

通信ライブラリ UCX の Tofu-D 対応の検討

渡部 裕^{1,a)} 佐藤 三久^{3,2} 辻 美和子³ 村井 均³ 朴 泰祐²

概要: 本稿では、抽象化された通信ライブラリである UCX を、富岳などで利用される A64FX CPU に統合された通信インターフェースである Tofu-D に対応するための検討を行う。Tofu-D インターフェース上で IP 通信が利用可能であることを利用し、予備評価として UCX の TCP モードで性能を測定した結果、TNI 1 つあたりの理論性能が 6.8GB/s であるのに対し実測値は約 175MB/s であった。よって Tofu-D インターフェースの性能を最大限に利用するためには UCX に uTofu を用いた実装を追加する必要があることを確認した。また、実装を進めている Tofu-D 対応 UCX について、UCT API を用いた PUT 通信の性能測定を行い、uTofu で直接記述した場合とほぼ同等の性能を得られることを確認した。

1. はじめに

高性能計算システムのノード間通信の技術や製品として、InfiniBand, OmniPath, Aries などの多種多様な通信インターフェースが開発され利用されている。これらのインターフェースは独自のユーザ API を持つため、通信を伴うソフトウェアを記述する際には、それぞれを個別に記述する必要がある。そのようなユーザの負担を軽減し統一的な記述を可能とするために UCX [1,2] などの抽象化された通信ライブラリが開発された。UCX においては、現在では OpenMPI や OpenSHMEM など様々なソフトウェアの下位の通信レイヤとして利用されている。

2021 年 3 月に一般運用が開始された理化学研究所のスーパーコンピュータ「富岳」[3] では、富士通製 CPU「A64FX」に統合された Tofu-D インターフェースを用いて計算ノード間の通信を行う。Tofu-D を利用するためのユーザレベル API として uTofu が提供されているが、現在 UCX は uTofu をサポートしておらず、UCX を用いて記述されたソフトウェアのエコシステムを十分に活用することはできない。そこで本稿では、UCX で記述された、あるいは今後開発されるソフトウェアに変更を加えることなく富岳等で利用可能とするため、UCX の Tofu-D インターフェース対応を行うことを検討した。

また、近年は多数のコアを搭載したメニーコアプロセッサが広く利用されるようになり、従来利用されてきた大規模ループを複数のコアで分割して実行するようなデー

タ並列モデルだけでなく、実行の最小単位をタスクとし動的に複数のコアで実行するタスク並列実行モデルにも注目が集まっている。我々は、複数のノードにまたがる分散タスク並列モデルにおいて、生成されたタスクグラフのデータの依存性に応じ、PGAS の片方向通信を利用してノード間のデータ依存性を解決しながらタスクの実行を行う、マルチタスク PGAS モデルを提案している。複数のノードでタスク並列を実行する際、タスクは別々のスレッドで実行されることがあるため、スレッドごとにノード間通信を行うことが必要となる。ノード間通信に利用される代表的なモデルは MPI であるが、Balaji らの研究 [4] により、MPI_THREAD_MULTIPLE を利用して各スレッドが自由に通信を行う場合、スレッド数の増加に応じて通信性能が低下することが指摘されている。また、MPI_THREAD_MULTIPLE がサポートされていたとしても、MPI のランクごとの通信エンドポイントやタグマッチなどの MPI の通信モデルをサポートするために、スレッド間の排他制御のオーバーヘッドが大きくなり、性能低下につながっている。

マルチスレッド環境で、通信を効率的に行うためにはメッセージ通信ではなく、メモリに直接書き込む片方向通信をつかうことにより、MPI のメッセージ通信のモデルによるオーバーヘッドを避けることが考えられる。しかし、MPI における片方向通信においてはウインドウ操作などのモデルにしたがわなくてはならず、煩雑にある。そこで、小田嶋らの研究 [5,6] では、MPI ではなく、InfiniBand 向けの API である InfiniBand Verbs や、Tofu-D インターコネクト向けの uTofu といった、各通信インターフェースごとの低レベル API を直接利用することによりマルチタスク

¹ 筑波大学 理工情報生命学術院

² 筑波大学 計算科学研究センター

³ 理化学研究所 計算科学研究センター

^{a)} ywatanabe@hpcs.cs.tsukuba.ac.jp

PGAS モデルで発生するスレッドごとのノード間通信の性能を改善することが可能であるかどうか検討を行い、改善可能であることが確認された。しかしながら、InfiniBand や Tofu-D などの複数のインターフェースへの対応を検討した場合、それぞれ個別に実装する必要があり、そのコストは無視できるものではない。そこで、様々な通信インターフェースを統一化された API を用いて利用可能とする軽量な通信ライブラリである UCX の利用を検討している。現時点では UCX は uTofu をサポートしていないことから、本稿では UCX の uTofu 対応について検討した。

本稿は、次章以降のように構成されている。2 章では関連研究の紹介を行う。3 章では UCX について、また 4 章では Tofu-D インターフェースについて説明する。5 章では、移植を行う前の予備評価として、UCX の TCP モードを利用した富岳での性能評価を行う。6 章では、UCX の下位レイヤである UCT に対し uTofu レイヤを追加するための実装の調査を行い、7 章では UCX の uTofu 対応の方法について述べる。8 章では、UCT API を利用し、実装を行っている Tofu-D 対応 UCX の片方向通信の性能の評価を行う。最後に、9 章にて結論を述べる。

2. 関連研究

RIKEN MPI [8] は、MPI の実装の 1 つである MPICH ベースとした Tofu-D インターフェース対応の MPI 実装である。MPICH では、通信ライブラリとして UCX や libfabric [7] などが利用可能である。libfabric は UCX と同様、様々なデバイスの通信ライブラリを抽象化し共通化された API を提供するライブラリである。RIKEN MPI では、主に libfabric に uTofu を用いた実装を追加することにより MPICH の Tofu-D インターフェースへの対応を行っている。

Papadopoulou らは、UCX の InfiniBand レイヤにおける最適化と性能評価を行った [9]。UCX の InfiniBand 対応においては、InfiniBand Verbs を用いた実装と、MLX5 ドライバを用いた 2 つの実装があり、MLX5 ドライバを用いた RC プロトコル向けの実装は実際には xRC (eXtended Reliable Connection) が利用されていると記述されている。PUT 通信における実行命令数については、MLX5 ドライバを用いた実装の命令数は InfiniBand Verbs を用いた実装の命令数よりも少なく抑えられている一方、InfiniBand の RC プロトコルは送信先ごとに個別に送信キューを用意するのに対し xRC では複数の通信先と 1 つの送信キューを共有するため、送信キューのポーリングの最適化が重要であることなどが述べられている。また、InfiniBand Verbs を直接利用する場合と比較し、UCX を用いて InfiniBand を利用する場合の性能については、オーバーヘッドが存在するものの最適化により最小限に抑えられている。そのため、本稿が取り組む UCX の uTofu ライブラリ追加による Tofu-D インターフェース対応においても、最適化により、uTofu を直接利

表 1: UCX を構成する 3 つのレイヤ

UCP	UCT を利用して実装されるプロトコル機能を提供
UCT	各種通信ライブラリを薄く抽象化し統一された API を提供
UCS	UCX で利用される共有ユーティリティの提供

用する場合と UCX を利用する場合の性能差を抑えることが可能であると推測される。

3. UCX: Unified Communication X

UCX は、様々な通信インターフェースに対する操作を統一された API を用いて利用可能とする通信ライブラリである。対応するハードウェアの例としては InfiniBand や OmniPath, また NVIDIA GPU 等があり、ベンダーや大学、研究所等の様々な機関によるコラボレーションにより開発が行われている。UCX により提供される機能としては PUT や GET の片方向通信やタグマッチング通信に加え、Atomic 操作や Active Message といった様々な機能を提供している。1 章で述べたマルチタスク PGAS モデルの実現においては、基本的な片方向通信を利用した通信に加え、Active Message を用いた関数呼び出しによるノード間のタスク依存性解決やノード間のタスクマイグレーションといったことが可能であると推測される。

UCX のソフトウェアアーキテクチャについて 3.1 節に記述する。

3.1 UCX のソフトウェアアーキテクチャ

UCX のソフトウェアアーキテクチャの概要を図 1 に示す (図は [1] より引用)。UCX は、UCP (UC-Protocols), UCT (UC-Transports), UCS (UC-Services) の 3 つのレイヤから構成される。それぞれの概要を表 1 に示す。

UCT レイヤでは、InfiniBand Verbs や uGNI 等の各種通信ライブラリを薄く抽象化し、統一された低レベル API を提供する。薄く抽象化しているため、UCT における抽象化によるオーバーヘッドは小さい。なお、InfiniBand のように RC (Reliable Connection), UD (Unreliable Datagram) 等の複数の通信プロトコルが提供される通信インターフェースの場合、当該通信レイヤ内においてそれぞれが実装されている。UCP レイヤでは、UCT を利用し、リモートノードと接続する際に自動的に最適な通信レイヤの選択を行う機能や、複数の通信ポートを自動的に利用する Multi-rail 機能などの、いわゆるプロトコル機能を提供する。UCT と比較し高級な機能を提供するためオーバーヘッドは大きくなるが、実装の最適化により最小限に抑えられている。UCS レイヤでは、メモリ確保 API や、ハッシュ、ツリーなどのデータ構造に関する機能を提供する。

4. Tofu-D インターフェース

本章では、A64FX CPU に統合された通信インターフェー

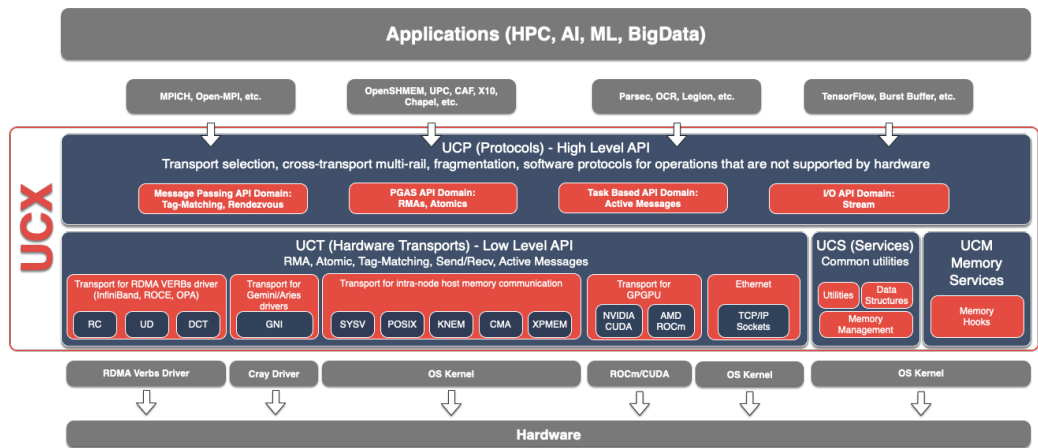


図 1: UCX のソフトウェアアーキテクチャ [1] より引用)

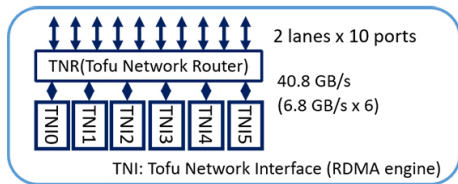


図 2: Tofu-D のダイアグラム ([3] より引用)

表 2: CQ, VCQ に関する制約

TNI あたりの CQ 数	9
CQ あたりの VCQ 数	8

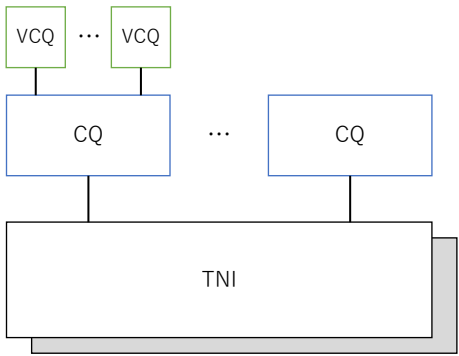


図 3: TNI と CQ, VCQ の関係 ([11] を参照)

スである Tofu-D について記述する。

4.1 Tofu-D の構成

Tofu-D インターフェースの構成を図 2 に示す ([3] より引用)。Tofu-D は物理的に、6 つの TNI (Tofu Network Interface) と、TNI に接続される 20 の通信レーンより構成される。TNI は RDMA エンジンであり、1 つあたり理論値として 6.8GB/s、合計 40.8GB/s の性能をもつ。次節に示すサンプルコードの通り、どの TNI を利用するかは uTofu を用いてユーザが指定して記述する。

また、TNI と物理および仮想キューの関係を図 3 に示す。それぞれの TNI には、CQ (Control Queue) と呼ばれる物理的なキューが存在する。また、CQ 上に VCQ (Virtual Control Queue) と呼ばれる仮想キューが存在する。4.2.2 項で例示する uTofu を利用したプログラム例のように、ユーザは利用する TNI を指定した上で VCQ を作成し、VCQ を経由して通信等の操作を行う。なお、表 2 に示す通り、TNI あたりの CQ 数、CQ あたりの VCQ 数には制約がある。TNI あたりの CQ 数は 9、CQ あたりの VCQ 数は 8 であり、計算ノード当たりの TNI 数は 6 であることから、1 台の計算ノード上に作成可能な VCQ 数は最大 432 となる。

4.2 uTofu API

uTofu とは、ユーザ空間から Tofu-D インターフェースを利用するためのプログラミングインターフェースである。uTofu を利用することで、PUT/GET の片方向通信、バリア同期、またバリア同期を利用した集合演算などの通信を行うことが可能である。

4.2.1 stadd: steering address

Tofu-D インターフェースでは RDMA 操作を行う際、受信信用および送信信用バッファとして、malloc() 関数等で確保したメモリ領域のアドレスを直接指定することはできない。これは、Tofu-D インターフェースが仮想アドレスを直接解釈することができないためである。そのため、RDMA 操作を行う事前準備として、ユーザが確保した送受信信用バッファの仮想アドレスを登録し、stadd (steering address) と呼ばれる Tofu-D インターフェースが解釈可能なアドレス形式に変換を行う必要がある。

4.2.2 uTofu を利用したプログラム例

ソースコード 1 は uTofu を用いた PUT 通信の例である。はじめに、utofu_get_onesided_tnis() 関数を利用し、利用可能な TNI の ID と TNI 数を取得する。次に、utofu_create_vcq() 関数を利用し、指定した TNI (この例では TNI 0) 上に VCQ

ソースコード 1: uTofu による PUT 通信の例

```

1 utofu_get_onesided_tnis(&tni_ids, &num_tnis);
2
3 utofu_create_vcq(tni_ids[0], 0, &hdl_vcq);
4 utofu_query_vcq_id(hdl_vcq, &vcq_id);
5
6 // register buffer to tofu device
7 utofu_reg_mem(hdl_vcq, buf, buf_size, 0, &
    local_stadd);
8
9 /* exchange vcq_id and stadd with remote node */
10 utofu_set_vcq_id_path(&remote_vcq_id, NULL);
11
12 ...
13
14 if (sender) {
15     utofu_put(hdl_vcq, remote_vcq_id, local_stadd,
16         remote_stadd, size, 0, flags, 0);
17     /* polling TCQ */
18 }

```

を作成する。また、`utofu_query_vcq_id()` 関数を利用し、当該 VCQ の ID を取得する。その後、`malloc()` 関数等で確保された、RDMA 通信に利用するメモリ領域を Tofu-D インターフェースに登録し、`stadd` を取得する。その後通信先ノードと VCQ ID および `stadd` を交換し、`utofu_vcq_id_path()` 関数を用いて通信経路を設定する。これにより RDMA 操作が可能となる。通信先ノードの VCQ の ID、送信元バッファに該当する `stadd`、送信先バッファに該当する `stadd`、長さ、フラグ等を `utofu_put()` 関数で指定し呼ぶことで送信処理の実行依頼が完了する。また、`utofu_put()` 関数を呼び出す際、フラグに指定するオプションによっては、送信完了の確認や、受信完了の確認が可能となる。例えば、`UTOFU_ONESIDED_FLAG_TCQ_NOTICE` フラグを指定した場合、送信時に指定した VCQ に対応する TCQ (Transmission Completion Queue) に対し、送信完了時にディスクリプタが書き込まれるため、uTofu を利用するプログラムにおいて TCQ のポーリングを行うことで送信完了を確認可能である。

5. TCP モードを用いた予備評価

UCX が TCP を用いて通信可能であること、また Tofu-D インターフェース上で IP 通信が可能であることを利用し、富岳において UCX の TCP モードの性能評価を行った。TCP モードでは PUT/GET の RDMA 操作が利用できなかったため、タグマッチングによる通信の性能の計測を行った。また、比較のため、Tofu-D インターフェースに対応する富士通 MPI を利用し、タグマッチングによる通信の計測を行った。計測は pingpong 形式で行い、各通信サイ

表 3: 評価環境

CPU	富士通 A64FX
インターコネクト	Tofu-D
TNI あたりのバンド幅	6.8 GB/s
インジェクションバンド幅	40.8GB/s

表 4: 測定で利用するソフトウェア

UCX version	1.10.0
Fujitsu MPI version	4.5.0 tcsds-1.2.31

ズごとに 10000 回試行し、バンド幅の平均を算出した。

5.1 測定環境

Tofu-D インターフェース上で、UCX の TCP モードの性能を測定を行うために理化学研究所のスーパーコンピュータ「富岳」を利用する。表 3 に富岳の概要を示す。また、利用するソフトウェアについては表 4 に示す。

5.2 測定結果

TCP モードを利用した UCX のタグマッチング通信の結果を図 4 に示す。また、Fujitsu MPI を利用した場合のタグマッチング通信の結果を図 5 に示す。TCP モードを利用した UCX の測定結果に着目すると、メッセージサイズが 1024B の場合の性能は 25MB/s であり、4MB (4194304 B) の場合は約 170MB/s であることが確認できる。また、最も高い性能を得られたのは、メッセージサイズが 2MB (2097152 B) の場合の約 175MB/s であり、TNI 1 つあたりの理論性能の約 3% 程度である。一方、Fujitsu MPI を利用した測定結果を確認すると、メッセージサイズが 1024B の場合は約 490MB/s、メッセージサイズが 2MB の場合は約 6280MB/s であり、最も良い性能については TNI 1 つあたりの理論性能の約 92% である。TCP モードを利用した UCX の性能については、IP over Tofu-D のオーバーヘッドによるものか、あるいは何らかの帯域制御によるものかは不明であるものの、この結果から、UCX を TCP モードで利用することは性能の観点から現実的ではないと考えられる。

6. UCT の詳細検討

各種通信ライブラリを抽象化するレイヤである UCT に uTofu による実装を追加するにあたり、公開されているソースコード [10] を元に実装に関する調査を行った。各種通信レイヤの抽象化箇所は複数のオブジェクトで構成されており、それぞれのオブジェクトは図 6 のような関連性だった。

6.1 `uct_md_{name}.t`

`uct_md.t` はメモリドメインに関する情報を管理するオブジェクトである。このオブジェクトは、各通信ライブラリごとに 1 つ、あるいは各通信ライブラリにおいて物理デバ

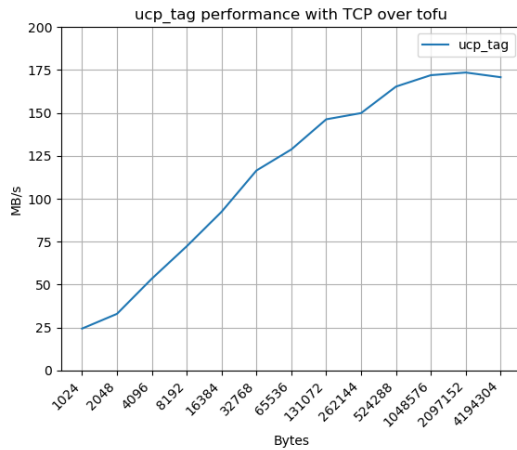


図 4: TCP モードを利用した UCX のタグマッチング通信の性能評価

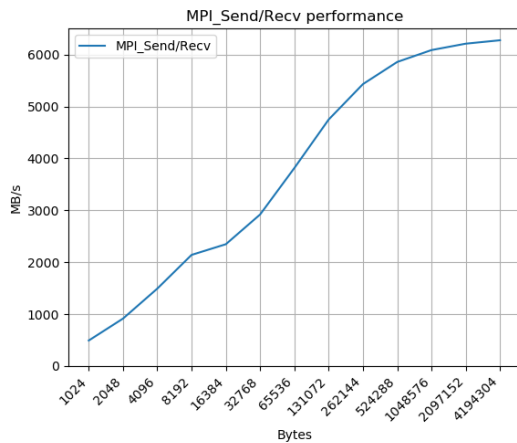


図 5: Fujitsu MPI を利用したタグマッチング通信の性能評価

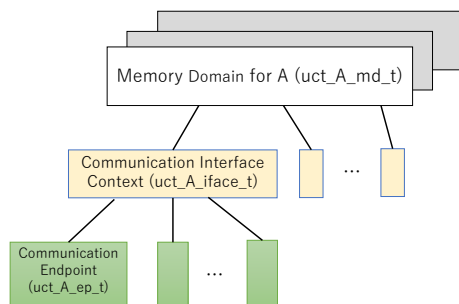


図 6: 各通信レイヤにおけるオブジェクトの構成

表 5: UCT のそれぞれの通信レイヤにおけるオブジェクト

uct_md_{name}_t	通信レイヤ {name} 用のメモリドメイン
uct_iface_{name}_t	通信レイヤ {name} 用の通信インターフェースコンテキスト
uct_ep_{name}_t	通信レイヤ {name} 用の通信先エンドポイントオブジェクト

イスごとに 1 つ用意される。例としては、RMA 通信のメモリ保護用オブジェクトの管理情報 (InfiniBand Verbs の場

合、メモリ保護用のキーを発行する際に必要となる `ibv_pd` オブジェクト) などが挙げられる。UCT の API においては、各通信インターフェースごとに実装されたメモリドメインのオブジェクトを `uct_md_t` として抽象化している。

6.2 uct_iface_{name}_t

`uct_iface_t` は、通信インターフェースコンテキストに関する情報を管理するオブジェクトである。このオブジェクトは `uct_md_t` に関連するオブジェクトであり、一つのメモリドメインオブジェクトに対し複数生成されうる。格納されうる情報の例としては、各通信先エンドポイントと共有して利用する情報 (`ibvverbs` の RC プロトコルの場合、複数ノードからの通信の受信に利用される Shared Receive Queue である `ibv_srq` オブジェクト) などが挙げられる。UCT の API においては、各通信インターフェースごとに実装された通信インターフェースコンテキストのオブジェクトを `uct_iface_t` として抽象化している。

6.3 uct_ep_{name}_t

`uct_ep_t` は、通信先エンドポイントに関する個別情報を格納するオブジェクトである。このオブジェクトは通信先となるリモートエンドポイントごとに個別に生成される。格納されうる情報の例としては、リモートエンドポイントのアドレス、リモートエンドポイントとの通信に利用するキュー (InfiniBand Verbs の場合、RC プロトコルの送信用キュー) などが挙げられる。UCT の API においては、各通信インターフェースごとに実装された通信先エンドポイントのオブジェクトを `uct_ep_t` として抽象化している。

7. UCX の Tofu-D 対応の検討

5 章での予備評価の通り、UCX の TCP モードを用いた場合、Tofu-D インターフェースの性能を十分に活用できないことが確認された。そこで、本章においては、UCX に uTofu のレイヤを追加することで Tofu-D の性能を十分に利用可能とするための検討を行う。

7.1 TNI の活用方法

4.1 節の通り、Tofu-D インターフェースには 6 つの TNI (RDMA エンジン) が搭載されており、すべてを効率的に利用することが重要であると考えられる。UCT に uTofu レイヤを追加するにあたり、下記の 2 つの方法が考えられる。

- TNI1 つを 1 インターフェースとみなし、UCX の multi-rail 機能を利用することで 6 つの TNI を利用する
- uTofu レイヤ内で 6 つの TNI を明示的に利用し、ネットワークインターフェースとして 1 つとみなす

UCX では、複数のポートを備えた InfiniBand HCA の場合、各ポートを 1 つのインターフェースとして取り扱う。つまり、2 つポートを持つ HCA の場合、2 つインターフェー

スが存在するものとして扱われる。この場合、Multi-rail 機能により、暗黙的に2つのインターフェースを利用することも可能となる。[1]によれば、例えば100MBのメッセージ通信を行う際、50MBずつに分割し、インターフェース1, 2を用いて50MBずつ送信される。一方、Multi-rail 機能にはいくつかの制約があり、本機能を用いて同時に利用可能なインターフェースは最大でも4つであることや、InfiniBandの場合Hardware tag-matchingが有効となっている場合は利用できないといった注意点がある。uTofuでMulti-rail機能を利用すると仮定した場合、あくまでもインジェクションバンド幅が40.8GB/sであり、2ノード間通信で得られるバンド幅ではないため、Multi-rail機能により恩恵が得られないと推測することも可能である。

一方、uTofuレイヤにおいて明示的にTNIを管理し、UCXを利用するユーザからは1つのネットワークインターフェースとして見えるように抽象化する場合、接続先ごとに利用するTNIを選択することが可能であるため、より効率的にTNIを利用することが可能であると考えられる。そのため、本手法が好ましいと推測されるが、UCXのソフトウェアアーキテクチャにおいてこのような実装が想定されているかどうか、また不都合が発生するかどうかといったデメリットについての調査は本稿執筆段階においては不十分である。

7.2 VCQの生成および利用方法

4.1節で説明した通り、1台の計算ノード上に作成可能なVCQ数は432となる。富岳の場合、システム全体で158976ノード存在するため、通信先ノードごとに独立したVCQを生成することは不可能である。したがって、1つのVCQを複数のノードとの通信に利用する必要がある。本稿執筆時点では、TNIごとに1つ以上VCQを生成し、何らかのルールにしたがって複数のノードと共有することを想定しているが、具体的な数やルールについては調査段階にある。また、UCXと他の通信ライブラリを併用する際、併用する通信ライブラリのVCQの消費数を考慮する必要がある。

7.3 stadd(steering address)の取り扱い

ソースコード2にuTofuのPUT APIの定義を示す。また、ソースコード2にUCPのPUT APIのうち1つの定義を示す(当該APIの定義は[1,11]から引用)。。uTofuのPUT APIでは引数として、キューであるVCQ(vcq_hdl)、通信先VCQのID(rmt_vcq_id)、送信元バッファに対応するstadd(lcl_stadd)、送信先バッファに対応するstadd(rmt_stadd)、長さ(length)、リモートノードに受信完了通知を行う場合に送信される値(edata)、送信フラグ(flags)、コールバック用データ(cbdata)の8つを指定する。UCXのPUT APIでは、ucp_put_nbi関数の場合、通信先エンドポイント(ep)、送信元バッファ(buffer)、長さ(length)、送信先バッファ

ソースコード 2: uTofu の RDMA PUT API の例

```
1 int utofu_put(
2     utofu_vcq_hdl_t vcq_hdl,
3     utofu_vcq_id_t rmt_vcq_id,
4     utofu_stadd_t lcl_stadd,
5     utofu_stadd_t rmt_stadd,
6     size_t length,
7     uint64_t edata,
8     unsigned long int flags,
9     void *cbdata
10 )
```

ソースコード 3: UCX の RDMA PUT API の例

```
1 ucs_status_t ucp_put_nbi(
2     ucp_ep_h ep,
3     const void *buffer,
4     size_t length,
5     uint64_t remote_addr,
6     ucp_rkey_h rkey
7 )
```

(remote_addr)、リモートアクセスに必要なメモリ保護用のエータ(rkey)の5つを指定する。

uTofuとUCXのPUT APIの違いに着目すると、uTofuでは送信元、送信先ともにTofu-Dインターフェースが理解可能なstaddを指定する必要があるのに対し、UCPではユーザが確保したバッファの仮想アドレスを直接指定するという違いがある。また、uTofuでは、UCPのrkeyに対応するメモリ保護用のオブジェクトはない。

この差異については、次の通り解決することが可能である。通信先のバッファのstaddについては、UCX上ではrkeyとして取り扱うことで対応が可能である。通常のUCXの使い方のように、RDMA操作を行う準備として事前に登録し通信先ノードと交換を行い、RDMAを行う際にrkeyをstaddとして利用すればよい。ローカルのバッファに対応するstaddについては、RMA操作を行うたびに登録および登録解除を行うことで対応可能である。

8. Tofu-D 対応 UCX の UCT API による性能評価

本稿執筆時点では、Tofu-D 対応 UCX の実装について、UCT API を用いた PUT 通信について動作する点まで確認できている。そこで、UCX の UCT API を利用した PUT 通信と、MPI および uTofu を用いた PUT 通信の性能比較を行った。

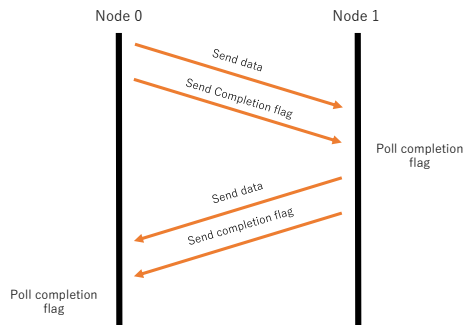


図 7: PUT 通信による pingpong のフロー

8.1 測定環境

測定環境は、表 3 および表 4 と同一である。なお、UCT の PUT 通信としては、ゼロコピー PUT API (`uct_ep_put_zcopy`) を利用する。また、図 7 に示すような pingpong 通信による性能測定を行う。リモートノードに対しデータを送信する際、「データの送信」と「完了フラグの送信」の 2 回の PUT 通信を行う。受信側においては、完了フラグのポーリングを行い、変更されたタイミングで受信完了とみなす。この通信を複数回実行し、その平均を性能として算出する。uTofu においては、キューのポーリングにより受信完了を確認することも可能であるが、本測定は利用していない。

8.2 測定結果

測定結果を図 8 に示す。uTofu の性能が最もよく、UCT、MPI と並んでいることが確認できる。uTofu と UCT の差は小さく、UCX では UCT における抽象化によるオーバーヘッドは小さいことが確認できる。一方、MPI についてはメッセージサイズが 4MB の場合は他 2 つとほぼ同等である一方、特にメッセージサイズが小さい場合の性能に差があることが確認できる。これは、MPI で PUT 通信を行う際に通信先メモリに対するロック処理を行っているため、そのオーバーヘッドによるものであると推測される。よって、メッセージサイズが大きくなることで通信時間に対するロック処理の時間の割合が小さくなり、メッセージサイズが 4MB の場合に他 2 つと同等の性能となっていると考えられる。

9. おわりに

本稿では、UCX の Tofu-D インターフェース対応に関する検討を行った。また、本稿執筆時点では、設計および実装を進めている Tofu-D 対応 UCX の UCT API を用いた PUT 通信の性能測定を行い、uTofu を利用した場合と同等の性能、また MPI よりも高い性能が得られることを確認した。今後も Tofu-D 対応 UCX の実装を継続し、UCP API を用いた通信測定や OpenSHMEM 等の UCX を利用するソフトウェアの動作確認等を進めたいと考えている。また、UCX を通信ライブラリとして利用する、分散タスク並列実行フ

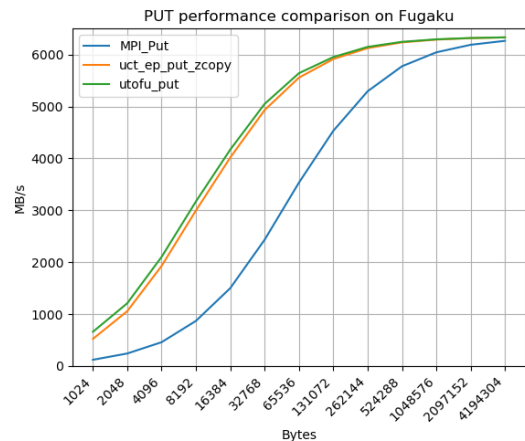


図 8: uTofu, UCX (UCT), MPI の PUT 通信性能の比較

レームワークの設計および実装を行う。

謝辞 本研究成果（の一部）は、理化学研究所のスーパーコンピュータ「富岳」を利用して得られたものです。

参考文献

- [1] The Unified Communication X Library <https://openucx.org/>
- [2] Shamis, Pavel, Manjunath Gorentla Venkata, M. Graham Lopez, Matthew B. Baker, Oscar Hernandez, Yossi Itigin, Mike Dubman et al. "UCX: an open source framework for HPC network APIs and beyond." In 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects, pp. 40-43. IEEE, 2015.
- [3] 富岳システム紹介 <https://www.r-ccs.riken.jp/fugaku/system/> (2021.8.31)
- [4] Balaji, Pavan, Darius Buntinas, David Goodell, William Gropp, and Rajeev Thakur. "Fine-Grained Multithreading Support for Hybrid Threaded MPI Programming." The International Journal of High Performance Computing Applications 24, no. 1 (February 2010): 49-57. <https://doi.org/10.1177/1094342009360206>.
- [5] 小田嶋哲哉, 森江善之, 李 珍泌, 佐藤三久 「マルチタスク PGAS モデルをサポートするノード間軽量通信レイヤの検討」 情報処理学会研究報告 (ハイパフォーマンスコンピューティング), Vol. 2019-HPC-168, No. 1 (2019)
- [6] 小田嶋哲哉, 李 珍泌, 佐藤三久 「ルチスレッドプログラムにおける通信ライブラリの性能比較」 情報処理学会研究報告 (ハイパフォーマンスコンピューティング), Vol. 2019-HPC-170, No. 33 (2019)
- [7] Libfabric - OpenFabrics <https://ofiwg.github.io/libfabric/>
- [8] An Overview of RIKEN MPI (MPICH-Tofu) <https://www.r-ccs.riken.jp/wp/wp-content/uploads/2021/01/MPICH-Tofu.pdf>
- [9] Papadopoulou, Nikela, Lena Oden, and Pavan Balaji. "A performance study of UCX over InfiniBand." In 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 345-354. IEEE, 2017.
- [10] openucx/ucx - Github <https://github.com/openucx/ucx>
- [11] Development Studio uTofu 使用手引書