Vol.2012-HPC-133 No.7
2012/3/26

片側通信による，グローバルデータ構造の効率的な操作方法の検討

安島雄一郎[†,††]　秋元秀行[†,††]
岡本高幸[†,††]　三浦健一[†,††]　住元真司[†,††]

片側通信はプロセス間で書き込み(Put)，読み出し(Get)等を行う，グローバルなメモリ参照手段である．片側通信とプロセス内のローカルなメモリ参照の間には大きな性能差が存在するため，適切に組合せた並列処理の実装は難しい．本稿では内部的に片側通信を使用し，複数プロセスのメモリに配置されたグローバルデータ構造を操作するライブラリの構築を提案する．効率的な並列アルゴリズムをライブラリとして提供することにより，プログラマは複雑な並列アルゴリズムの実装から解放される．さらにメッセージ通信実装でのグローバルデータ構造の使用を想定して性能見積りおよび分析を行い，Atomic Compare and Swap 機能とメモリ参照順序保証機能によって高性能なグローバルデータ構造ライブラリを構築できることを示した．

A Study of Efficient Manipulations of Global Data Structures via One-Sided Communication

Yuichiro Ajima[†,††]　Hideyuki Akimoto[†,††]
Takayuki Okamoto[†,††]　Kenichi Miura[†,††]
and Shinji Sumimoto[†,††]

One-sided communication is a global memory access means to write (Put) and to read (Get) between processes. Because of a big performance difference between local and global memory access, combining local and global memory access to implement an effective parallel algorithm is difficult. In this paper, we propose to build a library which uses one-sided communication internally and manipulates a global data structure placed across memory regions of multiple processes. By providing efficient parallel algorithms as a library, programmers become free from implementing complex parallel algorithms. Additionally, we estimate and analyze performance assumes the use of global data structures in the implementation of message communication. The results show that a highly efficient global data structure library can be built by the Atomic Compare and Swap memory access and the memory order guarantee.

1. はじめに

本稿では内部的に片側通信を使用し，複数プロセスのメモリに配置されたグローバルデータ構造を操作するライブラリの構築を提案する．効率的な並列アルゴリズムをライブラリとして提供することにより，プログラマは複雑な並列アルゴリズムの実装から解放される．また，グローバルデータ構造ライブラリの構築における 4 つの課題，メモリ領域の割当て・解放メカニズム，グローバルデータ構造操作の遅延，グローバルデータ構造の一貫性維持，データの最適配置について説明する．

さらにグローバルデータ構造インタフェースの性能特性と課題を抽出するため，現在最も一般的な通信モデルであるメッセージ通信を題材として性能見積りおよび分析を行った．性能見積は共有受信キューの操作オーバーヘッドを対象とし，メモリ参照パターンからオーバーヘッド時間を机上で積算した．ただし，基本通信性能値は実機の測定結果を使用した．測定環境は富士通の最新スーパーコンピュータ PRIMEHPC FX10[9]の試験機を使用し，片側通信ライブラリには Technical Computing Suite[10] MPI ライブラリの拡張 RDMA インタフェース(FJMPI_Rdma)を使用した．性能見積り結果では，Put/Get のみを使用して排他制御を行うと 100 万プロセス規模ではミリ秒オーダーのオーバーヘッドがかかるが，Atomic Compare and Swap による排他制御を使用する場合はオーバーヘッドが約 8μ 秒に削減された．メモリ参照順序が保証される場合はオーバーヘッドが約 3μ 秒まで削減されることも分かった．また，ロックフリーキュー[8]には通信バッファ量を動的に変更しやすい長所があるものの，オーバーヘッドは約 14μ 秒と排他制御より大きいことも分かった．

本研究は JST CREST の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の創出プロジェクトの研究課題「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」の一環として実施した．

以降では 2 章で本研究の背景，主に両側通信の問題について述べ，3 章で片側通信ライブラリについて説明する．4 章でグローバルデータ構造ライブラリの構築を提案し，5 章で共有キュー操作の性能見積りを行う．最後に 6 章で本論文をまとめるとともに，今後の課題について述べる．

2. 背景

2018 年から 2020 年頃に実現されるエクサスケールの時代には，超並列計算機の演算性能は 2012 年時点と比較して 100 倍に達すると予想されている [6]．今後，超並列計

† 富士通株式会社 次世代テクニカルコンピューティング開発本部
Fujitsu Limited, Next Generation Technical Computing Unit
†† 独立行政法人科学技術振興機構 戦略的創造研究推進事業
Japan Science and Technology Agency (JST), Core Research for Evolutional Science and Technology (CREST)

算機向け High Performance Computing (HPC)アプリケーションの実行時間における通信時間の比率の増加がますます深刻な問題となる．なぜなら，近年の超並列計算機の性能向上はノードあたりプロセッサコア数，およびノード数の増加でもたらされているからである．超並列ではワーキングセットを細かく分割して並列処理するので，ノード数の増加によりインターコネクト性能の重要性が上がる．また，ノードあたりプロセッサコア数の増加により，演算性能が律速であった既存のアプリケーションがメモリ，インターコネクトが律速に変化するケースが増加する．

超並列計算機の並列度の向上を利用して HPC アプリケーションの性能を向上するには，増加する通信時間への対処が必要となる．もっとも効果のある対処方法の一つは，計算と通信のオーバーラップである．インターコネクトのデータ転送がプロセッサの演算と完全に並行して行われれば，アプリケーションの実行時間に占める通信時間の割合は 0 になる．計算と通信のオーバーラップには送信側のオーバーラップと，受信側のオーバーラップの 2 種類が存在する．前者はデータ送信完了までデータの格納領域を破壊しない計算を並行して行い，後者はデータ受信完了まで受信データ格納領域を破壊しない計算を並行して行う．

計算と通信オーバーラップ時のデータ相互非破壊を保証するのは，一般的にプログラマの責任とされている．多くの通信ライブラリでは，計算と通信オーバーラップ時のデータ相互非破壊保証をプログラマの責任とするだけでなく，計算と通信をオーバーラップすることに対しする責任を負わない．送達保証などの基本機能や遅延削減などの性能最適化の実現を優先し，通信開始を遅延してもよい規則になっている．特にデータ転送両端間のプロトコル，すなわち両側通信によって通信機能を実現すると，計算と通信のオーバーラップが犠牲になる場合がある．

例えば通信ライブラリ MPI-1 の Send 系関数は，送信完了したデータは確実に宛先に届くことを保証する機能を持つ．このため MPI は，受信側に受信データを格納する領域が確保されたことを確認してから，インターコネクト・ハードウェアによるデータ転送を開始する．具体的には，確保されている受信バッファの空き容量を事前に伝えておく Eager プロトコルと，データ送信直前に受信側データ格納領域の確保を確認する Rendezvous プロトコルを使用する．これらのプロトコルが進むことができるのは MPI ライブラリに処理が移っている間のみのため，オーバーラップを意図した計算中にデータ転送が開始されず，Wait 関数でデータを待ち合わせる箇所までデータ転送開始が遅延され，計算と通信がシリアライズされる場合がある．

計算処理と並行して非同期に通信ライブラリのプロトコルを進めるには，大きく分けて 3 つの方式がある．1 つ目は，プロセッサコアを通信ライブラリのプロトコル専用に割当て，通信ライブラリのプロトコルを専用スレッドで進める方式である．この方式には，計算に使用できるプロセス／スレッド数が減少する短所がある．2 つ目は，計算プロセス／スレッド数を減らさずに，追加でスレッドを立ち上げて通信ライブラリのプロトコルを専用とする方式である．この方式ではプロトコル専用スレッドの動作が計算プロセス／スレッドの演算性能に対して外乱要因となるが，プロセッサコアが SMT 機能を備えていれば比較的良い方式である．3 つ目はインターコネクト・ハードウェアに通信ライブラリのプロトコル処理をオフロードする方式である．計算性能に対する影響は最も少ないが，多様な通信ライブラリに対応するには実質的にプロセッサコアの搭載が必要となる．本質的には 1 つ目の方式と同じであるにも関わらず，ハードウェア，ソフトウェアとも設計が複雑になる短所がある．

3. 片側通信ライブラリ

片側通信ライブラリとは，データ転送の両端の片側のユーザープログラムがデータ転送のパラメータを指定すれば，もう一方のユーザープログラムの介在無しにデータ転送が行われる種類の通信ライブラリである．本章では特に既存の片側通信ライブラリのうち，より計算と通信がオーバーラップできることを目指し，片側通信のセマンティクスを採用するだけでなく，データ転送両端の通信ライブラリ間プロトコルも不要にすることを考慮している片側通信ライブラリを紹介する．

片側通信ライブラリは一般的に，インターコネクト・ハードウェアで一般的な Remote Direct Memory Access (RDMA)通信機能を抽象化した Remote Memory Access (RMA)通信モデルを採用する．RDMA 通信では起点となるノードが対象になるノードにメモリ参照を要求し，対象は必要に応じて起点に応答する．RDMA 通信機能は InfiniBand[5]，Tofu インターコネクト [1][2]など複数のインターコネクト仕様で定義，実装されているが，主たるメモリ参照の種類は共通である．そのためメモリ参照の種類は RMA 通信モデルでも共通となっており，連続データの書き込み(Put)，連続データの読み出し(Get)，不可分の比較交換(Atomic Compare and Swap)などがある．一方，通信バッファの割当て・解放メカニズム，メモリ参照のオーダリング規則は各種インターコネクト・ハードウェアの RDMA 通信機能によって仕様が異なっている．RMA 通信モデルではポータビリティを確保するため，通信バッファの扱いやメモリ・オーダリング規則を最大公約数的なものとしている．例えば，通信バッファの割当ては複数ノードで同期的に行う，メモリ参照は同期関数で同期化されて他ノードに可視化される，同期関数の間に行われる複数のメモリ参照間では順序を保証しない，といった仕様である．

なお，片側通信ライブラリの機能は一般的に，通信ライブラリ間の両側通信で模擬する実装も許されている．これは，片側通信ライブラリの RMA 通信モデルが要求する通信機能がインターコネクト・ハードウェアの RDMA 通信機能に存在しない場合でも，ポータビリティを確保するためである．

以上に述べた特徴を持つ片側通信ライブラリには MPI-2 RMA 受動的対象通信 [11]，

ARMCI[3]，GASNet 拡張 API[4]がある．また，富士通の Technical Computing Suite MPI ライブラリに含まれる独自の拡張 RDMA インタフェース FJMPI_Rdma は，基本的には Tofu インターコネクトの RDMA 通信機能を利用するためのライブラリであるが，一部機能を抽象化して異種インターコネクト・ハードウェア間のポータビリティを確保している．以降ではこれら既存の片側通信ライブラリの概要を説明する．

3.1 MPI-2 RMA 受動的対象通信

MPI-2 の RMA 通信には，能動的対象通信と受動的対象通信がある．能動的対象通信は通信プリミティブが片側通信であるものの，同期メカニズムは両側通信での実装を前提としている．本稿で対象とする片側通信ライブラリに該当するのは受動的対象通信である．

MPI-2 RMA 受動的対象通信では，まず対象の通信バッファをロックしてアクセス区間に入る．続いて Put，Get 等のメモリ参照を行い，最後に対象の通信バッファをアンロックする．アンロックはアクセス区間中のメモリ参照完了を保証するとともにアクセス区間を終了する．ロックには排他ロックと共有ロックがある．共有ロック処理は片側通信で実現するのが難しいため，実際の MPI-2 RMA 受動的対象通信実装では両側通信が使用されている．この場合，起点でアクセス区間中の RMA をアンロック操作まで遅延し，対象は排他制御とメモリ参照をまとめて要求される．

セマンティクスが片側通信でも，両側通信によって実装されていれば計算と通信のオーバーラップに支障がある．現在仕様検討が進んでいる MPI-3 RMA では，全起点ノードに対して通信バッファのアクセス区間を開始するロックオールや，起点において先行して実行されたメモリ参照の完了を保証するフラッシュなどの追加により，真の片側通信実装が可能になることを目指している．また，MPI-2 RMA では対応していなかった Atomic Compare and Swap も対応される．MPI-2 RMA および MPI-3 RMA では，アンロックやフラッシュ等で区切られた区間内のメモリ参照に関して，順序を保証しない．

3.2 ARMCI

ARMCI は Global Arrays ライブラリなどの実装に使用された片側通信ライブラリである．Put，Get 等の基本的なメモリ操作に対応するが，不可分操作は Fetch and Add と Swap のみで，Compare and Swap には対応しない．代わりに，排他制御専用の Mutex 機能に対応する．ARMCI のブロッキング Put は起点からデータ送信完了を保証する．対象における Put メモリ参照の完了を保証するにはフェンスを使用する．また，ノンブロッキング Put に関してもデータ送信完了を個別に問い合わせ，および待ち合わせることが可能となっている．フェンスで区切られた区間内のメモリ参照に関しては，順序は保証されない．

3.3 GASNet 拡張 API

GASNet は PGAS 言語である Unified Parallel C 言語などの実装に使用された通信ライブラリであり，片側通信インタフェースである拡張 API を含む．GASNet の拡張 API は Put/Get メモリ参照に対応し，不可分操作には対応していない．GASNet のブロッキング Put は対象の Put メモリ参照が完了してから戻るので，計算と通信のオーバーラップには使用できない．計算と通信のオーバーラップには，ノンブロッキングの Put を使用する必要がある．GASNet では個別のノンブロッキング Put について，対象における Put メモリ参照の完了を問い合わせ，および待ち合わせることができる．区間でしか順序を制御できない MPI-2/3 RMA や ARMCI に比べ，やや細かく順序を制御することが可能である．ただし，GASNet が実装レベルでも個別の順序制御を行うには，Put に対して対象から起点へ応答するインターコネクト・ハードウェアが必要である．

3.4 FJMPI_Rdma

FJMPI_Rdma は富士通の Technical Computing Suite MPI ライブラリの拡張 RDMA インタフェースであり，Tofu インターコネクトや InfiniBand に対応する．Tofu インターコネクト特有の，複数ネットワーク・インタフェースや順序制御などの機能を使用するインタフェースを有しつつ，InfiniBand においては同等のセマンティクスをエミュレートすることでポータビリティを確保することを目指して設計されている．

FJMPI_Rdma は Put/Get メモリ参照に対応し，不可分操作には対応していない．Put/Get のインタフェースはノンブロッキングのみである．対象におけるメモリ参照の完了は，応答の個数で確認するインタフェースとなっている．これは Tofu インターコネクトでは起点からの要求順序が対象からの応答順序に保存される特性を利用している．一方，Tofu インターコネクトがローカルのメモリを参照する場合，連続データであってもキャッシュラインの長さに分割し，順不同に実行することで高スループットを実現する．このメモリ参照の順序を制御するために，FJMPI_Rdma は Strong Order (STO)オプションを実装している．STO が指定された場合，各メモリ参照要求の最後のデータは他の部分のより後に書き込まれることが保証され，フェンス等の特殊な操作をしなくても，末尾のデータを検査するだけで RDMA 転送データを同期化できる．また STO オプションは，Tofu インターコネクトおよび SPARC64 VIIIfx / XIfx のプロセッサバスに特有の機能を利用し，ナノ秒オーダーの非常に小さいオーバーヘッドで実現されている．

また，本章で紹介した他の片側通信ライブラリは集合通信によって同期的に通信バッファ割当てを行うのに対し，FJMPI_Rdma は通信バッファも非同期に割当てる．対象はローカルな処理で通信バッファを登録し，起点は専用の問い合わせ関数を使用することで，指定番号の通信バッファが割当てられているかを対象に問い合わせる．

4. グローバルデータ構造ライブラリ構築の提案

4.1 概要

既存の片側通信ライブラリは書き込み(Put)，読み出し(Get)，不可分操作(Atomic Operation)などの，共有メモリ計算機のメモリ参照プリミティブを踏襲したインタフェースを有する．このインタフェース仕様の設計意図は，共有メモリ計算機で研究開発された様々な並列処理アルゴリズムを使用可能とし，さらにはアルゴリズムの記述性の面でも共有メモリ計算機に近づけた PGAS 言語のような処理系をも実装可能にすることである．ここで，分散メモリ計算機である超並列計算機はローカルのメモリ参照とグローバルのメモリ参照に遅延，帯域とも大きな性能差が存在するため，ランダムアクセスを基本とする共有メモリ計算機の並列処理アルゴリズムが常に有効であるとは限らない．例えばリストデータ構造を複数のノードから操作する場合，リストが 1 プロセスのメモリ領域内に収まっている場合と，複数プロセスのメモリ領域に跨っている場合では，適当なデータ構造とアルゴリズムは自ずと異なる．さらに，分散メモリにおけるグローバルデータ構造とアルゴリズムは共有メモリにおける並列データ構造とアルゴリズム以上に複雑であり，プログラム開発の負荷を増大させる．

そこで我々は，片側通信ライブラリの機能拡張の方向性として，分散メモリ上に配置されたグローバルデータ構造を操作するインタフェースの追加を提案する．基本的なグローバルデータ構造に対応するインタフェースを提供することにより，プログラマは効率的なアルゴリズムの追求や実行環境に適応したアルゴリズム実装の負荷から解放される．グローバルデータ構造インタフェースは，グローバルデータ構造を直接利用する新しいアプリケーションに利点をもたらすだけでなく，通信ライブラリ実装や並列処理言語実装において，非同期性と遅延削減に優れたグローバルデータ構造ライブラリを選択することにより，アプリケーション・プログラムも間接的に恩恵を受けることができる．例えば，エクサスケールの時代ではノード数，プロセッサコア数の増加から，各ノードが全ノードに対して通信専用のメモリ領域を割当てる通信モデルは非現実的になる．しかし通信バッファの動的割当て，解放の片側通信での実装，遅延削減の最適化は開発負荷を増大する．実装アルゴリズムが貧困化すれば遅延の増大，計算通信オーバーラップの阻害といった形でアプリケーション性能に悪影響を及ぼす．ここでグローバルデータ構造インタフェースを使用すれば，通信バッファの動的割当て，解放をグローバルデータ構造へのデータの挿入，削除操作に置き換えることで，グローバルデータ構造インタフェースに実装された非同期性と遅延削減に優れるアルゴリズムを利用できる．

4.2 課題

グローバルデータ構造インタフェースを整備するにあたっては，計算通信オーバーラップを最大化するため，片側通信のプリミティブを用いたアルゴリズムを最大限に使用することが基本方針であるが，その他にも考慮すべき課題がある．本節では 4 つの課題について説明する．

1 つめの課題は，グローバルデータ構造に使用するメモリ領域の割当て，解放メカニズムである．メモリ領域の原資はローカルにのみ存在するため，グローバルデータ構造に使用可能なメモリ領域の割当てには，ローカルなメモリ領域割り当てが先行しなければならない．しかし，エクサスケールの時代では使用メモリ量の削減が重要であり，グローバルデータ構造のためのメモリ領域割当ては固定的ではなく，動的かつ必要最小限としたい．メモリ管理機構をグローバルデータ構造で構築し，ローカルにもグローバルにも利用することで空きメモリ領域を共有することが望ましい．OS レベルのメモリ管理機構をグローバルデータ構造化するのは現実的ではないが，ユーザープロセスで実現可能な範囲内でグローバルデータ構造化を検討すべきである．また，グローバルなメモリ領域割当て，解放は，計算通信オーバーラップの観点からも同期的よりも非同期的である方が望ましく，さらには片側通信のみで成立することが望ましい．

2 つ目の課題は，グローバルデータ構造操作の遅延である．特に新しいデータの挿入，削除はメモリ参照回数が多く，遅延が大きい．書かれたデータが即座に読み出される状況では，挿入，削除の遅延を回避するために，データの書換えが可能であることが重要になる．また，書き込みデータの同期化遅延が小さいことも重要である．

3 つ目の課題は，グローバルデータ構造の一貫性維持である．共有メモリ計算機では様々な種類の排他制御アルゴリズム，ロックフリー並列処理アルゴリズムが研究されている．一貫性維持を考慮するにあたっては，メモリ参照の順序制御の考慮，不可分な操作，特に Atomic Compare and Swap の活用が重要である．ただし分散メモリ計算機のグローバルメモリ参照は遅延が大きく，連続データの転送スループットは比較的高い特徴があるため，分散メモリ計算機特有の最適化が有効である可能性がある．

最後の課題は，データの配置である．データを配置するプロセス，および各プロセスにおけるデータ配置のどちらも重要な考慮点となる．例えば挿入するデータのメモリ領域を自プロセスでローカルに割当てればメモリ割当てに関するオーバーヘッドは小さい．しかし，データ構造全体が複数プロセスに跨るため，以降のグローバルデータ構造の操作のオーバーヘッドは逆に大きくなる可能性がある．また，ポインタ等の制御構造とデータの実体を組み合わせて配置するより，制御構造とデータの実体を分けて配置する方が，連続データ転送に適する可能性がある．

4.3 メッセージ通信への適用の初期検討

本節ではグローバルデータ構造インタフェースの性能特性と課題を抽出するため，現在最も一般的な通信モデルであるメッセージ通信を題材として，データ構造とアルゴリズムを検討する．検討の前提条件として，以下の 4 項目を挙げる．

第 1 の前提条件として，厳しい省メモリ要求を想定する．第 2 の前提条件として，

受信データを基本的に宛先ノードに配置することとする．第 3 の前提条件として，繰り返しデータを送信する起点は対象のメモリ上に専用のメモリ領域を割当てる．割当ては動的に行い，データ送信頻度が下がった時点で速やかに該当メモリ領域が解放されるものとする．第 4 の前提条件として，サイズの大きいデータを転送する起点は対象および起点のメモリ上にメモリ領域を割り当て，データをバッファリングする．第 1 の前提条件はグローバルデータ構造インタフェースの利用を促す根本的な要求である．第 2，第 3 の前提条件は，メッセージ通信性能において重要である受信処理における遅延を削減する．第 4 の前提条件は，計算通信オーバーラップの機会を最大化する．以上の前提条件は初期検討のために簡素化されており，実用的なメッセージ通信の実装には必須の要素をいくつか欠いている．例えば，転送したデータが即座に読み出される場合の低遅延化施策を含んでいない．また，遅延削減等の性能最適化の指針が変われば，前提条件も自ずと異なってくる．

以上の 4 つの前提条件を満たすメッセージ通信実装の概要と，使用するデータ構造を以下に述べる．第 1 および第 2 の前提条件より，各プロセスは個別に固定サイズの受信キューを持ち，複数の起点で共有するものとする．第 3 および第 4 の前提条件を満たすために，各プロセスは自身のメモリ領域上で空の通信バッファを複数構築しておき，グローバルに参照可能なデータ構造で管理してバッファプールとする．各プロセスのバッファプール全体は，該当プロセスのローカルなメモリ上に配置される．第 3 の前提条件を満たすため，頻繁にデータを送信する起点は対象のバッファプールから通信バッファを取り出し，専用の通信バッファとする．専用通信バッファにデータ転送した際の制御情報は対象プロセスの共有受信キューに書き込む．第 4 の前提条件を満たすため，サイズの大きなデータを転送する起点は対象のバッファプールから通信バッファを取り出す．データ格納領域が不足する場合は起点自身のバッファプールからも通信バッファを取り出す．対象及び起点の通信バッファにデータ転送した際の制御情報は，対象プロセスの共有受信キューに書き込む．以上で説明したメッセージ通信実装について，動作中のグローバルデータ構造の様子を概念図で図 1 に示す．図中の 3 ノードはそれぞれ共有受信キューとバッファプールを持ち，図中では左右のノードは中央のノードに対して大きなサイズのデータ送信を実行し，対象と起点の双方で通信バッファを割当ててデータをバッファリングしている．

本節で検討したメッセージ通信実装において，グローバルデータ構造インタフェースを使用するのは(a) 共有受信キューへのメッセージ挿入，(b) バッファプールからの通信バッファ取り出し，(c) バッファプールへの通信バッファの戻しである．(a) (c) で使用するのは対象メモリ内に配置された共有キューへのデータ挿入であり，(b) で使用するのは対象メモリ内に配置された共有キューからのデータ取り出しである．すなわち，本節で検討したメッセージ通信実装の性能特性と課題を検討するには，対象メモリ内に配置された共有キューに対する操作の性能特性を分析すれば良い．共有キューの操作には 2 つのアプローチがある．1 つ目は，キューの制御情報を排他制御して参照するアプローチである．キューの制御情報は，キュー本体のアドレス，サイズ，書き込みポインタ，読み出しポインタで構成される．このアプローチは可変長データの挿入，取り出しに適するが，キュー本体を連続領域に配置する必要があることから，キュー本体の動的なサイズ変更が不可能になる場合がある．2 つ目のアプローチはロックフリーのキューを使用するアプローチである．受信データが格納された通信バッファのキューと，空の通信バッファを格納したキューを操作する．データ長が可変の場合は，空の通信バッファを格納するキューを通信バッファサイズ別に用意するとメモリ仕様効率が上がる．このアプローチは空の通信バッファの追加，削減が比較的容易である．

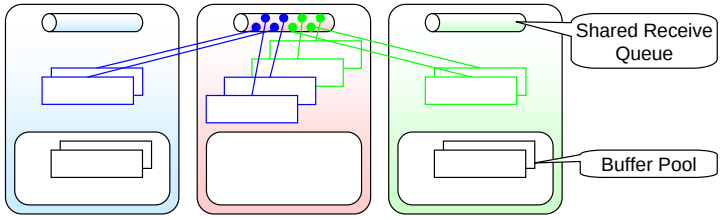

図 1　想定するメッセージ通信実装の概念図

5. 共有キュー操作の性能見積

本章ではグローバルデータ構造インタフェースの共有キュー操作において，排他制御を使用した場合と，ロックフリーキューを使用した場合の性能を評価する．性能見積のために，実機で基本的な通信性能を測定する予備評価を行った．予備評価には富士通の最新スーパーコンピュータ PRIMEHPC FX10 を使用し，片側通信ライブラリは Technical Computing Suite MPI ライブラリの拡張 RDMA インタフェース(FJMPI_Rdma)を使用した．

5.1 予備評価

評価環境は前述の通り PRIMEHPC FX10 である．沼津工場に設置された試験機を使用した．搭載されたプロセッサは SPARC64^{TM} IXfx であり，動作周波数は 1.848GHz，コア数は 16 である．ただし予備評価では 1 ノードあたり 1 コアのみ使用した．インターコネクトは Tofu インターコネクトであり，5GB/s×双方向のネットワーク・インタフェースを 4 つ搭載する．ネットワーク・インタフェースは Put および Get の RDMA 通信機能に対応する．予備評価では 1 ノードあたりネットワーク・インタフェースを

1 つだけ使用した．Tofu インターコネクトのネットワーク・トポロジーは 6 次元メッシュ／トーラスであり，ノードは 6 次元座標（X,Y,Z,A,B,C）で識別される．予備評価ではランク 0 とランク 1 間で通信を行った．全ての予備評価において，ランク 0 とランク 1 は A 軸で隣接となるノードに配置された. 使用した MPI ライブラリは Technical Computing Suite MPI ライブラリ 1.2.0 であり，予備評価では独自の拡張 RDMA インタフェース FJMPI_Rdma を使用した．また，Tofu インターコネクトのハードウェア機能を直接利用する Tofu ライブラリ(tlib)でも通信性能を評価し，FJMPI_Rdma の測定値と比較した．tlib は MPI ライブラリの実装に使用されている，非公開の低レベル・インタフェースである．本評価では，実行条件によって Tofu インターコネクトの通信資源が一部未使用になることを利用し，MPI プログラムから直接 tlib を使用した．

(1) Put レイテンシ

Put を使用してランク 0, ランク 1 間で Ping-Pong 通信を行い，レイテンシを評価した．Put データの到着は，ストロングオーダのオプションを付けた上で末尾のデータをポーリングすることで確認した．合計 1000 往復の Ping-Pong 通信時間を計測し，平均往復遅延時間の 1/2 を Put レイテンシとした．メッセージサイズは 4 バイトから 4M バイトまで変化させた．図 2 に測定結果を示す．最小レイテンシは約 1μ 秒である．FJMPI_Rdma と tlib の測定結果グラフはほぼ重なっており，FJMPI_Rdma の遅延オーバーヘッドは実用上問題ないことを確認した．

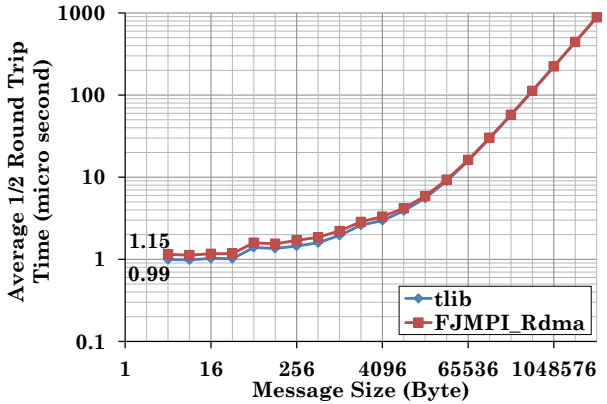

図 2　隣接ノード間の Put レイテンシ

(2) Put スループット

Put を使用してランク 0 からランク 1 へ連続データ転送を行い，スループットを評価した．Put データ転送の完了はランク 0 への応答通知をポーリングすることで確認した．1000 回連続の Put にかかった時間を計測し，総転送量を割った値を Put スループットとした．メッセージサイズは 4 バイトから 4M バイトまで変化させた．図 3 に測定結果を示す．最大スループットは約 5GB/s であった．FJMPI_Rdma は短いメッセージサイズで tlib よりも Put スループットが低いが，8KB 以上のメッセージサイズでは tlib と同等のスループットを実現しており，実用上問題ないことを確認した．

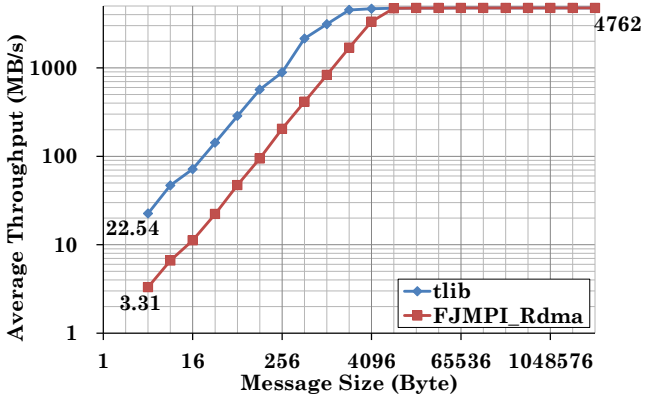

図 3　隣接ノード間の Put スループット

(3) Get レイテンシ（隣接）

Get を使用してランク 0 がランク 1 からデータを取得し，レイテンシを評価した．Get データ転送の完了は，ランク 0 における完了通知をポーリングすることで確認した．データ転送の完了を確認後，再度 Get 通信することを繰り返し，1000 回の Get データ転送にかかった時間を計測して往復遅延時間を Get レイテンシとした．最小レイテンシは約 2μ 秒であり，Put レイテンシのほぼ 2 倍の値となっている．この結果は，遅延の観点では片側通信と両側通信に差がないことを示している．また，Get レイテンシにおいても，FJMPI_Rdma と tlib の測定結果グラフはほぼ重なっている．

(4) Get レイテンシ（ローカル）

Get レイテンシ測定において, 対象を起点と同じランク 0 に変更して計測を行った．この計測では，ネットワーク・インタフェースから送信された Get 要求パケットはルーターで同じネットワーク・インタフェースにループバックし，同じネットワーク・インタフェースが Get 応答パケットを送信する．Get 応答パケットもまたルーターで同じネットワーク・インタフェースにループバックし，最終的に Get データ転送が完

了する．このケースのレイテンシは隣接ノードから Get した場合と比較して，約 0.2 μ秒しか短縮されておらず，ほぼ同等のレイテンシと言える．インターコネクト・ハードウェアの RDMA 通信機能を使用してメモリ参照を行う場合, 参照先がローカルのメモリ領域でも隣接ノードの場合と同等のレイテンシがかかることを確認した．

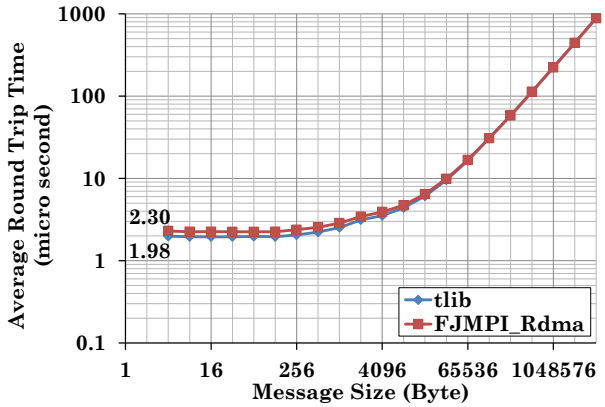

図 4　隣接ノード間の Get レイテンシ

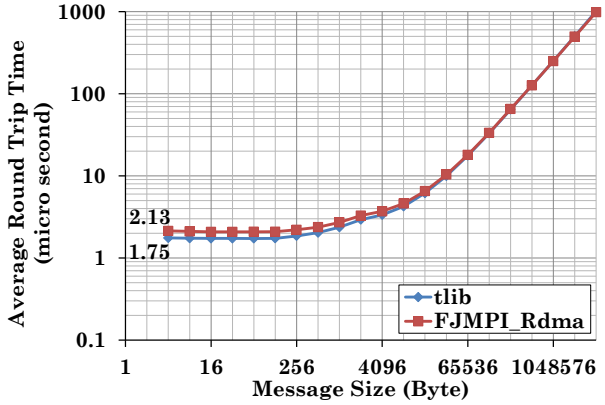

図 5　自ノードに対する Get レイテンシ

5.2 性能見積り

本節では予備評価の測定結果を使用し，共有キュー操作に必要なアルゴリズムの性能を見積る．まず，測定結果より Put レイテンシ 1 μ秒，Get レイテンシ 2 μ秒，スループット 5GB/s とする．Atomic Compare and Swap のレイテンシは，Tofu インターコネクト未対応のため予備評価できなかったが, Get と同等とみなし 2 μ秒とする. また，RDMA 通信ではローカルのメモリを参照する場合でも，レイテンシは短縮されないものとする．

(1) 排他制御: Lamport の Bakery アルゴリズム

Lamport の Bakery アルゴリズム [7]は不可分操作を使用せずに，少ないメモリ参照回数で複数プロセス間の排他制御を行うアルゴリズムである．プロセス数の要素を持つ 2 つの配列を使用して排他制御を行う．一つの配列は待ち行列のシーケンス番号に相当する値を格納する．もう一方の配列は，番号を格納する配列をアクセス中であるかどうかのフラグを格納する．1 回で排他制御が成功する典型的なケースでは，(a) 自プロセスのフラグ書き込み，(b) 番号配列全体の読み出し，(c) 自プロセスの番号書き込み，(d) 自プロセスのフラグ書き込み，(e) フラグ配列全体の読み出しの順でメモリを参照する．さらに(f) 共有キュー制御情報の読み出し，(g) 共有キュー制御情報の書き込み，(h) 排他制御区間から抜ける際の自プロセスの番号書き込みもオーバーヘッドとなる．ここで性能見積りのため，どちらの配列も 64 ビット整数を要素とし，全体のプロセス数は 100 万と仮定する．配列のサイズは 8M バイトとなる．

完了するまで順序が保証されない RMA 通信モデルでのオーバーヘッドを見積ると (a) 2 μ秒，(b) 1600 μ秒，(c) 2 μ秒，(d) 2 μ秒，(e) 1600 μ秒 (f) 2 μ秒 (g) 2 μ秒 (h) 2 μ秒，合計 3212 μ秒となる．一方，Tofu インターコネクトのような順序を保証する RDMA 通信機能でのオーバーヘッドを見積ると, (a)(b) 1600 μ秒, (c)(d)(e) 1600 μ秒 (f) 2 μ秒，(g)(h) 1 μ秒で，合計 3203 μ秒となる．Lamport の Bakery アルゴリズムによる排他制御では制御情報配列を転送するスループットが支配的であり，メモリ参照の順序制御ありなしは制御オーバーヘッドに大きな影響を与えない．

(2) 排他制御: Atomic Compare and Swap

Atomic Compare and Swap で排他制御を行い，1 回目で成功した場合，必要なメモリ参照はロック変数への 1 回の Atomic Compare and Swap である．共有キュー制御情報の読み出し，書き込み，排他制御区間から抜ける際のロック変数への書き込みもオーバーヘッドである．オーバーヘッドの合計は順序を保証しない場合で合計 8 μ秒である．順序を保証する場合，ロック変数への Atomic Compare and Swap と制御情報の読み出しを連続で行うことが可能で 2 μ秒，制御情報とロック変数への書き込みも連続で行え，ロック変数への書込みが同期化するまで 1 μ秒であるので，オーバーヘッドの合計は 3 μ秒となる．Atomic Compare and Swap で排他制御を行う場合，順序保証によってオーバーヘッドの大幅な短縮が可能である．

(3) **ロックフリーキュー**

ロックフリーキューは Head ポインタと Tail ポインタで制御されるリストにデータを蓄積するデータ構造である．Atomic Compare and Swap で操作するために，空の状態でも the only node と呼ばれるダミーの 1 データがリストにリンクされている．新しいデータをリストの Tail 側に挿入され，データの取り出しは Head 側から行われる．

データの取り出し操作が 1 回で成功する典型的なケースにおけるメモリ参照は，(a) Head ポインタの読み出し，(b) Head ポインタが指すデータの next を読み出し，(c) (b) で読み出した next ポインタが指すデータを読み出し，(d) Head ポインタに Atomic Compare and Swap である．データの挿入操作が 1 回で成功する典型的なケースにおけるメモリ参照は，(e) Tail ポインタの読み出し，(f) Tail が指すデータの next ポインタに Atomic Compare and Swap，(g) Tail ポインタへの Atomic Compare and Swap である．

ロックフリーキューを使用する場合，まず空の通信バッファを格納したキューから通信バッファを取り出し，データ転送を行った後で，使用済み通信バッファを格納するキューに該当通信バッファを挿入する．よって，1 回の共有キュー操作は内部的には 2 つのロックフリーキューに対する取り出しと挿入操作で構成される．ロックフリーキューの操作は基本的に応答が必要なメモリ参照で構成され，順序保証のありなしはオーバーヘッドに影響しないため，(a)〜(g)の合計オーバーヘッドは 14 μ 秒になる．

5.3 考察

Atomic Compare and Swap のないインターコネクト・ハードウェアで Put/Get のみを使用して排他制御を行う場合，オーバーヘッドはレイテンシではなく制御配列転送のスループットによって支配的される．100 万プロセス規模ではミリ秒オーダーのオーバーヘッドがかかる．また制御配列はプロセス数と等しい要素数が必要であり，グローバルデータ構造ライブラリの機能として一般的に使用することは難しい．

一方，Atomic Compare and Swap があるインターコネクト・ハードウェアでは 1 つの排他制御にロック変数 1 つしか使用せず，共有キュー操作のオーバーヘッドは合計 8 μ 秒程度と大幅に削減される．順序保証のあるインターコネクト・ハードウェアならば，さらに 3 μ 秒程度まで削減される．Atomic Compare and Swap によってロックフリーキューも実現できるが，動的な使用メモリ量制御が可能などの長所があるものの，共有キュー操作のオーバーヘッドは合計 14 μ 秒程度と排他制御方式に比べて大きい．

6. まとめと今後の課題

本稿では内部的に片側通信を使用し，複数プロセスのメモリに配置されたグローバルデータ構造を操作するライブラリの構築を提案した．効率的な並列アルゴリズムをライブラリとして提供することにより，プログラマは複雑な並列アルゴリズムの実装から解放される．また，グローバルデータ構造ライブラリの構築における 4 つの課題，メモリ領域の割当て・解放メカニズム，グローバルデータ構造操作の遅延，グローバルデータ構造の一貫性維持，データの最適配置について説明した．

さらにメッセージ通信実装でのグローバルデータ構造の使用を想定して性能見積りおよび分析を行った．性能見積では基本通信性能値として富士通の最新スーパーコンピュータ PRIMEHPC FX10 の実機測定結果を使用した．Put/Get のみを使用して排他制御を行うと 100 万プロセス規模ではミリ秒オーダーのオーバーヘッドがかかるが，Atomic Compare and Swap を使用すればオーバーヘッドが約 8 μ 秒に削減され，メモリ参照の順序が保証されればオーバーヘッドが約 3 μ 秒まで削減されることが分かった．

今後は共有キュー操作について実機での性能評価，競合時の性能評価等を実施して分析を深めるとともに，メッセージ通信でデータが即座に使用される場合の最適化や，PGAS 言語の実装で必要とされるグローバルデータ構造および操作の検討，評価，分析を進め，グローバルデータ構造ライブラリの適切な仕様を明らかにする．

参考文献

1) Ajima, Y., Inoue, T., Hiramoto, S., Shimizu, T., Takagi, Y.: The Tofu Interconnect. IEEE Micro, vol. 32, no. 1, pp. 21-31 (2012).
2) Ajima, Y., Sumimoto, S., Shimizu, T.: Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers. IEEE Computer, vol. 42, no. 11, pp.36-40 (2009).
3) ARMCI, http://www.emsl.pnl.gov/docs/parsoft/armci/.
4) GASNet, http://gasnet.cs.berkeley.edu/.
5) InfiniBand Trade Association, http://www.infinibandta.org/.
6) Kogge, P.M., Dysart, T.J.: Using the TOP500 to Trace and Project Technology and Architecture Trends, In Proceedings of SC ‘11, article 28, 11 pages, ACM (2011).
7) Lamport, L.: A New Solution of Dijkstra’s Concurrent Programming Problem, Communication, vol. 17, no. 8, pp. 453-455, ACM (1974).
8) Michael, M.M. and Scott, M.L.: Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms. In Proceedings of PODC ‘96, pp. 267-275, ACM (1996).
9) PRIMEHPC FX10, http://www.fujitsu.com/global/services/solutions/tc/hpc/products/primehpc/.
10) Technical Computing Suite, http://www.fujitsu.com/global/services/solutions/tc/hpc/products/primehpc/software/middleware/.
11) The Message Passing Interface (MPI) standard, http://www.mcs.anl.gov/research/projects/mpi/.