Vol.2009-EMB-14 No.6
2009/7/24

AUTOSAR通信ミドルウェアのマルチコア拡張

一場利幸[†1]　高田広章[†1,†2]
本田晋也[†2]　倉地亮[†2]

車載システム向けソフトウェアプラットフォームである AUTOSAR はシングルコアのみを想定しており，マルチコア環境を考慮していない．車載システムは相互に通信を行うことで協調制御するため，通信処理を実現するミドルウェアのマルチコア対応が必須である．本研究では，マルチコア環境での AUTOSAR 通信ミドルウェアの実現方法と評価を行う．まず，通信をコア内通信，コア間通信，チップ外通信の 3 つに分類し，それぞれの実現方法を検討する．チップ外通信については，ロック方式，通信サーバ方式の 2 方式を提案する．既存の通信ミドルウェアを 2 方式で拡張し，それぞれの送信処理時間の比較を行った．シングルコア環境に比べ，ロック方式は 41%，通信サーバ方式は 29%のオーバヘッドで実現可能であることがわかった．

An Extension of AUTOSAR Communication Layers for Multicore Systems

Toshiyuki Ichiba,[†1] Hiroaki Takada,[†1,†2]
Shinya Honda[†2] and Ryo Kurachi[†2]

AUTOSAR, a software platform for automotive systems, assumes only an ECU with a single core processor, and does not consider an ECU with a multicore processor. It is necessary that a communication on a multicore processor because ECUs communicate each other for cooperative control. There are three communication types: a inside core, a inter core, and a outside chip communications. This paper proposes two methods for outside chip communication: a lock method and a server method. An existing communication middleware is extended as the two methods. Compared to a single core processor, the results of extended middleware show that transmission time overheads of the lock method and the server method are 41% and 29%, respectively.

1. はじめに

近年，自動車に搭載される ECU (Electronic Control Unit) と呼ばれるコンピュータは複雑化，高機能化が進んでいる．自動車 1 台に搭載される ECU の総数は増大が進んでおり，ソフトウェアも大規模化の一途をたどっている．ECU はネットワークを用いて相互に接続されており，ECU 間の通信プロトコルとして CAN (Controller Area Network)[1] が広く用いられている．しかし，ECU の増加によって，ECU の設置スペースとそれらを接続する配線のスペースを確保するのが困難になってきている．このため，複数の ECU を 1 つの ECU に統合し，ECU の総数を減らす動きがある．

ソフトウェアの大規模化に対して，過去に開発したソフトウェアを他製品でも利用できるようなプラットフォームの標準化が進められている．その代表的なものとして，AUTOSAR (AUTomotive Open System ARchitecutre)[2] が挙げられる．AUTOSAR 仕様には OS や CAN 通信ミドルウェアなどが含まれており，プラットフォームとして利用できる．

ECU の総数を減らすために統合を行うには，プロセッサの性能向上が必須である．マルチコアプロセッサは消費電力や発熱量の点でシングルコアプロセッサより優れており，搭載の条件が厳しい自動車においても性能向上の一つの解として利用が検討されている．しかし，マルチコア環境で AUTOSAR プラットフォームを利用することは想定していない．

本研究では，AUTOSAR 仕様の CAN 通信ミドルウェアをマルチコア拡張する．マルチコア環境では，共有するリソースを使用する際にコア間の排他制御が必要である．CAN 通信ミドルウェアをマルチコア拡張するためには，コア間で共有する 1 つの CAN コントローラを排他制御する必要がある．マルチコア環境においては，コア間での通信のように CAN コントローラを利用する必要がない場合もあるため，まず，CAN 通信ミドルウェアの通信を分析し，マルチコア環境での通信を 3 つに分類する．そして，CAN コントローラを排他制御する必要のあるチップ外通信について 2 方式を提案し，その評価を行う．

本稿では，2 章で AUTOSAR の概要と通信処理の流れについて述べ，3 章でマルチコア拡張にあたっての前提と要件について述べる．4 章でマルチコア環境における通信を 3 つに分類した上で，実現可能な拡張方法を整理し，チップ外通信について 2 方式を提案する．5

†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

章で提案方式の評価を行い，6 章でまとめる．

2. AUTOSAR 通信ミドルウェア

2.1 AUTOSAR の概要

ソフトウェアの大規模化による開発コストの増加，部品メーカごとに異なるインタフェースを利用していたことなどによる互換性の低さを背景として，AUTOSAR による ECU のソフトウェアの標準化が行われている．標準化の取り組みは以前から行われており，その一例として OSEK/VDX[3] が挙げられる．OSEK/VDX が策定した OS や通信ミドルウェアは，AUTOSAR 仕様のベースとなっており，AUTOSAR ではさらにデバイスドライバなども仕様に含めることで，ECU のハードウェアの違いを吸収することを目指している．

AUTOSAR 仕様ではハードウェアやサービスを抽象化し，アプリケーションレベルでそれを意識する必要がない構成になっている．また，構成するモジュール間にはインタフェースが定義されており，モジュール単位での再利用が可能となっている．図 1 に示すように，AUTOSAR 仕様でのソフトウェア構成は大きく BSW (Basic Software)， RTE (Runtime Environment)， Application Layer の 3 つに分類することができる．

BSW はハードウェアの機能を抽象化し，ECU のサービスを提供する．OS や通信ミドルウェアはここに含まれる．RTE は，BSW により提供されるサービスとアプリケーションとの接続を行う．アプリケーションは Application Layer 内の SW-C (Software Component) と呼ばれる単位で構成される．SW-C は RTE を利用することで実行環境の違いが吸収されるため，再利用が可能となる．

2.2 通信ミドルウェア

通信ミドルウェアの構成は最小セットを図 2 に示す．COM は通信サービスを提供，PDU Router はデータのルーティングを行う．CAN Interface は CAN プロトコルを抽象化し上位層へのインタフェースを与える．CAN Driver は CAN コントローラに依存した処理を行う．

送信は，以下のように行われる．SW-C はシグナルと呼ばれる単位で RTE を経由して通信を行う．COM は RTE から渡されたシグナルを I-PDU と呼ばれる単位にまとめて PDU-R に渡す．COM は I-PDU のためのバッファを用意し，SW-C からシグナルが渡されたら対応する I-PDU バッファを更新する．I-PDU バッファを更新する際に，シグナルグループとしてまとめられたシグナルを一括して更新することができる．シグナルグループのためのバッファをシャドウバッファと呼び，COM はシグナルをシャドウバッファに入れてから，

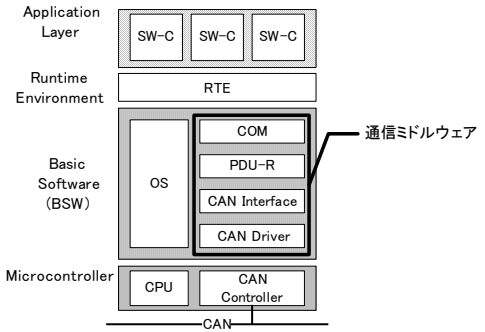

図 1 AUTOSAR 仕様の OS・通信ミドルウェアを利用する構成

I-PDU バッファにそれをコピーする．次に，COM は I-PDU バッファの I-PDU を次の 3 種類のモードで送信する．

- Direct/N-Times はシグナルが更新されたときに送信を開始し，一定間隔で N 回送信する．
- Periodic はシグナルが更新されてもすぐに送信をせず，一定間隔で送信し続ける．
- Mixed は上記 2 つの組み合わせで，シグナルが更新されたときに送信し，さらに一定間隔で送信し続ける．

COM が送信した I-PDU は，PDU-R，CAN Interface，CAN Driver の順に渡されて CAN Controller によって CAN バス上に流される．

3. マルチコア拡張の前提と要件

3.1 マルチコア拡張の前提

ハードウェアの構成は，図 3 のように 2 つのプロセッサコアと 1 つの CAN コントローラを想定する．プロセッサは，共有メモリを持ち，コア間割込みをサポートしている．CAN コントローラについては，両コアからアクセス可能とする．コア数と同数の CAN コントローラを想定するのはネットワークにおけるノード数の制限により実用的でないため，CAN コントローラを 1 つと想定する．SW-C とコアとの関係はシステム設計時に静的に決まるものとし，システムの動作中に SW-C がコアを移ることは考えない．AUTOSAR 仕様の通信ミドルウェアは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で開発したものを利用する．

Vol.2009-EMB-14 No.6
2009/7/24

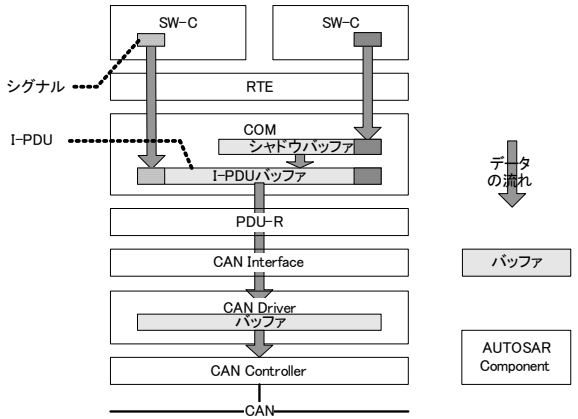

図 2 通信の流れ

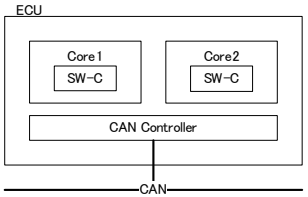

図 3 対象とするハードウェア構成

3.2 マルチコア拡張の要件

通信ミドルウェアのマルチコア拡張要件は以下のように定める．

(1) SW-C の変更は行わない
(2) マルチコア拡張による実行オーバヘッドを小さくする
(3) リアルタイム性を確保できる
(4) CAN コントローラの変更を最小限にする
(5) AUTOSAR 仕様に対する修正を最小にする

(1) を満たすことにより，シングルコア環境での SW-C をマルチコア環境でも利用できる．ただし，(2) を考慮しないと実用的なものにならない．(3) について本研究では，コア間の排他区間に注目し，排他処理にかかる時間の上限が定まること，そしてその上限値が小さいことを要件とする．(4) は，本研究の成果が可能な限り汎用的なハードウェアで利用できるようにするためのものである．(5) は，なるべくシンプルな拡張方法を提案するために定めた．

3.3 排他制御の必要性

各層で行う処理には，クリティカルセクションが存在する．例えば，COM 内部の I-PDU バッファにシグナルをパッキングする際，シグナルがバイト境界をまたいでいると一度に I-PDU バッファにパッキングすることができない．このパッキング処理中に別のパッキング処理を行わないようにしないと，送信を要求したシグナルとは別のシグナルを受信してしまう．COM の他に，CAN Driver にもバッファが存在し，クリティカルセクションとして保護しなければならない．AUTOSAR 仕様では同一コア内での排他制御について規定されているが，コア間の排他制御は考慮されていない．以降，本稿では同一コア内の排他とコア間の排他をそれぞれコア内排他，コア間排他と区別して呼ぶことにする．本研究で用いる通信ミドルウェアでのクリティカルセクション数を表 1 に示す．

表 1 各層の API とクリティカルセクション数

層	API	クリティカルセクション数
COM	`Com_UpdateShadowSignal`	1
	`Com_SendSignal`	3^{*1}
	`Com_SendSignalGroup`	3^{*1}
	`Com_MainFunctionTx`	(I-PDU) ×3+(タイムアウトした I-PDU)+1
CAN Interface	`CanIf_Transmit`	0^{*2}
CAN Driver	`Can_Write`	1

∗1 アップデートビットを用いない場合は 2
∗2 `Can_Write` が BUSY のときは 1

4. 通信の分類とマルチコア拡張

通信ミドルウェアを拡張するにあたって，マルチコア環境での通信を図 4 のようにコア内通信，コア間通信，チップ外通信の 3 つに分類し，それぞれのマルチコア拡張方法について議論する．

4.1 コア内通信

コア内通信は，同一コア内の異なる SW-C が行う通信である．コア内排他のみを考慮すればよいため，現状の AUTOSAR 仕様のまま実現可能である．

ここで，RTE の動作するコアについて考えるために，RTE が片方のコア（コア 1）のみで動作する構成を考える．この状況では，もう片方のコア（コア 2）にある SW-C が同じコアにある SW-C との通信をする際，SW-C は RTE を経由しなければならないため，コア 1 の RTE を経由する必要がある．コア内通信はコア内で閉じた処理になるべきであるが，コア 1 のみで RTE が動作する構成では，コア 2 のコア内通信にコア 1 を利用しなければならない．従って，RTE を片方のコアのみで実行する構成は考えず，コアごとに RTE が動作する構成を想定する．

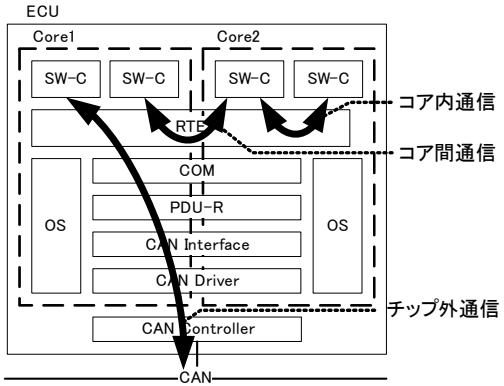

図 4 マルチコア環境における通信の分類

4.2 コア間通信

コア間通信は，異なるコアにあるアプリケーションが行う通信である．この実現方法として，RTE, COM, および PDU-R の 3 つの層で行う方法が考えられる．

RTE で行う方法は，コア内通信にコア間の排他制御を加えることにより実現する方法で，シグナル単位で通信を行う．経由する層が最も少なく処理時間が短いが，シグナル単位のため頻繁にコア間通信が発生する可能性がある．シグナルを複数にまとめてコア間通信を行う方法も考えられるが，COM で I-PDU にまとめる処理をしているため，同様の処理を RTE で提供することは AUTOSAR 仕様への変更が大きい．シグナルより大きい単位でコア間通信をするには，COM もしくは PDU-R による方法が自然である．

COM で行う方法は，COM がシグナルを I-PDU バッファにまとめるのでそれをコア間で共有することで通信を行う．COM を経由することによりコア間通信においても，周期送信などの送信モードを利用することができる．この方法は I-PDU バッファにコア間排他を加えれば実現可能であるが，I-PDU ごとにコア間通信かどうかを判別してコア間排他をしなければならない．また，COM では RTE から RTE への経路はないため，AUTOSAR 仕様への変更が大きい．

PDU-R で行う方法は，PDU のルーティング対象に別コアを追加してコア間通信に拡張する．ルーティングをする必要がないときは PDU-R は省略され，COM と CAN Interface 間は直接呼び出される（ゼロコストオペレーションと呼ばれる）が，この方法では PDU-R を省略することはできなく，ゼロコストオペレーションをサポートできない．また，他の方法に比べて経由する層が多いため，処理時間が長い．

表 2 コア間通信の実現方法

実現する層	RTE	COM	PDU-R
利点	経由する層が少なく 処理時間が短い	I-PDU で送信できる 送信モードを利用できる	I-PDU で送信できる 送信モードを利用できる
欠点	シグナル単位のため 頻繁に通信する 可能性がある	AUTOSAR 仕様への 変更が大きい， I-PDU ごとに判別が必要	経由する層が多く 処理時間が長い， ゼロコストオペレーション をサポートできない

3 つの層で実現する方法について，以上の議論を表 2 にまとめる．PDU-R で行う方法の利点は COM で行う方法と同じため，有用なのは RTE と COM で行う方法である．

4.3 チップ外通信

マルチコア拡張の前提として，SW-C はコア固定なので，SW-C に対応するシグナルと I-PDU もコア固定である．従って，I-PDU バッファをコア間で共有する必要がなく，COM はコアごとに動作させることができる．つまり，COM ではコア間排他をする必要がなく，コア内排他のみでよい．

チップ外通信では，RTE から CAN Driver までの層を経由する必要があり，CAN コントローラが 1 つであるためコア間排他をどの層で行うのかを検討しなければならない．本研究ではロック方式と通信サーバ方式の 2 方式を提案する．

4.3.1 ロック方式

ロック方式では，ロックを取得してからクリティカルセクションの処理を行う．あるコアがロックを取得しているときは，他のコアはロックに対応するクリティカルセクションを実行することができない．CAN Driver によるロック方式を，図 5 に示す．

ロック方式では，ロック取得回数とロック競合に注意しなければならない．ロック取得回数が増えると，通信処理全体でのロック取得時間が長くなる．しかし，少数のロックで広範囲のクリティカルセクションを保護すると，ロック競合が発生しやすくなり，ロックを取得するまでの待ち時間が長くなる．

本研究ではコア間排他は 1 回だけ行うので，ロック取得回数は最小である．COM でロックを取得する方法は COM のクリティカルセクションが多いため，ロック競合が発生しやすい．そのため，CAN Driver でコア間排他のためのロックを取得する方法を提案する．こ

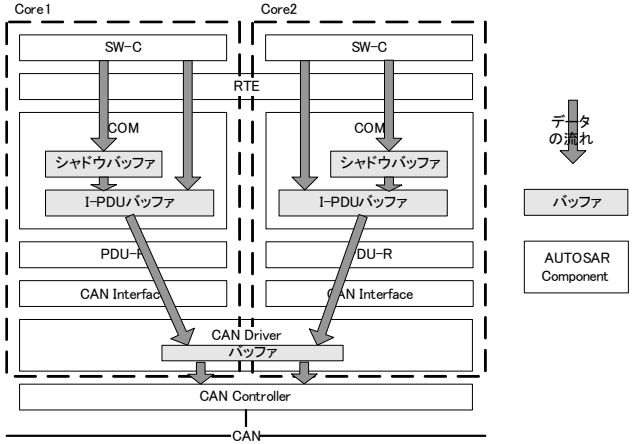

図 5 ロック方式の概念図

の方式では，CAN Driver 以外の通信ミドルウェアはシングルコア構成のまま利用することができ，AUTOSAR 仕様に対する変更点が少ない．

4.3.2 通信サーバ方式

通信サーバ方式は，通信処理を 1 つのタスク（サーバタスク）にして通信処理を並列に動作させない．SW-C はクライアントタスクとなりサーバタスクに通信処理を依頼する．サーバタスクは 1 つのコアで動作するため，コア間排他が必要なくなる．また，サーバタスクが多重に起動させなければ，コア内排他も必要ない．本研究ではサーバタスクは 1 つのみとする．CAN Interface と CAN Driver をサーバタスクにした場合を図 6 に示す．

通信サーバ方式では，クライアントタスクとサーバタスクに分割されるため，各層の API の返り値に注意しなければならない．クライアントタスクがサーバタスクに処理を依頼してもクライアントタスクがサーバタスクの返り値が必要だと，クライアントタスクは依頼した処理が終了するまで待たなければならない．サーバタスクには通信処理が集中するので，クライアントタスクに処理結果を通知するまで時間がかかる可能性がある．

AUTOSAR 仕様の通信ミドルウェアでは，COM の API が下位層の API の返り値を利用しない．そのため，COM の処理までをクライアントタスクにし，COM より下位層をサーバタスクにすることでクライアントタスクが返り値のために待つ時間を少なくできる．クライアントタスクとサーバタスク間のデータ通信はデータキューを利用する．クライアントタスクからデータキューへの送信は PDU-R で行うのが簡単なので，PDU-R までをクライアントタスクとする．この方式では，タスクとデータキューを通信ミドルウェアに取り入れる必要がある．

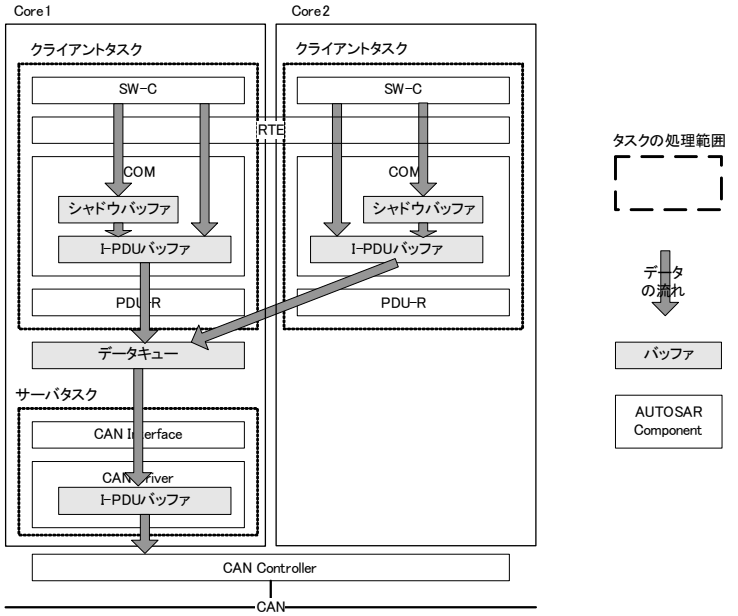

図 6 通信サーバ方式の概念図

5. 評　　価

チップ外通信についてロック方式と通信サーバ方式を実装し，シングルコア環境の場合と比較することでそれぞれの処理時間のオーバヘッドを計測した．

5.1 評 価 項 目

以下の項目について評価を行う．

- シングルコア環境での送信処理時間
- ロック方式の送信処理時間
- 通信サーバ方式の送信処理時間

シングルコア環境との比較を行うことで，送信処理時間のマルチコア拡張によるオーバヘッドを明確にする．

5.2 評価環境

評価は Altera の NiosII プロセッサを用いた．コンパイラは GNU gcc version 3.4.6 である．OS は TOPPERS/FMP カーネル Release 0.B.0[4]（以降，FMP カーネル）を用い，FMP カーネルの API を利用してコア間排他を実現する．FMP カーネルは μITRON 仕様をベースにしているため AUTOSAR 仕様の OS とは異なるが，AUTOSAR 仕様の OS での排他制御はリソースにより実現可能なので，マルチコア拡張は同様の方法が行える．

2 つの NiosII プロセッサ（85MHz）と 1 つの 10Mbps CAN[5] コントローラを 1 台のボードに用意し，2 台のボード間で通信を行う．10Mbps CAN コントローラは CAN コントローラをベースにしているため，差異はわずかであり，CAN Driver でその違いは吸収される．そのため，本研究で対象としているマルチコア拡張の比較には 10Mbps CAN と CAN の違いは問題とならない．

5.3 実　　装

コア内排他はセマフォを利用する方法（FMP カーネルの `wai_sem/sig_sem` で実現）と割込み禁止による方法（FMP カーネルの `loc_cpu/unl_cpu` で実現）がある．

ロック方式における CAN Driver でのコア間排他は，`wai_sem/sig_sem` を利用して実装した．FMP カーネルの利用により，実装は容易である．

通信サーバ方式でのデータキューは `snd_dtq/rcv_dtq` により実装した．CAN Interface には送信データへのポインタと I-PDU の ID などを渡す必要があるため，1 つの送信データに対しデータキューを 2 つ用いる．サーバタスクはデータキューが空のときは受信待ちをしており，クライアントがデータキューに送信するとサーバタスクが起動する．

5.4 測定方法

シングルコア環境は，ロック方式，通信サーバ方式と同じ評価環境で測定する．このとき，片方のコアは無限ループでアイドリング動作させ，もう一方のコアのみで処理を行う．

本研究で利用する通信ミドルウェアは下位層の API を呼び出した後，返値を返すだけなので，API の入り口に計測ポイントを置いて各層の処理時間を計測している．

送信処理に関して，RTE を呼び出してから戻ってくるまでの時間を 20000 回繰り返し計測し，最頻値を採用する．このとき，送信するシグナルは 18 ビットであり，対応する I-PDU は Direct/N-Times (N=1) に設定してあるため，すぐに送信処理が行われる．COM の API は `Com_SendSignal` を利用する．

ロック方式の計測は次の条件下で行った．

- RTE に送信要求を出すタスクがコアごと 1 つずつ存在する
- 片方のコアを計測している間，一方のコアはアイドリングさせ，セマフォは必ず取得できる状態にする

通信サーバ方式の計測は次の条件下で行った．

- RTE に送信要求を出すタスクがコアごと 1 つずつ存在する
- コア 1 に，サーバタスクが 1 つ存在する
- 片方のコアを計測している間，一方のコアはアイドリングさせ，セマフォは必ず取得できる状態にする

サーバタスクがコア 1 のみで動作するため，計測条件がコア 1 とコア 2 で一致しない．そのため，サーバタスクが起動しない様に `snd_dtq` の処理時間は別途計測した．通信サーバ方式では，サーバタスクの優先度はクライアントタスクより低く設定されているため，RTE を呼び出してから戻ってくるまでの処理にサーバタスクの処理時間は含まれない．サーバタスクの処理時間は `rcv_tsk` と CAN Interface, CAN Driver の処理時間であるので，`rcv_tsk` と，シングルコア環境での CAN Interface, CAN Driver の処理時間を加えて比較を行う．

5.5 結　　果

コア内排他にセマフォを利用した結果を図 7 に，割込み禁止を利用した結果を図 8 に示す．1 回の計測で 2μs 程度ぶれることがわかっており，コア 1 とコア 2 の結果の違いはこの計測誤差だと思われる．通信サーバ方式をシングルコアと比較すると，データキューの送受信処理のため PDU-R の処理時間が延びている．割込み禁止を利用した場合，シングルコアの処理時間に比べてコア 2 は 28.5%のオーバヘッドが生じている．本来，データキューの受信処理は PDU-R の処理時間ではないが，送受信処理のオーバヘッドを明確にするために PDU-R にまとめている．一方，ロック方式は，セマファを利用した場合ではシングルコアとほぼ同じだが，割込み禁止を利用した場合では CAN Driver でコア間排他のために処理時間が 2 倍以上かかっている．割込み禁止を利用した場合，シングルコアの処理時間に比べてコア 1 での通信処理全体のオーバヘッドは 40.8%であった．

図 7 と図 8 を比較すると，コア内排他の実現方法により処理時間は大きく変化することがわかる．割込み禁止によるコア内排他は処理時間が短いが，割込みを禁止しているために割込みハンドラすべてに影響を与える．一方，セマフォによるコア内排他はそのセマフォを取得するタスクにしか影響を与えないが，取得にかかる時間が長いため処理時間が長くなる．COM では 3 つのクリティカルセクションがあるためコア内排他にセマフォを用いると

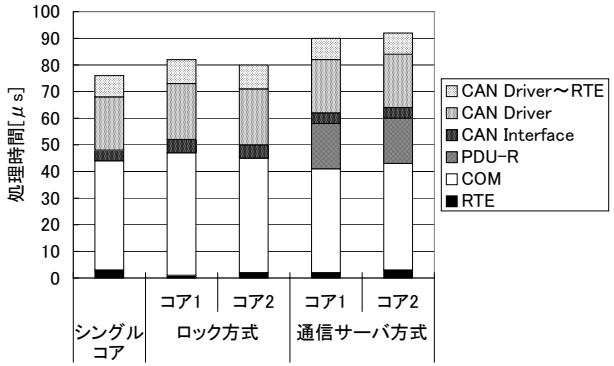

図 7 コア内排他にセマフォを利用

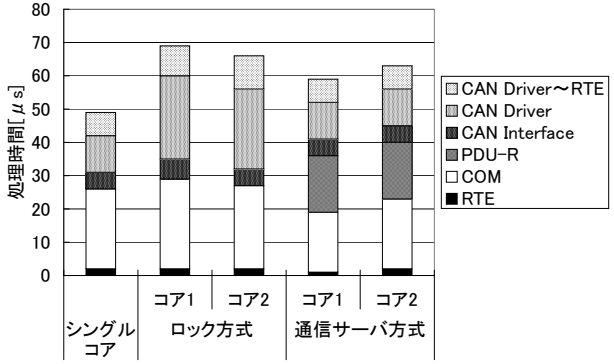

図 8 コア内排他に割込み禁止を利用

送信処理時間が長くなる．コア内排他の違いはシングルコアでの実現方法に依存することであるため，マルチコア拡張によるオーバヘッドに着目すると，ロック方式は CAN Driver でのコア間排他にセマフォを取得する分のオーバヘッドが生じる．ただし，シングルコアで排他制御をセマフォで実現するとロック方式のオーバヘッドはない．通信サーバ方式はシングルコアの実現方法に関わらず，データキューの送受信の分だけオーバヘッドが生じる．

6. ま と め

本研究では，AUTOSAR 仕様の通信ミドルウェアについて，マルチコア環境での通信をコア内通信，コア間通信，チップ外通信の 3 つに分類しそれぞれの実現方法を検討した．

コア内通信は従来の AUTOSAR 仕様で実現可能である．コア間通信については，RTE，COM，PDU-R で実現する方法が考えられるが，有用なのは RTE，COM で実現する方法であり，通信単位がシグナルと I-PDU のため一長一短であることを述べた．チップ外通信ついてロック方式と通信サーバ方式の 2 方式を提案し，ロック方式は CAN Driver でのロック取得が，通信サーバ方式ではデータキューの送受信がオーバヘッドとなることを述べた．それぞれの最大オーバヘッドは，ロック方式 41%，通信サーバ方式 29%であった．

今後の課題は，提案した方式をより多くのケースで評価を行うことが挙げられ，受信も含めた通信全体で提案方式の評価を行う必要がある．また，リアルタイム性の観点からも提案方式の評価を行う必要がある．

謝辞 本研究で用いた 10Mbps CAN は株式会社オートネットワーク技術研究所との共同研究の成果であり，オートネットワーク技術研究所の関係者の皆様に御礼申し上げます．

参 考 文 献

1) ISO 11898 : Road vehicles – Interchange of digital information – Controller area network (CAN) for high-speed communication (1993).
2) AUTOSAR, “http://www.autosar.org/”.
3) OSEK/VDX, “http://www.osek-vdx.org/”.
4) TOPPERS プロジェクト, “http://www.toppers.jp/”.
5) 倉地亮，宮下之宏，高田広章：CAN の高速化に関する研究，情報処理学会研究報告（EMB）， Vol.2007, No.121, pp.21–25 (2007).