Vol.2014-HPC-143 No.8
2014/3/3

エクサスケール通信向け ACP スタックの設計思想

住元 真司[1]　安島 雄一郎[1]　佐賀 一繁[1]　野瀬 貴史[1]　三浦 健一[1]　南里 豪志[2]

概要：エクサスケールシステムでは、極限までの通信遅延の削減を省メモリに実現する必要がある。本論文では、エクサスケール通信をめざして開発中の ACP スタックの設計についての設計方針と ACP スタックの概要について述べる。

Design Policy of ACP Stacks for Exa-Scale Communication

Abstract: In communication on exascale system, low latency communication with reduction of memory usage is required. This paper discusses design policy of ACP stacks and overview for communication of exascale system.

1. はじめに

エクサスケール規模のシステムでは、電力対性能の制約が厳しいため総メモリ量の性能比での増加が見込めなくなる。しかし、現状の通信ライブラリは、並列プロセス数が増加した場合に、送受信バッファや通信制御用メモリが比例増加する問題がある [1]。さらに、エクサスケール規模のシステムでは System on Chip が主流になり、インターコネクトデバイスもプロセッサと同じチップに実装され、ハードウェア遅延も、より小さくなる。

このため、エクサスケール規模のシステムでは、より低遅延で省メモリ性を確保した通信ライブラリの実現が課題である。我々は、この課題を解決する通信ライブラリの開発を ACE(Advance Communication for Exa) プロジェクトで実施している。

本論文では、以上の課題を解決しエクサスケール規模での通信を実現する ACP(Advanced Communication Primitives) スタックの設計思想について述べる。

本論文の構成について述べる。第 2 章で、既存通信ライブラリのメモリ使用量解析について述べ既存通信ライブラリのメモリ使用の課題を整理する。これを受け、第 3 章でアプローチの指針を示す。第 4 章で、課題を解決するために開発する ACP スタックの狙いと設計思想を述べ、第 5 章で ACP スタックの概要について述べる。

[1] 富士通/JST-CREST
川崎市中原区上小田中 4-1-1
[2] 九州大学/JST-CREST
福岡市東区箱崎 6-10-1

2. 既存通信ライブラリのメモリ使用量解析とエクサスケールに向けた課題

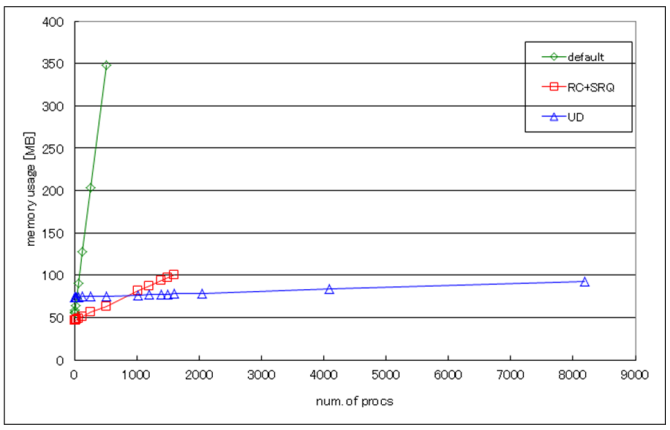

図 1　Open MPI 1.4.5 におけるメモリ使用量

エクサスケール規模のシステムでは、総メモリ量の性能比での増加が見込めない。しかし、現状の通信ライブラリは、並列プロセス数が増加すると、送受信バッファや通信管理制御用バッファが増加する。実際に既存の MPI ライブラリである Open MPI[2] を調べた結果を図 1 に示す [3]。

図 1 を外挿し InfiniBand 上でプロセス数を増加させた場合のメモリ使用量を見積もった結果を表 1 に示す。表 1 よ

表 1 Open MPI におけるノード数によるメモリ使用見積り

# of Node	RC/RQ	RC/SRQ	UD
100,000	56.23 GB	3.33 GB	0.29 GB
1,000,000	561.87 GB	32.86 GB	2.24 GB
10,000,000	5,618.22 GB	328.17 GB	21.75 GB

り、もっともメモリ使用量の少ない UD(Un-reliable Datagram) 通信の場合でも、1000 万プロセス時に 21.75GB ものメモリがプロセスあたりに必要であることがわかった。さらに、UD は高性能なデータ転送である遠隔メモリアクセス (RMA) が利用できない。一方、RMA が利用可能な RC(Reliable Connection)+SRQ の場合は、328.17GB ものメモリが 1 プロセスあたりに必要である。エクサスケール規模に対応した MPI ライブラリを実現するには大規模プロセス実行時のメモリ使用量の抜本的な削減が必要である。

本章では、ACP スタック開発の背景として、MPI ライブラリの使用量メモリ調査を通して、エクサスケール時代の通信ライブラリの省メモリ化の課題について述べる。

2.1 DMATP-MPI による MPI ライブラリのメモリ使用量調査

現状の MPI 通信ライブラリが、どのようにメモリを利用しているのかを調べるために、DMATP-MPI というメモリ使用量解析ツールを開発し [4]、Open MPI のメモリ使用量について調査した。

DMATP-MPI による調査の結果、以下のことがわかった。[1]

(1) Unexpected Message 用の受信バッファメモリは、受信した Unexpected Message に必要なだけ制限なく割り当てられ、一度割り当てられるとプログラム終了まで開放されない。プログラムにより GB クラスものメモリ使用量となる。

(2) プロセス数が増加すると集団通信 (MPI_Alltoall 関数等) に比べ、500 ノード以上で MPI_Init 関数呼び出し時の方が多くのメモリが割り当てられ、そのメモリ量はプロセス数に比例して増加している。(図 2)

(3) 集団通信関数で利用される通信バッファは集団通信のアルゴリズムに依存するが、ノード数に対して対数オーダーでの増加である。

MPI のメモリ使用を整理した結果、MPI が割り当てるメモリは、表 2 のように分類されることがわかった [1]。

以下に、表 2 の中で制御用バッファに関連する Open MPI 内の関数毎のメモリ使用量の調査結果について述べる。調査では、PC クラスタ上でインタコネクトとして InfiniBand と tcp(socket) を用いた場合について測定した。ここでは、プロセス数の変化に対して特徴的なものを取り上げる。なお、各測定結果のグラフ中各プロット中にある数値 (A/B) はそれぞれ malloc 関数を呼び出した回数 (A)、mfree 関数を呼び出した回数 (B) である。

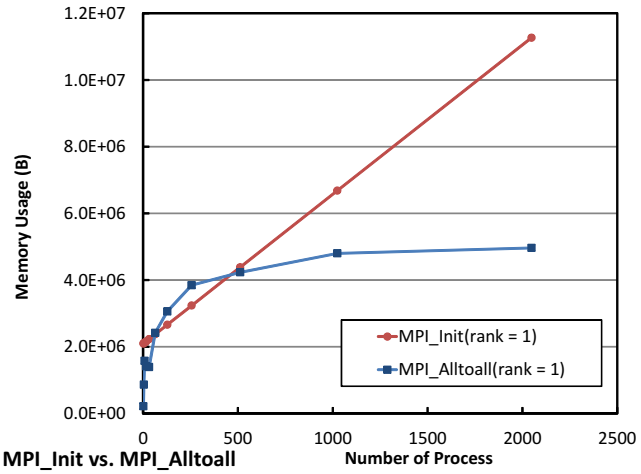

図 2 MPI_Alltoall と MPI_Init のメモリ使用量比較

デバイス依存＆制御用バッファ： インタコネクトの種類とネットワークインターフェイス数によりメモリ使用量が異なる点からデバイス依存であることがわかる。

(1) 図 3 に MCA_PML_CALL(add_procs()) 関数のプロセス数を増加させた場合のメモリ増加量を測定した結果を示す。このメモリ増加量とは、関数を呼び出すことにより増加したメモリ使用量と定義する。 図 3 より、メモリ増加量がプロセス数に比例して増加している点、プロセス数が増加する毎に、malloc 関数の呼び出し回数が増加している点が上げられる。ただし、インタコネクトの違いによる malloc 関数の呼び出し回数は同じである。これらの結果より、プロセス毎にデータ構造を割り当てていると推測できる。

(2) 図 4 に ompi_proc_set_arch() 関数のプロセス数を増加させた場合のメモリ増加量を測定した結果を述べる。 図 4 より、MCA_PML_CALL(add_procs()) 関数と同様な傾向を示している。ただし、インタコネクト種類が増えると malloc 関数の呼び出し回数とメモリ増加量が増加している点が異なる。これらの結果から、プロセスとインタコネクト種類毎にデータ構造を割り当てていると推測できる。

デバイス非依存＆制御用バッファ： インタコネクトの種類とネットワークインターフェイス数によりメモリ増加量に違いがないことからデバイス非依存であることがわかる。

(1) 図 5 に ompi_proc_init() 関数のプロセス数を増加させた場合のメモリ増加量を測定した結果を述べる。 図 5 より、プロセス数が増加すると、malloc 関数の呼び出し回数が比例して増加する点が上げられる。これらの結果から、プロセス数に比例し

表 2 Open MPI におけるメモリ利用分類

	デバイス依存	デバイス非依存
通信データ用バッファ	デバイスバッファ	集団通信向け, Unexpacted Message 等
管理制御用バッファ	制御用構造, コマンドキュー, 完了キュー等	Communicator, Tag match Table 等

た回数のデータ構造を割り当てていると推測できる。

(2) 図 6 に MCA_PML_CALL(add_comm()) 関数のプロセス数を増加させた場合のメモリ増加量を測定した結果を述べる。 図 6 より、プロセス数が増加しても、malloc 関数の呼び出し回数に変化がない点が上げられる。これらの結果から、プロセス数に比例した大きさの固定データ構造を割り当てていると推測できる。

(3) 図 7 に ompi_comm_init()) 関数のプロセス数を増加させた場合のメモリ増加量を測定した結果を述べる。 図 7 より、インタコネクトの種類と数によりメモリ増加量に違いがなく増加量が増加している点、プロセス数が増加しても、malloc 関数の呼び出し回数に変化がない点が上げられる。これらの結果から、プロセス数に対応した（例えば木構造）大きさの固定データ構造を割り当てていると推測できる。

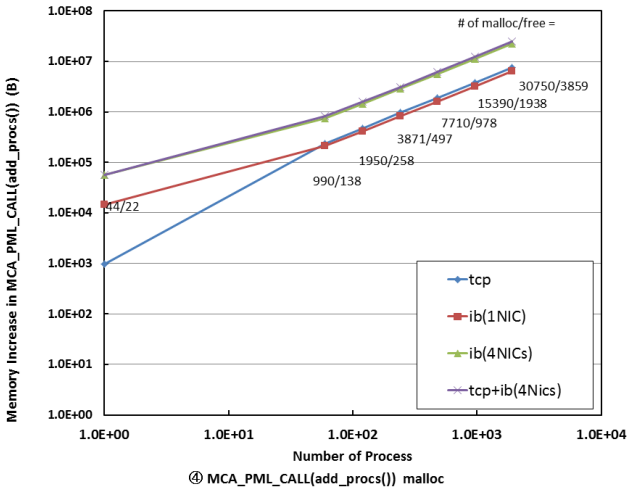

図 3 プロセス数の変化による MCA_PML_CALL(add_procs()) のメモリ使用量 (MPI_Init 関数内)

2.2 エクサスケールに向けた省メモリ化の課題

本節では第 2.1 節での MPI ライブラリのメモリ使用量調査結果を元に、エクサスケールのシステムに置ける通信ライブラリの課題を以下の点から整理する。

(1) Unexpected Message 処理

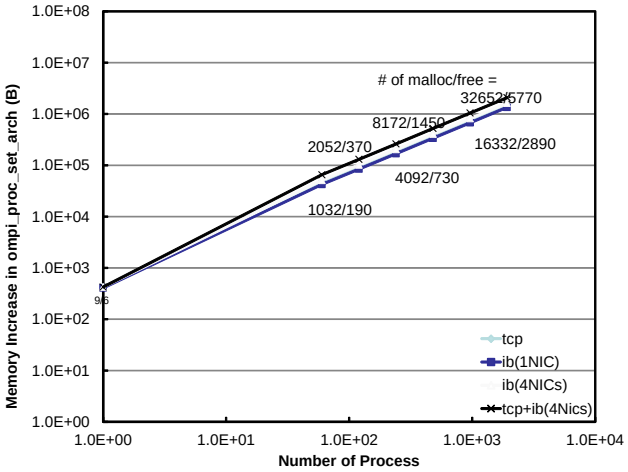

図 4 プロセス数の変化による ompi_proc_set_arch() のメモリ使用量 (MPI_Init 関数内)

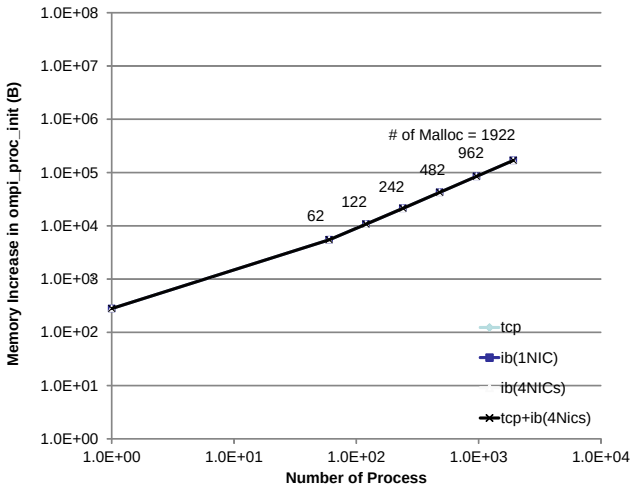

図 5 プロセス数の変化による ompi_proc_init() のメモリ使用量 (MPI_Init 関数内)

(2) 通信用 と 管理制御用　のバッファ

(3) デバイス非依存 と デバイス依存　のバッファ

2.2.1 Unexpected Message 処理

MPI で定義しているメッセージ通信の実装プロトコルとしては、Eager 通信とランデブ通信がある。この中で、Eager 通信は通信先で MPI_recv() が発行されているかどうかに関係になく、メッセージを送ることができる。この時、相手先の MPI_recv() が発行されていない場合に、送られたメッセージに対し、対応する受信バッファが存在しない状態となる。この状態のメッセージを Unexpected Message と呼ぶ。

この状態になった場合、メッセージ廃棄し再送するよりも、新たに受信バッファを確保し蓄えておき、MPI_recv() が発行された時に、蓄えたバッファからコピーすることで、通信遅延を減らすことができる。この処理を Unexpected

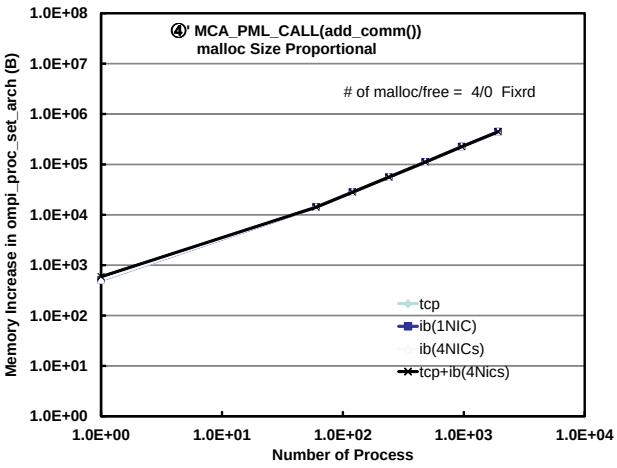

図 6 プロセス数の変化による MCA_PML_CALL(add_comm()) のメモリ使用量 (MPI_Init 関数内)

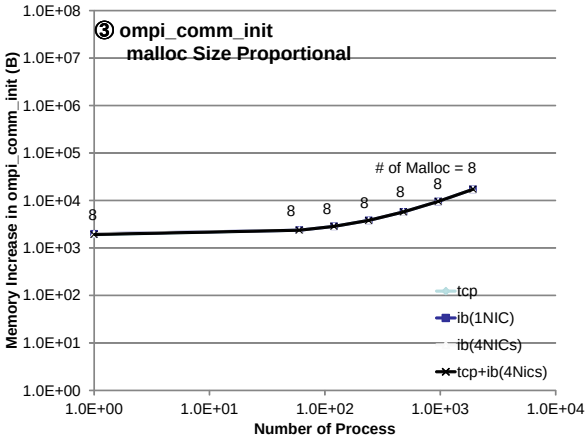

図 7 プロセス数の変化による ompi_comm_init() のメモリ使用量 (MPI_Init 関数内)

Message 処理と呼ぶことにする。

Unexpected Message 処理の仕組みは通信遅延を抑制する上で有効であるが、省メモリという観点からは、Unexpected Message 処理に利用する受信バッファメモリに制約を加えるべきである。しかし、受信先にどれだけ空きメモリ余裕があるかは、プログラム次第であるので MPI ライブラリ実装での制御は難しい。例えば、無理に一定量以下に Unexpected Message 処理に使うメモリ量を抑えた場合、通信遅延のバラツキが発生しえる。これは、通信性能自体も予測不能になるために回避すべきである。この問題を回避するためには、アプリケーションを含めて通信バッファを最小限に抑制しながら、通信遅延を抑える仕組みが必要である。

2.2.2 通信用と管理制御用のバッファ

通信用バッファ

通信用のバッファについては、既に Open MPI や MPICH[5] の実装は必要な場合にのみ、バッファを割り当てている、このため、バッファ資源を無駄に使わない実装になっている。しかし、アプリケーションレベルで多数の通信先と通信した場合に、使用メモリ量を抑制するのは容易ではない。

「京」の MPI では、この課題を高速モードと省メモリモードという 2 つのモードを用意して対応している [6]。初期化時においては、通信バッファは割り当てられず、最初の通信時に省メモリモードとなり、一定頻度通信した通信先に先着順で一定数まで高速通信モードに移行することにより、一定量のメモリ使用に抑制している。　しかし、この方式では、一定の高速モード数に達した後に性能が要求される通信先があった場合には対応できず、常時、通信頻度を確認し、高速モードと省メモリモードの移行を動的に行う等の仕組みが必要になるが、２つのモードの移行に伴う、通信性能の変化はアプリケーションに影響を及ぼすため問題となる。

本来は、アプリケーション毎に高速性が要求される通信先を指定できる等、アプリケーションが通信ライブラリと連携できる仕組みを提供すべきである。

管理制御用バッファ

管理制御用バッファの省メモリ化については、既存の MPI ライブラリでは、考慮されておらず、プロセス数分のメモリを確保している。このため、省メモリ化については、各プロセス上に分散して配置するのが適切である。そして、あるプロセス上には、必要で優先度の高い情報のみを管理制御用のバッファに格納し、必要な時に他のプロセスから獲得するように変更すべきである。

このような場合、第 2.1 節で整理したように、既存の MPI 実装では、配列とリスト結合によるデータ構造が多く採用されている。これらのデータ構造を容易に表現し、操作できる機能を持つことが望ましい。

2.2.3 デバイス非依存とデバイス依存のバッファ

デバイス非依存バッファ

デバイス非依存バッファの省メモリ化については、他のプロセス上への分散配置化が可能であるため、第 2.2.2 節で、述べたように必要かつ優先度の高いデータを他のプロセスから獲得しローカルなメモリに配置すべきである。

デバイス依存バッファ

デバイス依存バッファの省メモリ化については、例えば、InfiniBand の Queue Pair(QP) 資源など、割当てしないことにより通信機能に影響がある場合があるため、通信リソースの動的な管理を含めて、デバイス毎にきめ細やかなメモリ使用管理を考える必要がある。

3. エクサスケール向け通信ライブラリ実現のアプローチ

第 2 章で述べた課題についてエクサスケールシステム向けの通信ライブラリ実現の観点から整理し、エクサスケール向け通信ライブラリ実現のためには、どうするべきかについて述べる。

Unexpected Message 処理の問題： MPI 仕様には送受信可能なバッファ量など、計算機資源量の規程がない。このため、ユーザがどのようにもプログラムを書くことができる。しかし、プログラムの書き方により使用メモリ量が大きく異なり、性能においてもバラついてしまう。エクサスケールシステム向けの通信ライブラリは、使用メモリ量を一定以下に抑制しながら、低遅延かつ高バンド幅通信を書くことができるプログラムインターフェイスを提供すべきである。

全プロセス情報の管理： プロセス数に比例して使用量が増える管理制御バッファの中で他のプロセス情報をプロセスの数だけメモリ上に保持しているデータがある。このようなデータは、それぞれのプロセス上に確保しておき、必要な時に利用頻度の高い順に局所的に確保すればよいものである。しかし、データ間の依存関係がありリスト構造で管理されている場合、データを複数のプロセス間で分散して置くと依存関係の変更が問題である。グローバルにデータ構造を扱えるようにすべきである。

アクティブな通信先の把握： プロセス数に比例して必要な管理制御バッファを減らすには、実際に通信しているプロセスにだけ、バッファを割り当てるのが自然である。しかし、MPI の API 仕様では、どのプロセスがどの相手先とどのくらい長く通信するのかは、アプリケーションが通信しないとわからない。このため、既存の MPI ライブラリは、MPI の通信関数の呼び出しから把握する以外に手段はない。しかし、限られたリソースをリソース以上の要求から限定するのは、容易でない。効果的にリソースを活用するためにはアプリケーションが直接リソース配分をすべきである。

以上より、エクサスケールシステム向けの通信ライブラリは、使用するメモリの用途と量を明確に意識し、グローバルにデータ構造を扱うことができるものにすべきである。

4. ACP スタック開発の狙いと設計思想

ACP スタックの開発の狙いは、エクサスケール規模の通信を、省メモリ、かつ、低遅延で実現可能な通信スタックを提供することにある。開発する通信スタックは、その上に様々なプログラムが動作することが重要であるため、より多くの種類のインターコネクト上で稼働すべきである。

エクサスケール時代に相応しい通信ライブラリを設計する上では、通信の基本機能を根本的に見直すことが必要であると考えている。必要とされる基本機能をインタコネクトデバイスに最小限の抽象化を施し提供した上で、この基本機能の上に必要な機能を順次積み上げていく。なぜなら、省メモリ性確保も低遅延性の実現もインタコネクトデバイスレベルから、きちんとボトムアップ的に機能を積み上げていかないと実現できないからである。こうすることでエクサスケール時代に必要とされる機能と性能を実現する。

これまで述べてきたように、エクサスケール時代の通信を実現するには、省メモリ性の確保と低遅延でのデータ交換が必須である。このため、以下の設計思想で ACP スタックを設計する。

- 省メモリ性の確保においては、既存の通信ライブラリのように暗黙的に省メモリを実現するプローチでは限界がある。だから、ACP においては、メモリ使用を明示的に制御するインターフェイスとする。
- 低遅延のデータ交換の実現においては、転送遅延が短い他、プロセッサを介在しないデータ交換が重要である。このため、InfiniBand[7] や Tofu インタコネクト [8], [9] には、インタコネクトの機能として RMA 機能が実装されている。故に、インタコネクトデバイスが提供する RMA 機能の抽象化を基本として ACP を設計する。

5. ACP スタックの概要

我々は、エクサスケール規模のシステムにおいて、通信遅延の極小化、使用メモリ量の極小化が必須になると考え、これを実現するために ACP スタック開発を進めている [10], [11]。本章では、ACP スタックの概要について述べる。

ACP スタックは、基本層とチャネルインターフェイスとグローバルデータ構造ライブラリから構成される。以下は ACP スタックの概要である。

- 1000 万並列プロセスで動作、プロセス数に比例しないメモリ使用を実現
- インタコネクトは、UDP, InfuniBand, Tofu をターゲットとし、アプリケーションの移植性を確保
- 基本機能として、グローバルメモリ参照 (GMA) とグローバルメモリ管理 (GMM) を提供する。[10]
- 基本機能上にストリーム転送の使用メモリを最適化するためのチャネルインタフェースと、大域的なデータ配置を最適化するためのグローバルデータ構造ライブラリ機能の提供を検討している。

以降、ACP 基本層、ACP チャネルインターフェイスとグローバルデータ構造ライブラリの概要について述べる。

5.1 ACP 基本層

ACP 基本層は、第 3 で述べたグローバルにデータ構造

を扱うための基本機能の実現と、第 4 章で述べた RMA 機能の抽象化を担うために、グローバルメモリ参照、グローバルメモリ管理機能、ならびにインフラストラクチャに必要な機能を提供する。

GMA: 送信元でも宛先でもないプロセスが、プロセス間のメモリ間コピーとデータ制御を実行することができる。GMA を使うことで、プロセスは、グローバルメモリ上のデータが実際にどのプロセスにあるのかを意識することなくプログラミング可能である。

GMM: 各プロセスの論理アドレスをグローバルアドレスに変換する機能を提供する。

通信インフラストラクチャ: 通信の初期化、終了、再初期化、プロセス同期などの機能を提供する。

GMA と GMM により、ACP 基本層は必要な時に必要な領域だけを非同期にグローバルメモリとして登録しアクセスできる機能 [12] を提供する。API の詳細と実装については、文献 [10], [11] を参照されたい。

5.2 チャネルインターフェイスとグローバルデータ構造ライブラリ

ACP は、基本層と共にチャネルインターフェイスとグローバルデータ構造ライブラリから構成される。チャネルインタフェースは、ストリーム転送の使用メモリを最適化するためのインターフェイスであり、グローバルデータ構造ライブラリは、大域的なデータ配置を最適化するためのライブラリコレクション機能である。

5.2.1 チャネルインタフェース

チャネルインタフェースは、ACP の設計方針である「メモリ使用を明示的に制御する」と言う考え方に基づき、ACP 基本層上に送信側と受信側を接続するチャネルを介したストリーム転送を提供する。このストリーム転送は、送信したデータが確実に送信順の通りに受信側に届くことを保証した通信を意味する。

チャネルインターフェイスの一つの利用法として、グラフにより記述されたパイプライン転送がある。複数ノード間でデータが流れるプロセス順を決めておき、各ノードでは到着したデータを処理し、次の転送先に転送する。

パイプライン転送を省メモリの観点から見た場合、ノード間のデータ転送方向は片方向となることが多いため、チャネルは片方向転送機能を提供する。両方向転送については、反対方向のチャネルを設定することにより実現される。なお、チャネルは、プログラマによって必要時に確保され、不要になった時に解放される。

以上述べたように、チャネルインターフェイスは、チャネルの獲得と解放をプログラマに明示させることで、メモリ使用を明確に意識したストリーム転送を提供する。

5.2.2 グローバルデータ構造ライブラリ

グローバルデータ構造ライブラリは、グローバルメモリ上での低消費メモリながら、大域的なデータ配置を最適化し、データ転送のオーバヘッドが小さいプロトコルとアルゴリズムを提供する。

6. まとめ

本論文では、より低遅延で省メモリ性を確保したエクサスケール規模の通信を実現するための低レベル通信スタックである ACP スタックの設計思想について述べた。

エクサスケール時代の通信を実現するためには、省メモリ性の確保と低遅延でのデータ交換が必須である。このため、ACP の設計では、メモリ使用を明示的に制御するインターフェイスを定義することにした。また、低遅延のデータ交換を実現するために、エクサスケール時代のインタコネクトデバイスが提供する RMA 機能の抽象化を基本とした。

さらに、効果的なスタック機能の実現のために、ACP のスタックは基本層とチャネルインターフェイスとグローバルデータ構造ライブラリから構成される。基本層では、GMA と GMM を提供し、基本層の上にストリーム転送の使用メモリを最適化するためのチャネルインタフェースと、大域的なデータ配置を最適化するためのグローバルデータ構造ライブラリ機能が実現される。

現在、ACP 基本層の設計と実装を進めており、今後、基本層の評価とチャネルインターフェイスとグローバルデータ構造ライブラリの設計を進めていく予定である。

謝辞 本研究は、科学技術振興機構 戦略的創造研究推進事業（CREST）「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」研究領域、「省メモリ技術と動的最適化技術によるスケーラブル通信ライブラリの開発」の一部として実施された。

参考文献

[1] 住元真司, 秋元秀行, 安島雄一郎, 安達知也, 岡本高幸, 三浦健一. DMATP-MPI を用いた MPI ライブラリの関数別メモリ使用量評価. 情報処理学会研究報告 13-HPC-138(14). 情報処理学会, Feb. 2013.

[2] OpenMPI: http://www.open-mpi.org/.

[3] 三浦健一, 秋元秀行, 安島雄一郎, 岡本高幸, 住元真司. エクサスケールコンピューティングに向けた省メモリ通信ライブラリの検討. 情報処理学会研究報告 12-HPC-133(14). 情報処理学会, Mar. 2012.

[4] 秋元秀行, 安島雄一郎, 安達知也, 岡本高幸, 三浦健一, 住元真司. DMATP-MPI: MPI 向け動的メモリ割当分析ツール. 情報処理学会研究報告 13-HPC-138(14). 情報処理学会, Feb. 2013.

[5] MPICH2: http://www.mcs.anl.gov/research/projects/mpich2/.

[6] 住元真司, 川島崇裕, 志田直之, 岡本高幸, 三浦健一, 宇野篤也, 黒川原佳, 庄司 文由, 横川三津夫. 「京」のための MPI 通信機構の設計. SACSIS 2012 - 先進的計算基盤システムシンポジウム. 情報処理学会, May 2012.

[7] InfiniBand Trade Association:

http://www.infinibandta.org/.

[8] Yuichiro Ajima, Shinji Sumimoto, and Toshiyuki Shimizu. Tofu: A 6d mesh/torus interconnect for exascale computers. In *IEEE Computer*, pp. 36–40, Nov. 2009.

[9] Yuichiro Ajima, Yuzo Takagi, Tomohiro Inoue, Shinya Hiramoto, and Toshiyuki Shimizu. The tofu interconnect. In *Hot Interconnects*, pp. 87–94, 2011.

[10] 安島雄一郎, 佐賀一繁, 野瀬貴史, 三浦健一, 住元真司. ACP 基本層のインタフェース. 情報処理学会研究報告 14-HPC-143(9). 情報処理学会, Mar. 2014.

[11] 佐賀一繁, 安島雄一郎, 野瀬貴史, 三浦健一, 住元真司. ACP 基本層の実装と初期評価. 情報処理学会研究報告 14-HPC-143(10). 情報処理学会, Mar. 2014.

[12] 安島雄一郎, 秋元秀行, 岡本高幸, 三浦健一, 住元真司. 非同期グローバルヒープの提案と初期検討. 情報処理学会研究報告 13-HPC-138(10). 情報処理学会, Feb. 2013.