Vol.2010-SLDM-144 No.40
Vol.2010-EMB-16 No.40
Vol.2010-MBL-53 No.40
Vol.2010-UBI-25 No.40
2010/3/27

組込み機器向け on-chip/off-chip コア間通信機構の実装と評価

三浦信一[†1] 鈴木良平[†2] 塙敏博[†1,†2]
朴泰祐[†1,†2] 佐藤三久[†1,†2]

組込み機器に用いられるプロセッサチップは，マルチコアプロセッサ技術が多く使われるようになっている．しかし，組込み機器のマルチコアプロセッサでは機能に応じた異種プロセッサコアを複数配置することも多く，共有メモリに依らないプロセッサコア間のデータ通信機構が必要である．これらのプロセッサコア間の通信をアーキテクチャに依らず記述できる，移植性の高い通信 API として MCAPI が標準化されている．本稿では，MCAPI の規格に従った新たな実装として，複数のチップで構成されたシステムにも適用できるように拡張した XMCAPI を，socket を用いて実装し性能評価を行った．

Performance Evaluation for Inter-Core Communication Interface on Inter-/Intra-Chip on Embedded Parallel Systems

Shin'ichi Miura ,[†1] Ryouhei Suzuki,[†2]
Toshihiro Hanawa ,[†1,†2] Taisuke Boku [†1,†2]
and Mitsuhisa Sato[†1,†2]

The multicore processor technology is applied to the processors for embedded systems as well as ordinary PC systems. In such multicore embedded processors, however, a processor may consist of heterogeneous CPU cores which do not configure a shared memory and require any communication mechanism for inter-core communication. MCAPI is a highly portable API for such a purpose providing inter-core communication independent from architecture heterogeneity. In this paper, we extend current MCAPI to apply to multi-chip configuration named XMCAPI, and propose its portable implementation on commodity network stack. In XMCAPI, the inter-core communication method for intra-chip cores is extended for inter-chip ones. We implement XMCAPI on standard socket named xmcapi/ip, for software development with XMCAPI.

1. はじめに

現在，組込み機器に用いられるプロセッサチップは，近年の半導体集積技術の向上や求められるデータ処理量の増加にともなって，同一チップ上に複数のプロセッサコア（以後，コア）を配置するマルチコアプロセッサ技術が使われている．

IA32 アーキテクチャに代表される汎用マルチコアプロセッサでは，チップ上にホモジニアスなコアが均一に配置され，キャッシュ一貫性制御を持った共有メモリシステムを持つのが一般的である．しかし組込み機器では，多様なサービスを提供するために，機能に応じた異種プロセッサを複数配置することも多く，また必ずしも共有キャッシュや共有メモリが提供されるとは限らない．したがって，これらのコア間で協調した動作を行う場合，コア間で何らかのデータ共有機構が必要になる．共有メモリを持つ組込み向けマルチコアプロセッサにおいては，POSIX threads（pthreads）を用いたマルチスレッド処理が利用できるが，性能最適化のためにアーキテクチャ毎にチューニングが必要であることや，資源競合に伴うロックや適切なバリア同期の管理が複雑である．一方，共有メモリを持たない場合には，各メーカが独自の API を用いて通信を記述することが多い．そのため，これらのコア間の通信をアーキテクチャに依らず記述できる，移植性の高いプログラムインタフェースが求められている．この問題を解決するために，Multicore Association[1] により，コア間の通信に特化して通信のためのオーバヘッドを削減し，メモリフットプリントを小さくした， MCAPI（Multicore Communications API）がコア間の通信 API として標準化されている[2]．

一方，処理すべきデータ量の増加から，組込み機器には，今後より一層の性能向上が要求されると考えられる．しかし組込み向けのマルチコアプロセッサにおいては，パッケージサイズや消費電力などの観点から，4 コア程度が限界であると言われている．そのため，組込み機器においてもプロセッサチップ自体を複数配置し，これらをネットワークで結合し，これらを協調動作させることで性能の向上を行うマルチプロセッサ環境になってくると考えられる．本来マルチプロセッサとはプロセッサを複数配置するもので，広義ではマルチコアプロセッサを含むものだが，本稿では同一のチップ上に複数のコアを配置したものをマルチコアプロセッサとし，複数のチップを接続した環境をマルチプロセッサ環境と定義する．これ

†1 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba
†2 筑波大学大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

Vol.2010-SLDM-144 No.40
Vol.2010-EMB-16 No.40
Vol.2010-MBL-53 No.40
Vol.2010-UBI-25 No.40
2010/3/27

らのマルチプロセッサ環境で，High Performance Computing（HPC）等の分野で培われてきた並列処理や分散処理の技術を用いることにより，処理能力の向上が期待できる．また，マルチプロセッサ技術は処理能力の向上ばかりではなく，プロセッサチップ自体を冗長に構成することで耐故障機能の向上にも役立つ．今後，処理性能や耐故障機能の向上が求められる過程において，組込み機器においてもマルチコアプロセッサかつマルチプロセッサの構成が増えていくものと考えられる．このような環境において，プロセッサチップ間でもデータ交換や同期などに通信が必要になる．それらの通信 API は，マルチコアプロセッサ内の各コア間の通信と同様に，様々な規格や実装がアーキテクチャ毎に異なり規格の統一性がない．このことから，今後マルチコアプロセッサ内部のコア間通信と同様に，組込み機器向けのマルチプロセッサ環境における，各プロセッサチップ間の通信 API の標準化が必要になる．

我々はマルチコアプロセッサ向けにチップ内のコア間通信として仕様策定された MCAPI を，異なるチップに存在するコア間の通信にも用いることを検討している．機種に依存しない統一された API が提供されることで，様々な組込み機器のソフトウェア開発や移植が容易になると考えられる．我々は MCAPI の API の仕様に基づき，新たに on-chip および off-chip のコア間通信 API として XMCAPI を提案・開発している[3]．

本稿では，我々が開発した XMCAPI 実装の概要を示し，その基本的な性能評価を行う．本稿の構成は次のとおりである．第 2 節では，マルチコアプロセッサ間の通信 API として提唱されている MCAPI を紹介する．第 3 節では本稿で提案する XMCAPI の概要について述べ，第 4 節において XMCAPI 環境を実現するために，socket によって実装される通信ライブラリ xmcapi/ip について示す．その後，第 5 節で実装した xmcapi/ip ライブリの性能評価を行う．第 6 節では XMCAPI の今後の予定について述べる．

2. MCAPI の概要

マルチコアプロセッサには，各コアで同一のメモリアドレス空間を共有する Symmetric Multiprocessor（SMP）[⋆1] と，基本的に各コアは独立に異なる処理を行うことを想定した Asymmetric Multiprocessor（AMP または ASMP）がある．SMP 構成では，System-V IPC として知られているプロセス間通信（shmem）などが使われていた．しかしシステムに依存するパラメータが多く，移植性に欠ける問題がある．現在は System-V IPC に代わって POSIX threads（pthreads）が用いられることも多いが，性能を十分に引き出すには，アーキテクチャ毎に独自の実装が必要であり，組込み機器向けの pthreads 実装では必ずしも十分な性能が得られない場合も多い[4),5)]．またプログラミングの際には複数スレッドの競合状態を防ぐためにロック操作が必要となり，プログラムを困難にする要因となる．一方，AMP 構成においては，アドレス空間は共有しているがキャッシュ一貫性制御が行われない場合や，さらに共有メモリを持たずコア毎に独立したメモリを持つ場合もある．前者はソフトウェアキャッシュにより SMP のように使用することも可能ではあるが，AMP では通常，各コア間でデータ交換のために何らかのデータの送受信が必要である．このように分散したメモリ空間では，何らかのプロセッサ間の通信（IPC: Inter-Processor Communication）を明示的に行い，データの交換を行うことが一般的である．このような IPC の代表例として，UNIX 環境で用いられる socket がある．しかし，socket はメモリの取り扱いなどで通信処理のオーバヘッドが大きく，また通信手続きが複雑であるなどの問題がある．また，HPC 向けに広く使われている並列プログラミング環境のメッセージ通信 API として MPI[6)] があるが，MPI は主として多数のノード間で高バンド幅の通信を実現するために開発されており，組込み機器向けのマルチコアプロセッサで用いるには，通信のためのオーバヘッドが大きく，またメモリ使用量が多くオフチップのメモリに対するアクセスにより性能が低下してしまう．そのため，これらの API は組込み機器に用いるマルチコアプロセッサの特性や，その内部ネットワーク構成に必ずしも適してはいない．このようなことから，組込み機器のソフトウェア開発ではハードウェア環境や目的に応じた独自の通信 API を用いて通信を記述する必要があり，プログラムの移植性や開発プロセスの複雑化といった問題があった．

このような問題を解決するために，Multicore Association[1)] によって MCAPI と呼ばれるプロセッサ間通信用の API が提唱されている[2)]．MCAPI では，マルチコアプロセッサ環境のコア間通信に特化することで，通信のためのオーバヘッドを削減し，メモリフットプリントを小さくしている．MCAPI は socket や MPI と非常に近い API の体系になっているが，MPI のような SPMD 的なプログラムモデルではなく，各々のコアでは独立な処理が行われることを想定している．**図 1** に MCAPI の全体像を示す．次に MCAPI の特徴について述べていく．

2.1 MCAPI ノードと MCAPI エンドポイント

組込み機器の中で閉じることができる 1 つネットワーク空間を MCAPI ドメインと定義する．各コアは MCAPI ノードと定義され，各 MCAPI ノードは MCAPI ドメイン内で一意に定められる *node_id* を持つ．またそれぞれの MCAPI ノードは，*port_id* で関連付けられた MCAPI エンドポイントを持つ．通常 *port_id* はネットワークインタフェースにバインドされ，

⋆1 一般的な共有メモリシステムのアーキテクチャを示す SMP とは定義が異なる．

Vol.2010-SLDM-144 No.40
Vol.2010-EMB-16 No.40
Vol.2010-MBL-53 No.40
Vol.2010-UBI-25 No.40
2010/3/27

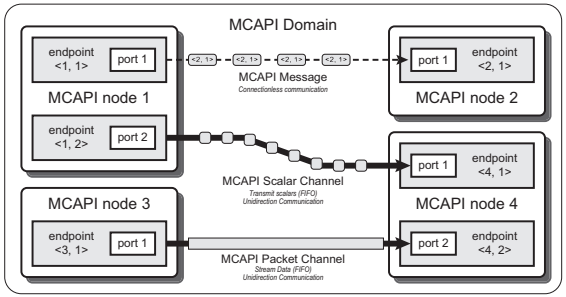

図 1 MCAPI の全体像

各 MCAPI ノードは所有する (物理 or 論理) ネットワークポート数に応じて，MCAPI エンドポイントを 1 個，もしくは複数個作成する．その結果 MCAPI ドメイン内には，<*node_id, port_id*> の組み合わせによる MCAPI エンドポイントが複数個作成される．*node_id* と *port_id* は，それぞれ TCP における IP アドレスとポート番号と等価だと考えることができる．MCAPI におけるすべての送受信処理は，MCAPI エンドポイントを送受信先として用いる．

2.2 MCAPI の通信モデル

MCAPI の通信には，Message 型と Channel 型の 2 種類がある．すべての通信は，reliable な通信路を仮定し，通信経路の状態を原因とする，パケットロスやパケットの追い越しなどは起こらない．

Message 型はコネクションレス指向の通信環境を提供し，事前の通信手続きは不要である．そのため，ユーザアプリケーションが柔軟に通信を行うことができる．ただし，送受信毎にデータの宛先として MCAPI エンドポイントを指定する必要がある．送信処理では優先度が設定可能であり，パケットの QoS を制御できる．

Channel 型はコネクション指向の FIFO 型ストリーム通信であり，Packet Channel と Scalar Channel の 2 種類が定義されている．Packet Channel では，不定長のデータに対するストリーム通信を行う．一方，Scalar Channel は，特定の整数型（TYPE: uint8，uint16，uint32 および uint64）のデータ 1 要素を送受信する．2 種類のチャンネル型通信は，コネクション指向の通信であるため，送受信を開始する前に connect，open といった事前手続きが必要である．

MCAPI によって定義されている API では，上で述べたそれぞれの通信型に対して，通信が完了するまで次の処理に進まない blocking 型と，通信と計算のオーバラップ，そしてデッドロックの防止のために，non-blocking 型をサポートする．

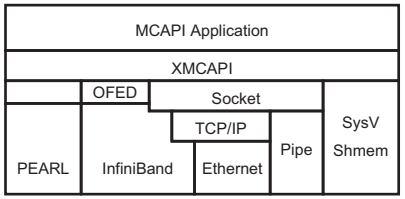

図 2 XMCAPI の通信階層

3. XMCAPI

MCAPI は，マルチコアプロセッサチップ内の IPC を目的とした API である．コア間データ転送の手段として，共有メモリやオンチップネットワークを想定して規格化されている．本来 MCAPI は，マルチコアプロセッサ内ネットワークが持つネットワークの性能を生かし，socket などの既存の通信 API と比較して極めて軽量な API である．しかし，このような特性は同一チップだけに閉じたネットワークだけでなく，複数のチップで構成されたコア間の通信に対しても必要である．この際，チップ内外によらず，同様の通信 API を用いることができれば，シームレスな通信を実現でき，チップ間に跨るプロセスの割り当てを自由に行うことができる．そこで我々は，この MCAPI を複数のマルチコアプロセッサのチップで構成されたシステムに適用することを検討し，MCAPI で規格化されたチップ内のコア間通信 API を，チップ外のコアとの間にも利用する，XMCAPI(eXtended MCAPI) を提案した[3]．

XMCAPI の実装は，下位の通信レイヤにおいて様々な物理通信インタフェース，通信プロトコルを想定する．**図 2** に XMCAPI の実装モデルについて示す． 現在公開されている MCAPI の実装は，Multicore Association が提供するリファレンス実装のみとなっている．このリファレンス実装は，共有メモリを想定し System-V shmem を用いた実装であり，共有メモリを持つ SMP 構成のマルチコアプロセッサのチップ内通信を除いて使用することができない．そこで，XMCAPI では汎用ネットワークである Ethernet を用いた実装を行う．MCAPI が reliable な通信経路を過程していることから，XMCAPI で用いるネットワークも reliable な通信経路を使用することを前提とする．たとえば，Ethernet を用いるネットワークの場合は TCP を利用することで reliable な通信を担保する．無論，共有メモリシステムを持つコア間では，現状の MCAPI のリファレンス実装のように共有メモリも利用する．

3.1 xmcapi/ip ライブラリ

XMCAPI の実装の 1 つとして，下位の通信 API として TCP socket を用いた XMCAPI の

Vol.2010-SLDM-144 No.40
Vol.2010-EMB-16 No.40
Vol.2010-MBL-53 No.40
Vol.2010-UBI-25 No.40
2010/3/27

実装を行う．本実装を xmcapi/ip ライブラリと呼ぶ．xmcapi/ip ライブラリは，標準的な通信プロトコルである TCP を用い，主に Ethernet を物理インタフェースとして用いる．ただし，一般的な TCP を用いるため，Ethernet に限らず他のネットワークにも適用可能で移植性が高い．しかし，本来 MCAPI は，socket の問題点を解決するために，通信の軽量化とインタフェースの簡略化を意図した API である．そのため，socket を利用することは，MCAPI 本来の目的からは逸脱する．ただし現実的には多くのシステムで採用されている TCP socket を用いることができれば，MCAPI がターゲットとする，共有メモリや専用ネットワークを持つ環境以外にも幅広く MCAPI アプリケーションを適用することが可能になる．また，多少の性能的な問題があったとしても，実際の製品開発時において，対象のマルチコアプロセッサの評価環境やシミュレータなどが存在しない場合においても，MCAPI アプリケーションの開発を行うことが可能になる．

4. 実　　装

本節では，現状の xmcapi/ip ライブラリの実装について説明する．現在の xmcapi/ip ライブラリは，OS として Linux 環境を想定する．実装の基本部分は UNIX 環境で標準的な socket API を用いることで，他の環境へ比較的容易に移植可能である．4.2 節に示すように，実装内部で POSIX スレッドを使用したスレッドを利用する．POSIX スレッド API のみを用いることで，特定のプロセッサアーキテクチャへの依存を避け，他のアーキテクチャへの移植を容易にする．

4.1 ノードとエンドポイントの接続管理

通信を確立するためには，それに先立ってどのホストがどの *node_id* を持つか MCAPI ドメイン内のノード ID の決定が必要になる．現状の実装では，各ノード上に *node_id* と IP アドレスに対応付けた mcapihosts ファイルを用意し，MCAPI の初期化時に mcapihosts ファイルを参照することで *node_id* から IP アドレスに変換を行い各 MCAPI ノードに接続する．各 MCAPI ノードに接続される TCP コネクションは，接続先の MCAPI ホスト毎に 1 つとする．

4.2 通信スレッド

2.2 節で述べたように，MCAPI の API 仕様による non-blocking 型を提供するために，通信はユーザアプリケーションと並行して処理する必要がある．MCAPI が本来想定するようなチップ内ネットワークの場合は，通信を DMA などのハードウェア支援によって行い，計算と通信のオーバラップを可能にする．しかし，このような機能をユーザレベルのアプリケーションで実現するためには，ユーザプログラム部とは別に，DMA の代わりとして通信専用の処理部が必要になる．また，6 節に示す，通信中継機構についても考慮する．そこで通信処理を行うスレッドを別に用意し，全ての通信はこのスレッドが統括する．本稿では通信を統括するスレッドを通信スレッド，実際のユーザアプリケーションが動作するスレッドをユーザスレッドと表す．**図 3** に，xmcapi/ip におけるユーザスレッドと通信スレッドの構成を示す．

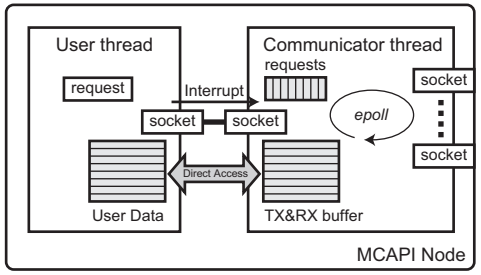

図 3　xmcapi/ip の構成

本来，軽量通信を目的とする MCAPI の実装としてマルチスレッドを用いることは，システムの大きなオーバヘッドの原因になる．特に，プロセッサコア数が処理するスレッド数を下回る場合には，資源の競合により性能低下の原因になる．しかし，xmcapi/ip ライブラリの目的は，MCAPI アプリケーションの機能検証を中心とした開発環境の構築を第一の目的とし，通信スレッド動作のオーバヘッドは問題としない．また，現在では複数のプロセッサコアを搭載したデスクトップ PC も一般的になりつつあるため，性能面からも解決できる問題と考えている．

通信スレッド内部では，Linux によって提供されているシステムコール *epoll()* を用いて，他ノードと接続されるすべての socket を監視する．ユーザスレッドとのデータ交換については，スレッド間で共有メモリを介して直接アクセスする．ただし，外部ノードとの通信のための socket とユーザスレッドとのイベント待機は両立できない．そこで，通信スレッドがユーザスレッドからのリクエストを受け付けるために，pipe を用いた socket を用意し，ユーザスレッドはこの pipe に書き込むことで通信スレッドに割込みをかける．ただし，基本的な送受信データの受け渡しは pipe を使用せず，メモリコピーを極力抑えることで，オーバヘッドを最小限にする．

4.3 データの送受信

データ送信は，ユーザスレッドが通信スレッドに送信リクエストを登録し，通信スレッド

Vol.2010-SLDM-144 No.40
Vol.2010-EMB-16 No.40
Vol.2010-MBL-53 No.40
Vol.2010-UBI-25 No.40
2010/3/27

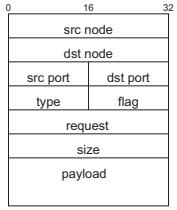

図 4 共通パケットフォーマット

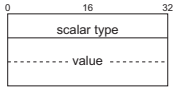

図 5 Scalar Channel のペイロード

がリクエストに応じた送信処理を行う．一方，データ受信は，各通信タイプに合わせて用意した受信バッファへ通信スレッドがデータを直接格納することで完了とする．これらのリクエストの登録や受信処理は，通信スレッド内部のスケジューラに基づき処理される．

図 4 に xmcapi/ip が通信に用いるパケットフォーマットを示す． **src/dst node** と **src/dst port** は，MCAPI アプリケーションで指定される *node_id* と *port_id* に対応する．**type** は通信の種類であり，Message や Channel，もしくはコントロールを行うための各種データタイプを表す．**flag** はパケットがデータなのか ACK なのかを示すためのものである．また，**request** を用いて，受信した ACK のイベントの内容について判断する．**size** はペイロードのサイズとなる．これらの合計 24 Byte のヘッダが，送受信のデータ毎に付加される．

次に，MCAPI の基本的な 3 種類の通信について xmcapi/ip で行った実装の概略を述べる．

Message

Message は socket における UDP 通信と同様のコネクションレスの通信である．ただし，MCAPI の仕様ではデータの到達が保障されることが前提である．xmcapi/ip では通信経路のデータ到達性は下位の通信レイヤである TCP によって保証されるが，到達したデータを格納する受信側のバッファが受信段階で常に空いているとは限らない．受信側のバッファは一定量であるので，これをすべての送信ノードで共有することになる．受信段階でバッファの空きを確実にするために，データ交換前に送信側が受信側にバッファの予約を行うランデブー型の通信方式も考えられるが，通信遅延時間の上昇が考えられる．加えて，Message に規定されている送受信のプライオリティ制御も難しくなる．

そこで，送信側は受信側の空きバッファの有無に関わらず，データを送信する．もし，受信側のデータ到着段階でバッファが一杯であった場合には，送信側に送信停止フラグを立てた ACK パケットを送信する．受信側はバッファが解放されたのちに，送信側に対して再送要求を行う．そのため，送信側は一度送信したデータは受信完了の ACK が届くまで維持し続け，また送信の終了をユーザ側に通知しない．

Packet Channel

Packet Channel は通信開始前にコネクションの確立が必要とする．コネクションの確立段階で，送受信双方でバッファを確保し，これを介して送受信を行う．一度のリクエストの最大送信データサイズは，事前に決定されているので XMCAPI 内部で受信バッファはこのサイズで分割される．受信処理は，この分割された受信バッファポインタの所有権をユーザスレッドに提供することで，受信処理は完了となり，バッファのメモリコピーは発生しない．ユーザスレッドが受信バッファポインタの所有権の解放を行うことで，受信側は新たな空きバッファを得ることになり，送信側からの新たなデータを受け入れられる．受信側は，この空いたバッファの数を window 値として送信側に返信する．送信側は空いた window 値に基づきデータを送信する．

Scalar Channel

Scalar Channel は，同一のチャンネルで 8 ∼64 bit の int 型のデータ 1 要素を送信する．基本的な送受信処理の手続きは，Packet Channel と同様だが，それぞれのデータ要素の送信毎に，それらのデータ型の情報も同時に送信しなくてはいけない．そこで，**図 5** に示す，データタイプ部 4 Byte とデータ部 8 Byte の合計 12 Byte からなるペイロードを用いてデータを送受信する．結果として，Scalar Channel の送受信は最大送信データサイズが 12 Byte の Packet Channel とほぼ同様になる．ただし，受信データのユーザデータへの受け渡しはポインタではなく 1 要素の値となる．

5. 性 能 評 価

実装した xmcapi/ip ライブラリにおいて 3 種類の通信 (Message，Packet Channel および Scalar Channel) の基本性能を評価する．

4 節に示したように，xmcapi/ip ライブラリの実装は，ユーザプログラム部分と通信部分の計 2 つのスレッドが存在するマルチスレッド動作になる．そのため，シングルコアのシステムでは，これらの 2 つのスレッドが単一プロセッサコアで動作するために資源競合が発生する．本評価では，これらの影響を比較するために，各ノードのプロセッサがシングルコア，マルチコアの 2 種類の評価を行う．プロセッサコア数以外の影響を排除するため，本来はデュアルコアであるノード環境を Linux Kernel の起動オプション (nosmp) を用いることで，シングルコアと同等のシステムにする．評価で用いるノード構成を**図 1** に示す． これらの環境を用いて，2 ノード間での通信性能を評価する．なお TCP socket の送受信バッファサイズは 256 KByte とした．

Vol.2010-SLDM-144 No.40
Vol.2010-EMB-16 No.40
Vol.2010-MBL-53 No.40
Vol.2010-UBI-25 No.40
2010/3/27

表 1 評価環境

Item	Specification
CPU	Intel(R) Xeon(R) 5110 @ 1.60GHz
Memory	DDR2-667 Dual Channel 2.0 GBytes
NIC	Broadcom 5708 Gigabit Ethernet controller
OS	Linux Kernel 2.6.27.25
Compiler	GCC 4.3

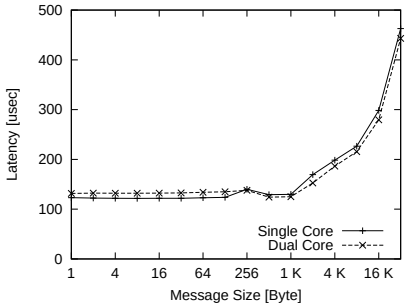

図 6 通信遅延時間 （MCAPI Message）

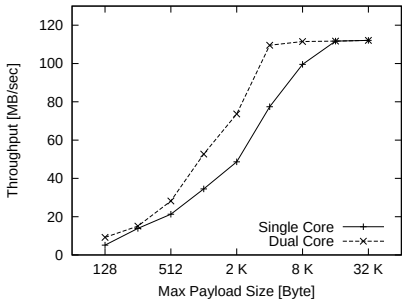

図 7 スループット （Packet Channel）

Message

メッセージサイズを変化させ，2 ノード間で 10,000 回の ping-pong 通信時間を計測する．それらの時間に基づき片方向平均通信時間を算出する．結果を**図 6** に示す．

シングルコア環境，デュアルコア環境の Message 1 Byte のデータ転送時間は，それぞれ約 123 μsec，132 μsec となり，メッセージサイズが Ethernet MTU 1.5 KByte 以下の場合でほぼ一定であった．シングルコア環境ではマルチスレッド動作によりオーバヘッドが現れると思われたが，メッセージサイズが小さい場合においてデュアルコア環境よりも良い性能を示している．本評価では基本的に通信のみであり，ユーザスレッドの処理時間の多くは通信完了の待機となる．その結果，多くの時間で通信スレッド以外は動作せず，シングルコア環境においても，プロセッサコアの資源競合は発生していない．一方で，デュアルコア環境の場合は複数のプロセッサコアがあるため，受信完了等の割込みの調停が必要になり，割込みに要する時間が長くなる．このことから，シングルコア環境と比べてデュアルコア環境のオーバヘッドが大きくなったと考えられる．しかし，実際の MCAPI アプリケーションでは計算と通信のオーバラップが発生する．そのようなアプリケーションの場合におけるシングルコア環境では，通信以外のオーバヘッドにより性能が劣化することが予想される．今後これらの条件を踏まえた，より詳細な評価が必要である．

表 2 平均送受信間隔 (Scalar Channel)

Single Core	Dual Core
33 μsec	23 μsec

Packet Channel

次に Packet Channel を用いた場合のスループットの評価を行う．評価では 2 ノード間で計 1.0 GByte のデータの送信に要した時間を計測し，スループットを算出する．この時，Packet Channel が送受信できる 1 回あたりの最大転送サイズを変化させる．結果を**図 7** に示す．

最大のスループットはシングルコア環境，デュアルコア環境で共に最大転送サイズ 32 KByte 時に約 112 MByte/sec となった．これは Gigabit Ethernet の最大性能 125 MByte/sec に対して約 90%の性能であり，十分な性能が得られている．しかし最大転送サイズが小さい場合，シングルコア環境で得られるスループットはデュアルコアよりも低い．最大転送サイズは，XMCAPI 内部の 1 回あたりの *read()* /*write()* の処理サイズになり，このサイズが小さい場合 *read()*/*write()* システムコールの回数が増加する事に加え，1 度のシステムコールでの転送サイズが低下し，XMCAPI 内部の送受信手続きとデータ送受信時間のオーバラップ部分が小さくなる．そのため，最大転送サイズが小さい場合には，XMCAPI の送受信手続きが増加するとともに，通信スレッド部の作業内容も大きくなるため，シングルコア環境においては，プロセッサコアの資源競合が発生し，スループットが低下する．しかし，最大転送サイズが 4 KByte より大きくなることで，性能の改善が得られる．このことから，1 度の転送サイズが大きいようなアプリケーションにおいては，シングルコアのような環境においても，XMCAPI の実装は有効に機能すると期待できる．

Scalar Channel

Scalar Channel を用い，2 ノード間で 100,000 要素の 64bit の整数型のデータを送受信に要する時間を計測し，送受信間隔の平均時間を算出した．結果を**表 2** に示す．

4.2 節に示したように，Scalar Channel は，最大転送サイズが 12 Byte の Packet Channel の通信とほぼ同様になる．そのため，Packet Channel と同様にシングルコア環境ではプロセッサコアの資源競合が発生し，送信間隔が長くなる．このようなサイズの転送では，ネットワークの持つ最大スループット性能よりも，遅延時間が支配的となる．しかし，要素の転送間隔が 100 μsec 程度であるアプリケーションならば，実用上問題にならないと期待できる．

Vol.2010-SLDM-144 No.40
Vol.2010-EMB-16 No.40
Vol.2010-MBL-53 No.40
Vol.2010-UBI-25 No.40
2010/3/27

6. 今後の予定

実アプリケーションでの評価

今回は，基本的な通信の性能を評価した．これらは通信に特化した評価であり，ユーザスレッドは基本的に通信の完了を待つのみとなる．しかし，実際のアプリケーションは通信と計算のオーバラップが発生し，ユーザスレッドと通信スレッドが同時に動作する．この状況の場合，今回の評価結果と異なり，シングルコア環境では大幅な性能低下が起きる可能性がある．今後，ユーザスレッドと通信スレッドが同時に動作する状況において xmcapi/ip ライブラリがどのような特性を示すか具体的なアプリケーションにより評価する．

通信中継機構の実装

xmcapi/ip ライブラリを利用することで，TCP を介して他の異なるチップ上のプロセッサコアとの通信を可能にした．しかしながら，マルチコアプロセッサの設計によっては，直接チップ外のコアと通信できないコアが存在する場合がある．これらの直接通信できないコア間では，データを中継するコアを用意することで，通信できる可能性がある．現在の MCAPI の API を用いた場合でも，ユーザアプリケーションレベルでデータ中継用の機能を実現できるが，このような実装ではユーザアプリケーションの開発が複雑になる．そこで，ユーザアプリケーション上ではなく，XMCAPI の実装内部でデータの中継を可能にし，ユーザのプログラム開発効率を向上させることを考えている．これを実現するために，XMCAPI 独自の新たな API を定義し，これによりデータ中継を実現する方法を検討する．特に，現在実装中の xmcapi/ip ライブラリなどと連携し，様々なネットワーク環境をまたがったコア間通信を実現する．

他の通信インタフェースへの適用

チップ内のコア数が増大に伴い，チップ単体のデータ処理能力が向上していることから，チップ間のネットワークバンド幅などの性能向上が必要である．加えて，組込み機器の特性より，低消費電力化や耐故障がネットワークにも求められる．既存の PC などで培われてきたネットワークは，これらの要求を満たすことは難しく，既存のネットワークに代わる，新たな標準的になるネットワークが必要である．我々は，この新たなプロセッサチップ間ネットワークとして，PCI-express をネットワークとして用いる PEARL の提案・開発を行っている[7),8)]．この PEARL の通信インタフェース用いてコア間の通信を実現するために，通信 API として XMCAPI を提供する．すでに図 2 で示したように，XMCAPI を PEARL のユーザ API としてユーザに提供することを検討している．今後，PEARL 用の XMCAPI の実装を検討し，PEARL 独自の通信仕様に合わせた，MCAPI の仕様拡張についても検討する．

7. おわりに

本稿では，プロセッサコア間通信 API として標準化されている MCAPI の新たな実装として XMCAPI の提案と実装を行った．XMCAPI は本来マルチコア内におけるコア間通信 API として規格化された MCAPI を，チップ外のコア間の通信にも適用可能な新たな実装である．XMCAPI の実装に TCP socket を用いることで，既存の Ethernet 環境を用いた通信が可能である．実装した XMCAPI を用いた基礎的な性能評価の結果，通常の socket API を直接使う場合と比較して大きなオーバヘッドなしで MCAPI を使用できることが分かった．これにより，MCAPI アプリケーションの開発に XMCAPI 環境を有効に利用できることを期待できる．

謝辞 本研究の一部は，科学技術振興機構戦略的創造研究推進事業（CREST）研究領域「実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム」，研究課題「省電力高信頼組込み並列プラットフォーム」による．

参 考 文 献

1) Multicore Association: http://www.multicore-association.org/.
2) Multicore Communications API Working Group: Multicore Communications API, http://www.multicore-association.org/workgroup/mcapi.php.
3) 三浦信一，塙 敏博，朴 泰祐，佐藤三久：組込み機器向け on-chip/off-chip コア間通信機構，情報処理学会研究報告， Vol.2009-ARC-184, No.2, pp.1–7 (2009).
4) Hotta, Y., Sato, M., Nakajima, Y. and Ojima, Y.: OpenMP Implementation and Performance on Embedded Renesas M32R Chip Multiprocessor, *Proceedings of 6th European Workshop on OpenMP (EWOMP'04)*, pp.37–42 (2004).
5) Hanawa, T., Sato, M., Lee, J., Imada, T., Kimura, H. and Boku, T.: Evaluation of Multicore Processors for Embedded Systems by Parallel Benchmark Program Using OpenMP, *5th International Workshop on OpenMP (IWOMP 2009)*, pp.15–27 (2009).
6) Message Passing Interface Forum: MPI: A Message-Passing Interface Standard (1994).
7) 塙 敏博，朴 泰祐，三浦信一，岡本高幸，佐藤三久，有本和民：ディペンダブルな組込みシステムに適した省電力高性能通信機構，情報処理学会研究報告，Vol.2007-HPC-113, pp.31–36 (2007).
8) 塙 敏博，朴 泰祐，三浦信一，佐藤三久，有本和民：小規模システム向け省電力高性能ディペンダブル通信機構:PEARL，先進的計算基盤システムシンポジウム (SACSIS2009) 論文集 (2009).