Vol.2009-ARC-184 No.2
2009/8/4

組込み機器向け on-chip/off-chip コア間通信機構

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

組込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が多く，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，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 を提案する．XMCAPI は，MCAPI で規格化されたチップ内のプロセッサコア間の通信 API を，チップ外のプロセッサコア間の通信に拡張する．また，XMCAPI を Ethernet 環境に適用するために，socket を用いた通信ライブラリとして xmcapi/ip を実装している．

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

SHIN'ICHI MIURA ,[†1] 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 Ethernet socket named xmcapi/ip, for software development with XMCAPI.

1. はじめに

近年，テレビや携帯電話，カーナビゲーションシステムなど，様々な高機能組込み機器が広く使われるようになってきた．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的な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が多く，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．しかし，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．しかし現状のマルチコアプロセッサ技術の集積能力はチップあたり 2〜6 程度のコア数である．そのため，組込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が必要あると考えている．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であるが，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

†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

境をマルチプロセッサ環境と定義する．これらのマルチプロセッサ環境で，並列処理や分散処理といった High Performance Computing（HPC）の分野で培われてきた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．しかし，マルチコアプロセッサ内の各コア間の通信と同様に，マルチプロセッサ環境の各プロセッサチップ間においても様々な実装が存在し，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，このような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ネットワークを用いた通信 API の標準化が必要である．

そこで，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 が提供されれば，様々な組込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．そこで MCAPI の API の仕様にもとづき，新たに on-chip および off-chip のコア間通信 API として XMCAPI を提案する．また XMCAPI を，既存の socket を用いた TCP 通信により実装した，基本通信ライブラリである xmcapi/ip を開発する．

本稿では，開発中の MCAPI の新規実装である XMCAPI について提案する．また，ネットワークインタフェースとして socket を採用し，下位の通信プロトコルとして TCP を用いた XMCAPI の通信ライブラリである xmcapi/ip についての概要と実装を示す．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 節において，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．また，現在は System-V IPC に代わって POSIX threads（pthreads）が用いられることが多い．しかし，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[3),4)]．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[5)] 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，またメモリ使用量も比較的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．

このような問題点を解決するために，Multicore Association[1)] によって MCAPI と呼ばれるプロセッサ間通信用のため 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．そのため，MCAPI では MPI で定義されているような HPC 向けの集合通信などは定義されていない．**図 1** に MCAPI の全体像を示す．次に MCAPI の特徴について述べる．

⋆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．

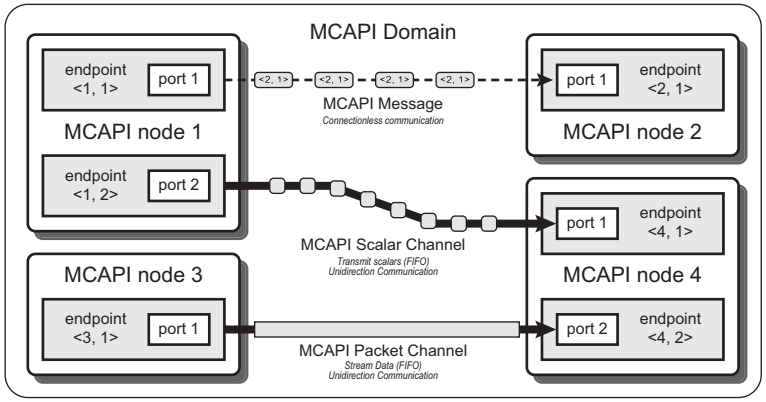

図 1 MPCAI の全体像

2.1 MCAPI ノードと MCAPI エンドポイント

組込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．この $node\_id$ の値は，*mcapi_initialize()* によってプログラムの初期化時に指定される．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，*mcapi_create_endpoint()* を用いて作成する．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，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．通信の確立には，通信先の $< node\_id, port\_id >$ の情報から *mcapi_get_endpoint()* によって取得できる MCAPI エンドポイントを用いる．$node\_id$ 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考えることができる．

2.2 MCAPI の通信モデル

MCAPI の通信には，コネクションレス指向のメッセージ型とコネクション指向のチャンネル型の 2 種類がある．2 種類の通信は共に，reliable な通信路を過程し，通信経路の状態に伴う，パケットロスやパケットの追い抜きなどは起こらないものとしている．

メッセージ型はコネクションレス指向の通信環境を提供し，コネクション指向のような事前の通信手続きが不要である．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．送受信処理はそれぞれ *mcapi_msg_send()*，*mcapi_msg_recv()* の API を用いる．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．

チャンネル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．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事前手続きが必要である．チャンネルを使用する場合は，予め *mcapi_connect_XXXchan_i()* を用いて，送受信双方のエンドポイントを接続する必要がある（XXX には pkt もしくは scl が入る）．その後，実際の送受信の前に，*mcapi_open_XXXchan_send_i()*，*mcapi_open_XXXchan_recv_i()* を用いて明示的に送信側，受信側を指定し，チャンネルを開く必要がある．その際に送受信ハンドルを取得し，以後の通信ではこの送受信ハンドルを用いてデータを送受信する．Packet Channel では，*mcapi_pktchan_send()*，*mcapi_pktchan_recv()* の 2 種類の API を用いることで，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 *mcapi_sclchan_send_TYPE()*，*mcapi_sclchan_TYPE()* の 2 種類 API を用い，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．

2.3 送受信バッファの取り扱い

MCAPI のデータ送受信では，メモリコピーを最小に抑える API 上の工夫がなされている．代表例はストリーム型の Packet Channel を用いたデータ受信である．Packet Channel を用いた受信処理では，ユーザアプリケーション独自の受信バッファを持たない．ユーザアプリケーションは，独自の受信バッファの代わりに受信処理 API が返したバッファポインタを参照することでデータを取得する．一般的なアプリケーションでは，受信データを再使用することは稀である．この一度の読込みのために，システム上の受信バッファからユーザアプリケーションの指定する受信バッファにメモリコピーすることはメモリアクセスが増えることになる．ユーザアプリケーションが直接システム上の受信バッファを参照することで，受信バッファのコピー回数を削減することができる．無論，今までと同様に受信データを保持するには，ユーザアプリケーションがそのシステム上の受信バッファを自分自身でコピーすれば良い．受信処理で返されたシステム上の受信バッファは，ユーザアプリケーションが使用後に明示的に MCAPI の API を用いて解放（返却）する必要がある．

2.4 ブロッキング処理とノンブロッキング処理

MCAPI の各種 API では基本的にブロッキング処理用もしくはノンブロッキング処理用の 2 種類が定義されている．2 種類が定義されている場合は，処理の内容に応じて使い分

ける．ノンブロッキング処理 API では，リクエストオブジェクトが返されるので，このオブジェクトを用いて処理の完了を調べることができる．完了確認のために，*mcapi_test()* や *mcapi_wait()*，*mcapi_wait_any()* などの API が用意されている．

3. XMCAPI

2 節で示した MCAPI は，マルチコアプロセッサ内の IPC を目的とした API である．コア間のデータ共有の手段として，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，チップ間を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 の規格に従った新たな実装と，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（eXtended MCAPI）を提案する．XMCAPI は MCAPI で規格化されたチップ内のコア間の通信 API を利用し，これをチップ外のコア間への通信にも利用する．

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 で新たな 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．このような XMCAPI 独自の拡張機能の詳細については現在設計中である．

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．本リファレンス実装は，共有メモリを想定した実装が古い System-V shmem を用いた実装である．MPI[5] の実装である MPICH[6] や OpenMPI[7] のように，XMCAPI では他に InfiniBand[8]

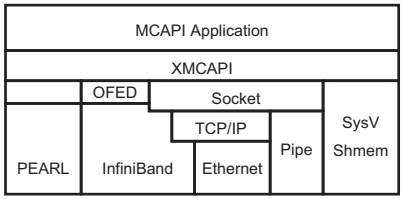

図 2 XMCAPI の通信階層

を用いた実装を検討している．これらの通信経路を用いる場合は，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 を利用することにする．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 の標準的な実装の 1 つとして，下位の通信 API として socket を用いた XMCAPI の実装を行う．本機能を実装したライブラリを xmcapi/ip と呼ぶ．xmcapi/ip ライブラリは各コア間の通信インタフェースに Ethernet を想定し，通信に用いるプロトコルとして TCP を用いる．socket を利用することで，現状多くの UNIX 環境下で動作することが期待できる．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．しかし，xmcapi/ip ライブラリは XMCAPI アプリケーションの開発環境を構築することを重視している．そのため，socket を用いることで処理は比較的重くなるが，さまざまな OS 環境において動作させることを可能にする．したがって，XMCAPI の基本実装として今回 socket を用いた TCP による通信を行うこととした．また，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した System-V shmem や，名前付きパイプを用いた socket の利用も検討する．

4. xmcapi/ip ライブラリの実装

現在の xmcapi/ip ライブラリは，ターゲット OS として Linux 環境を想定し実装している．ただし，基本的な実装の大部分は UNIX 環境で標準的な socket と pthreads を用いた実装とし，他の OS 環境へは比較的容易に移植できるように実装する．xmcapi/ip ライブラリの実装を**図 3** に示す．

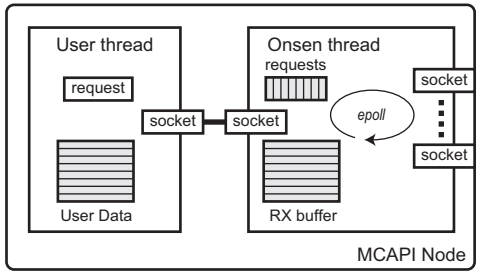

図 3 xmcapi/ip の実装

4.1 xmcapi/ip スレッド

MCAPI の API 仕様では，ノンブロッキング処理のサポートのために，ユーザアプリケーションと並行処理されるべき通信処理が必要である．これら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，アプリケーションとは別に，通信処理専用のスレッドが必要である．そこで xmcapi/ip ライブラリの実装では pthreads を用いて通信処理を統括するスレッドを用意する．本スレッドを xmcapi/ip スレッドと呼び，xmcapi/ip ライブラリ下で行われるすべて通信を担当する．また，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する．本来，軽量通信を目的とする MCAPI に pthreads を用いることは，システムに大きなオーバヘッドを与える．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であり，今回の実装では pthreads のオーバヘッドは考慮にしない．

基本的な動作は，ユーザスレッドが xmcapi/ip スレッドにリクエストを発行し，ユーザスレッドは，そのリクエストの処理完了を待つ形態とする．xmcapi/ip スレッドは該当するリクエストが終了した時点で，リクエストに設定されたコールバック関数を呼び出すことで，リクエストの完了をユーザスレッドに通知する．xmcapi/ip ライブラリの内部では，Linux によって提供されているシステムコール *epoll()* を用いて，xmcapi/ip スレッドに接続されているすべての socket を監視する．そのため，ユーザスレッドから発行されるリクエストの処理のために，ユーザスレッドと xmcapi/ip スレッド間に pipe を用いた socket を用意し，xmcapi/ip スレッドはこれを用いてユーザスレッドからのリクエストを受け付ける．ただし，基本的な送受信データの受け渡しは pipe を使用しない．それらのデータアクセスについては，スレッド間の共有メモリを介してアクセスすることで，メモリコピーを極力抑えて，オーバヘッドを最小限にする．

4.2 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，どのホストがどの $node\_id$ を持つか MCAPI ドメイン内のノードの探索が必要になる．現状の実装では多くの MPI 実装と同様に，各ノード上に $node\_id$ と IP アドレスに対応付けた hosts ファイルを用意し，その情報を用いて各 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 エンドポイント（$port\_id$）毎に 1 つとする．そのため，MCAPI ノードに複数の MCAPI エンドポイント（複数の $port\_id$）がある場合，同一の MCAPI ノード間で複数の TCP セッションが張られる場合がある．

4.3 データの送受信

通信処理の大部分は xmcapi/ip スレッド担当し，データの送信を行うユーザスレッドは，xmcapi/ip スレッドにデータ送受信のリクエストを発行することで通信処理が開始される．xmcapi/ip スレッドは適当なスケジュールに基づき，バッファの送受信処理を行う．そのため，基本的に通信はノンブロッキング通信となる．ブロッキング通信はノンブロック通信のリクエスト完了を待つことにより実現する．次に代表的な送受信の手順を示す．

送 信 処 理

(1) ユーザスレッドは socket を介して，送信リクエストを xmcapi/ip スレッドに通知する．リクエストには，バッファポインタとコールバック関数が登録されている．
(2) xmcapi/ip スレッドは，リクエストを受信し，優先度付きのキューに登録する．
(3) 受信側エンドポイントにデータ送信可能か問い合わせる．
(4) 受信側エンドポイントからの送信許可を受信後に，直接ユーザスレッド中のデータを用いて，エンドポイントにデータを送信する．
(5) 登録されたコールバック関数を用いて，ユーザスレッドに送信完了を通知する．

受 信 処 理

(1) ユーザスレッドは socket を介して，受信リクエストを xmcapi/ip スレッドに通知する．このリクエストには，受信終了時のコールバック関数が登録されている．
(2) xmcapi/ip スレッドはリクエストを優先度付きのキューに登録し，必要な受信バッファをあらかじめ確保する．
(3) 送信側エンドポイントから，送信要求が来た後に送信許可を送信する．
(4) 送信側エンドポイントから到着したデータを受信バッファに格納する．
(5) 登録されたコールバック関数を用いて，ユーザスレッドに受信完了を通知する．

現在の実装では，送受信の基本はランデブー型の通信を基本としているが，一定サイズ以下

のデータについては，送信処理（3）において，確認メッセージに送信データを付加し，同時に送信することで遅延時間を減少させることを検討している．

5. 今後の予定

現在 XMCAPI の通信ライブラリ xmcapi/ip は実装中であり，まもなく実装が完了する予定である．xmcapi/ip ライブラリは，テスト環境用の実装であるが，下位レイヤにおいて TCP を用いているために，様々な環境で使用することができる．これを生かし，通常の Ethernet 環境以外に TCP 通信が可能な他のネットワーク環境に XMCAPI を適用する．我々は，Ethernet マルチリンクを用いた高性能・耐故障ネットワークとして RI2N[9),10)] を開発してきた．RI2N/DRV は，通常の TCP がそのまま動作するため，RI2N/DRV 上で xmcapi/ip ライブラリを動作させることが可能である．これを用いることで，Ethernet を用いた場合でも高性能かつ耐故障性を持つネットワーク上に XMCAPI 環境を構築することができる．

TCP を用いた実装に加えて，プロセッサの負荷の低減や性能向上のために RDMA 機構を持つネットワークに対する実装が XMCAPI で必要になると考えている．そこで，RDMA 通信などに対応する InfiniBand に XMCAPI を対応させる予定である．今後は InfiniBand の標準的なインタフェースである OFED[11)] を用いた XMCAPI の実装を検討する．

一方で，このようなマルチプロセッサ技術を使用するためには，各コア間のネットワークが重要になる．特にマルチコアプロセッサ技術によってチップ内のコア数が増大し，チップ単体のデータ処理能力が向上している．そのため，チップ間のネットワークにバンド幅などの性能向上が求められている．加えて，組込み機器であることから，ネットワークにも低消費電力化や耐故障への対応が求められる．これらの理由から，既存の PC などで培われてきた Ethernet や InfiniBand のようなネットワークは，組込機器に適合しない場合がある．また，実装密度や耐故障機能などの点から，既存のネットワークに代わる，何らかの標準的なネットワークが必要である．本問題を解決するために，我々は新たなプロセッサチップ間ネットワークとして，PEARL の提案・開発を行っている[12),13)]．

PCI express 通信機構 PEARL

PEARL は，PCI express Generation 2 を基本技術とし，今まで，プロセッサ-デバイス間の接続にのみ用いられてきた PCI express をプロセッサ間ネットワークに拡張するネットワークである．PEARL では，PEACH（PCI express Adaptive Communication Hub）と呼ばれるコミュニケータを介してノード間を PCI express で直接接続する．**図 4** に PEARL を用いた場合の，複数プロセッサの接続例を示す． PCI express Base Spec. Rev. 2.0（以降 Gen2 と呼ぶ）

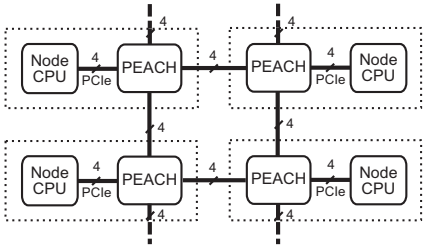

図 4 PEARL の概要

では，リンク速度は Gen1 での 2.5Gbps に加えて 5Gbps が選択できる．また複数のレーンを束ねて使用することができ（レーン数を“ x4 ”のように表記する），本数に応じて自動的に 1byte 毎にインタリーブで送信する．リンク速度と使用レーン数を動的に変更することで，通信トラフィックに応じた省電力を実現できる．また，特定のレーンにエラーが生じた場合は，そのレーンを使用しないように本数を減らして再構成することによって，正常な動作を継続することができる．また，消費電力は PCI express x4 でポート当たり数 100mW 程度を見込んでおり，他の高速ネットワークに比べるとポート当たりで数分の 1，スイッチが不要なためシステム全体ではさらに有利になる．PEARL は PCI express そのものであるため，リンクの先に SATA コントローラなどのデバイスも直接接続できる．PEACH チップにより，PEARL で接続されたノードすべてからアクセスすることが可能になる．これにより複数のノードで外部デバイスを切り替えてフェイルオーバすることも可能になり，ディペンダビリティの向上に役立つ．

PEACH チップには，PCI express の物理層コントローラ（PHY）が 4 ポート配置され，それぞれが 4 レーン分の信号伝送を行う．PCI express generation 2 では，レーン当たり 5Gbps のデータ転送が可能であるため，ポート毎に最大 20Gbps の転送レート，実際の最大バンド幅は 8b10b 符号化のため 2GB/sec となる．これらの各ポートは，一般的な PCI express のカードエッジやボード上での直接接続，もしくは PCI express の外部接続ケーブルによって接続することが可能である．PEACH チップは一種のルータチップの役割をし，PEACH チップに用意された 4 ポートのうち 1 ポートはノード CPU との接続のために使用され，残り 3 ポートを用いて隣接ノードと接続される．

PEARL が提供する通信インタフェースとして InfiniBand などで用いられる OFED の仕様を拡張し利用する予定である．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．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．今後，PEARL のユーザ API として本稿で提案した XMCAPI の適用を検討し，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．

6. おわりに

本稿では，本来マルチコアプロセッサ内部のプロセッサコア間通信 API として標準化された，MCAPI の実装として XMCAPI を提案した．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り，様々なネットワークインタフェースに対応する予定である．本稿では，まず XMCAPI に対応するネットワークとして，既存の Ethernet 環境に使用可能な socket を用いた実装である xmcapi/ip ライブラリの概要について示した．今後，xmcapi/ip ライブラリの実装を進め，実際に MCAPI アプリケーションなどを動作させ評価を行う予定である．

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) 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).
4) 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 (IWOPM 2009)*, pp.15–27 (2009).
5) Message Passing Interface Forum: MPI: A Message-Passing Interface Standard (1994).
6) Gropp, W., Lusk, E., Doss, N., and Skjellum, A.: A high-performance, portable implementation of the MPI Message-Passing Interface standard, *Parallel Computing*, Vol.22, No.6, pp. 789–828 (1996).
7) Gabriel, E. et al.: Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation, *PVM/MPI*, pp.97–104 (2004).
8) InfiniBand Trade Association: InfiniBand, http://www.infinibandta.org/.
9) 岡本高幸，三浦信一，朴　泰祐，塙　敏博，佐藤三久：ユーザ透過に利用可能な高性能・耐故障マルチリンク Ethernet 結合システム，情報処理学会論文誌. コンピューティングシステム， Vol.1, No.1, pp.12–27 (2008).
10) Miura, S., Hanawa, T., Yonemoto, T., Boku, T. and Sato, M.: RI2N/DRV: Multi-link Ethernet for High-Bandwidth and Fault-Tolerant Network on PC Clusters, *Proceedings of Workshop on Communication Architecture for Clusters (CAC2009) in IPDPS 2009* (2009).
11) The OpenFabrics: OFED, http://www.openfabrics.org/.
12) 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 2007-HPC-113, Vol.2007, No.122, pp.31–36 (2007).
13) 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).