AMP では，動的な負荷分散をサポートしていないため，CPU コアを効率よく使うための機構をソフトウェアレベルで実現する必要がある．すなわち，従来のソフトウェアを機能分割し各 CPU コア上に割り当て，連携や協調するための機能の追加が必要となる．

しかし，従来のソフトウェアは各種ライブラリが密に結合されているため，機能分割は容易ではない．また，他 CPU コアからの応答順によっては，ソフトウェアの実行順序が予測不可能となり，リアルタイム性を保証することが困難となる．

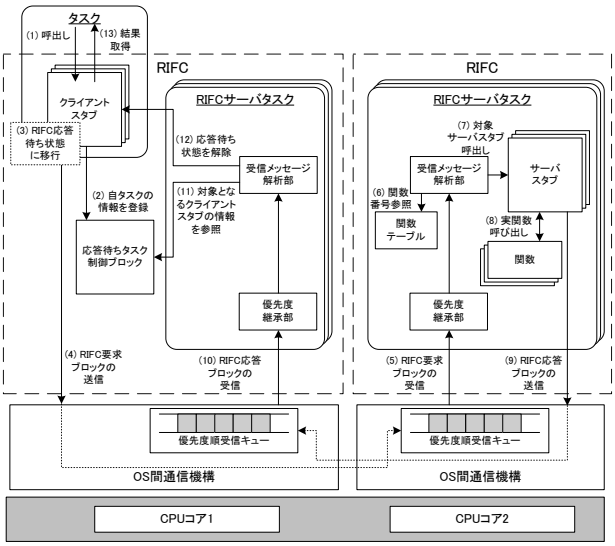

図 1: 機能構成と処理手順

2.3 方針

機能分割を実現するための手法として，ネットワークを通じて異なる計算機上にある関数を呼び出す枠組みである RPC[1] をマルチコア環境に適用する．他コア上にある機能を一般の関数呼び出しの形で行うことが可能となるため，ソフトウェアを改変することなく機能分割を実現することが可能となる．しかし，既存の RPC はリアルタイム性を考慮していないため，別途そのための機構を設ける．

RPC と同様に分散処理を実現するための枠組みでさらにリアルタイム性を考慮した Real-time CORBA[2] がある．しかし，本来ネットワークを介すことを前提としているため，CPU コア間で用いるには冗長な機能が含まれており，実装サイズが大きく，制約の厳しい組込み機器には適用できない．本方式では，省メモリで軽量に動作するように設計する．

2.4 構成

本方式の機能構成と処理手順を図 1 に示す．本方式は，RPC にリアルタイム性を考慮した RIFC(Real-time Inter-core Function Call) と共有メモリ上に置かれた優先度順受信キューを介して各 CPU コア上の OS と通信を実現する OS 間通信機能により構成される．以下に，各構成要素の概要について示す．

RIFC 要求/応答ブロック RIFC 要求/応答に必要となるパラメータ (要求元タスク ID，優先度，RIFC

The Design and Evaluation of Communicaton Mechanism for Multi-core Application
Tsutomu MOTOHAMA†, Yoshiaki KATAYAMA†,
Toshio MATSUMOTO†and Osamu YAMAMOTO‡
†Mitsubishi Electric Corp. ‡Renesas Technology Corp.

対象関数番号，引数，返り値) などをパッキングしているブロックである．

クライアントスタブ RIFC 対象関数を呼び出すための RIFC 要求ブロックを生成し，該当する関数を処理可能なコア上の RIFC サーバタスクに送信する．また，処理結果を自 OS 上の RIFC サーバタスクより受け取り，呼び出し元に返す．クライアントスタブは，RIFC 対象関数ごとに用意され，対象関数と同じインタフェースを持つ．

サーバスタブ クライアントスタブから送られてきた RIFC 要求ブロックから RIFC 対象関数を実行するための情報を取り出し，実行する機構である．また，RIFC 対象関数より得た結果を返すための RIFC 応答ブロックを作成し，他コア上の RIFC サーバタスクに送信する．サーバスタブもクライアントスタブと同様に対象関数ごとに用意される．

RIFC サーバタスク RIFC サーバタスクは，RIFC 要求/応答ブロックを受信し，その受信したブロックに応じて，RIFC 要求/応答の処理を実行するタスクである．RIFC サーバタスクは，RIFC 発行元のタスク優先度を継承して処理を行う．また，1 つの CPU コア上で複数の RIFC サーバタスクを持つことも可能である．

応答待ちタスク制御ブロック RIFC 要求後から結果が返ってくるまでの間，クライアントスタブの実行をブロッキングする必要がある．応答待ちタスク制御ブロックは，受信結果を格納する領域のアドレスとスタブを実行しているタスク ID を管理する情報を持つ．これらの情報を基に RIFC サーバタスクは，クライアントスタブへの実行結果の受渡しとブロッキング解除を行う．

OS 間通信機能 本機能は，OS 間インタフェース [3] に優先度情報を考慮したものである．RIFC の Low-Level API であり，共有メモリを介した優先度つき可変長メッセージの転送を実現し，それらのメッセージを優先度順にキューイングする機構を持つ．

クライアントスタブによって，RIFC 要求をするためのパラメータパッキング処理と OS 間通信が隠蔽されるため，コンパイル時にリンクするライブラリをクライアントスタブのコードに変更するのみで，ソフトウェアを改変することなく RIFC を実現することが可能である．また，各種スタブのコードはジェネレータにより自動生成されるため，開発にコストをかけることなく容易に機能分割を実現することが可能である．

RIFC サーバタスクで実行される RIFC 対象関数の処理は，RIFC 要求元のタスク優先度を継承して行うため，各 CPU コアでの優先度の差異を解消することが可能である．また，RIFC 要求を優先度順にキューイングする機構を持つため，多数の RIFC 要求を受けた際にも発行順によって優先度逆転現象が発生することはない．

3 性能評価

本方式の基本性能を評価するために，入力/出力パラメータともに同じサイズである関数を RIFC 化し，クライアントスタブ呼び出しから返り値を取得するまでの経過時間とその経過時間の中から OS 間通信に費やした時間を測定した．ただし，サーバスタブ上で実行される RIFC の対象となる実関数の実行時間は含めておらず，RIFC に必要なタスク以外は実行されていない．表 1 に測定結果を示す．

評価環境として、M32R コア [4] を 2 つ搭載するシングルチップマルチプロッセサ上に OS として T-Kernel[5] を用いた．

表 1: 測定結果

サイズ (byte)	処理全体 (usec)	OS 間通信 (usec)
16	373.3	236.2
64	386.0	249.0
256	427.9	280.4
1024	660.6	430.8
4096	1860.7	1267.8
16384	6465.9	4752.0

上記結果により，RIFC の処理全体のうち 6 割後半が OS 間通信によるオーバヘッドであることがわかる．大半の関数はデータサイズが 64byte 程度で収まることが想定され，そのオーバヘッドは 400usec 程度であり RIFC 化によるオーバヘッドは問題ないと考えられる．しかし，画像や動画のエンコード処理を行う関数や大きなリスト構造を持つ関数など受け渡すデータサイズが大きい場合は，RIFC 化のオーバヘッドが問題となる可能性がある．ボトルネックとなっている OS 間通信の改善やデータの受渡し方法を工夫することで適用可能としていきたい．

4 おわりに

本稿では，RIFC の設計とその基本性能評価について述べた．RIFC を用いることにより，従来のソフトウェアを改変することなくリアルタイム性や並列処理といった問題を解決することが可能となる．今後，OS 間通信の性能を改善するとともに，今回，性能評価として外的要因のない中で行ったが，従来のソフトウェアに適用し，多数の RIFC が競合したときなどにも優先度が正しく継承され，リアルタイム性能に問題がないか検討していく予定である．

参考文献

[1] RPC:http://www.ietf.org/rfc/rfc1831.txt

[2] Real-time CORBA:http://www.omg.org/

[3] 菅井，遠藤，山口，近藤:「シングルチップマルチプロッセ上のハイブリット OS 環境の実現 - OS 間インタフェースの実装 -」，情報処理学会第 66 回全国大会 (2004)

[4] M32R:http://japan.renesas.com/fmwk.jsp?cnt=m32r_family_landing.jsp&fp=/products/mpumcu/m32r_family/

[5] T-Kernel:http://www.t-engine.org/T-Kernel/tkernel.html