Vol.2014-HPC-143 No.9
2014/3/3

ACP 基本層の設計思想とインタフェース

安島雄一郎[†1,†2]　佐賀一繁[†1,†2]　野瀬貴史[†1,†2]　三浦健一[†1,†2]　住元真司[†1,†2]

Advanced Communication for Exa (ACE)プロジェクトにおいて，我々は Advanced Communication Primitives (ACP)ライブラリを開発している．ACP は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コレクションを中核とするライブラリである．チャネルおよびグローバルデータ構造コレクションは ACP 基本層の上に，ポータブルに実装される．本稿では ACP 基本層の設計思想とインタフェース仕様を紹介する．ACP 基本層は，全プロセスのメモリを 64 ビット空間に割付けるグローバルアドレス空間モデルと，リモートプロセス間のメモリツーメモリ転送によりデータ移動を最小化するグローバルメモリ参照インタフェースを備える．

The Concept and Design of the ACP Basic Layer Interface

YUICHIRO AJIMA[†1,†2]　KAZUSHIGE SAGA[†1,†2]
TAKAFUMI NOSE[†1,†2]　KENICHI MIURA[†1,†2]　SHINJI SUMIMOTO[†1,†2]

We are developing the Advanced Communication Primitives (ACP) library in the Advanced Communication for Exa (ACE) project. The ACP library comprises two core interfaces; the channel interface and the global data structure collection. The channel interface is designed to optimize the memory consumption for stream data transfer, and the global data structure collection is designed to optimize inter-process data placement. For portable implementations of core interfaces, the ACP library has a basic layer. In this paper, we introduce the design philosophy and interface specification of the ACP basic layer. The global memory model allocates memory areas of any process into a single 64-bit address space, and the global memory access interface minimizes inter-process data movement by remote memory to remote memory transfer.

1. はじめに

Advanced Communication for Exa (ACE) プロジェクトではプロセッサのメニーコア化が進むエクサスケールの時代に向けて，プロセスあたりの消費メモリ量を抑制しつつ，低遅延通信を実現する通信ソフトウェア技術の創出に取り組んでいる．我々は ACE プロジェクトの目標を実現する中核技術として，低レベル通信ライブラリ Advanced Communication Primitives (ACP)を開発している[1].

従来のメッセージパッシングライブラリは暗黙的にメモリを消費し，通信遅延増加に繋がるメモリ使用量抑制には消極的な実装となっている．また，Remote Direct Memory Access (RDMA)ハードウェアはメモリを明示的に使用するが，プロセス間メモリ管理方式が標準化されていないのでポータビリティの確保に Partitioned Global Address Space (PGAS)言語等の利用が必要である．しかし PGAS 言語のメモリ確保は基本的に同期かつ斉一であり，省メモリアルゴリズムの導入余地は小さい．

ACP ライブラリではメモリ消費量の明示的な制御のために，通信の基本機能を根本的に見直した．ACP の中心となる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コレクションである．チャネルおよびグローバルデータ構造コレクションは低メモリ消費ながらデータ転送のオーバヘッドが小さい高度なプロトコルおよびアルゴリズムを提供する．このような複雑な処理はポータブルに実装することが望ましいので，我々は通信ハードウェアを抽象化する ACP 基本層を導入する．

本稿では ACP 基本層の設計思想とインタフェース仕様を紹介する．以降では，2 章で ACP 基本層の設計思想，構成および関数定義を述べる．3 章で関連研究を紹介し，4 章で今後の課題を挙げ，最後に 5 章でまとめる．

2. ACP 基本層

2.1 設計思想

ACP 基本層はエクサスケール時代に相応しいインターコネクトデバイス抽象化層の機能を定義する．エクサスケールの実現には性能あたりの消費電力削減が課題であり，特にデータ移動消費電力の削減が重要である．また，エクサスケール時代に主流となるハードウェア実装技術との親和性も重要となる．

従来のインターコネクトデバイスは自ノードと他ノード間でデータを転送するリモートメモリ参照(Remote Memory Access, RMA)機能を有する．RMA は片側通信とも呼ばれ，読出し側もしくは書き込み側のプロセッサがデータ転送を制御する必要がある．RMA を使用して複数プロセス間のデータ配置を変更する場合，データ転送を最小化する通信パターンによって通信を制御するプロセスが限定

†1 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
†2 独立行政法人科学技術振興機構　戦略的創造研究推進機能
Japan Science and Technology Agency (JST),
Core Research for Evolutional Science and Technology (CREST)

される．これは不要な同期待ちによる性能低下を招く原因となる．解決策の一つとして専用ハードウェアによる非同期集団通信が挙げられるが，ポータビリティの確保が難しく基本層には適さない．そこで我々はグローバルメモリ参照(Global Memory Access , GMA)を提案，導入する．GMA は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ツーメモリコピーを制御する通信機能である．GMA は複数プロセス間の複雑な通信パターンを非同期化し，通信遅延および同期待ち時間の隠蔽を可能にする．

エクサスケール時代の主流技術は System on Chip であり，インターコネクトデバイスはプロセッサと同じチップに実装される．SoC 実装はノード内データ移動電力を削減するだけでなく，従来のインターコネクトデバイスの機能を制約していた標準 IO バスが無くなるため，プロセッサとインターコネクトデバイスの緊密な連携を可能にする．例えば，プロセッサのロードストア命令と通信機能間でのメモリ階層バイパスや，不可分メモリ参照の相互不可分性保証などである．このような技術革新は従来インタフェースの低遅延化にも役立つが，それだけでは潜在能力の活用として不十分である．プロセッサとインターコネクトデバイスのより緊密な連携を実現するために，我々は 64 ビットのフラットなアドレス空間で全プロセスのメモリを参照するグローバルメモリモデルを導入する．現在のプロセッサ及び処理系はアドレスおよび参照を 64 ビットで処理しておりエクサスケール時代も変わらないと想定されるため，64 ビットアドレスはプロセッサ，処理系と親和性が高い．また 64 ビットアドレスは 16 エクサバイトをアドレッシング可能であり，エクサスケール時代のグローバルアドレス空間として十分な大きさがある．

異種のインターコネクトデバイス間には機能差が存在する．そこで様々なインターコネクトデバイス上に ACP 基本層を実装する場合でもメモリ使用量，通信性能両面で大きなペナルティを与えないことを考慮する．例えば，プロセス間メモリ管理機能は RMA で実装可能かつ必要最小限の仕様とする．また，現在のインターコネクトデバイスが有する様々なアドレス変換機構を有効に利用するために遅延一括メモリ登録の仕組みを導入する．また ACP 基本層は低オーバヘッドのソフトウェア実装が可能なだけでなく，ハードウェア化が容易で将来のインターコネクトデバイスによるサポートが期待できる仕様とする．

2.2 ACP 基本層の構成

ACP 基本層は，初期化，終了処理や総プロセス数，プロセス番号取得などの共通機能を提供するインフラストラクチャ関数，メモリの登録やグローバルアドレス変換を行うグローバルメモリ管理関数，グローバルアドレスを指定してメモリを参照するグローバルメモリ参照関数で構成される．以降では関数カテゴリごとにインタフェース仕様を説明する．

また本稿では詳しく説明しないが，ACP 基本層はチャネルインタフェースやグローバルデータ構造コレクションなどを初期化，終了処理するための内部インタフェースや，ACP を下位層として MPI や PGAS 言語が実装される場合に，ユーザーによる ACP ライブラリの直接使用を可能にするための仮想デバイスインタフェースも備える．

2.3 インフラストラクチャ関数

表 1 にインフラストラクチャ関数の定義を示す．ACP ライブラリを使用するには，最初に各プロセスが初期化関数を呼ぶ必要がある．初期化関数以外の全ての ACP 関数は，初期化後に呼ぶ必要がある．初期化関数は全プロセスで通信に必要なリソースが確保し，通信が可能な状態になると戻る．初期化関数には main 関数の引数へのポインタを渡す．これは，ACP 実行環境が各プロセスの main 関数引数に ACP ライブラリの初期化に必要な情報を挿入することを想定している．初期化関数は ACP 実行環境が挿入した引数を削除して戻る．よって，ユーザーが指定した main 関数の引数は ACP 初期化後に解釈されなければならない．ACP ライブラリを使用するアプリケーションは，終了前に終了処理関数を呼ぶ必要がある．終了処理関数は全プロセスが終了処理に入ったことを確認し，使用していたリソースを解放する．

ACP ライブラリは初期化後に再初期化を行うことができる．再初期化関数は初期化直後の状態に戻すだけでなく，各プロセスのランク番号を変更することができる．ランク番号の変更機能により，プログラム本体を変更することなく通信パターンをネットワークトポロジーに適合させることができる．タスク実行中にプロセスが回復不可能な状態に陥った場合，そのプロセスは ACP の強制終了関数で終了させることが望ましい．強制終了では他のプロセスの終了を待ち合わせることなく使用していたリソースを解放する．

表 1　インフラストラクチャ関数

Table 1　Infrastructure functions

名称	定義
初期化	int **acp_init**(int* *argc*, char*** *argv*);
終了処理	int **acp_finalize**(void);
再初期化	int **acp_reset**(int *rank*);
強制終了	void **acp_abort**(const char* *str*);
全プロセス同期	int **acp_sync**(void);
プロセスランク取得	int **acp_rank**(void);
総プロセス数取得	int **acp_procs**(void);

表 2　グローバルメモリ管理関数
Table 2　Global memory management functions

名称	定義
スターターアドレス取得	acp_ga_t **acp_query_starter_ga**(int *rank*);
メモリ登録	acp_atkey_t **acp_register_memory**(void* *addr*, size_t *size*, int *color*);
メモリ登録解除	int **acp_unregister_memory**(acp_atkey_t *atkey*);
グローバルアドレス取得	acp_ga_t **acp_query_ga**(acp_atkey_t *atkey*, void* *addr*);
論理アドレス取得	acp_ga_t **acp_query_address**(acp_ga_t *ga*);
ランク番号取得	int **acp_query_rank**(acp_ga_t *ga*);
カラー番号取得	int **acp_query_color**(acp_ga_t *ga*);
最大カラー数取得	int **acp_colors**(void);

また，インフラストラクチャとして全プロセス同期関数が提供される．全プロセス同期はアプリケーションからの使用だけでなく，ACP 基本層の上位に構築されるソフトウェアスタックの初期化，終了処理からも利用されることが想定されている．

最後に，ACP の共通機能としてタスクに含まれる総プロセス数を返す関数と，該当プロセスのプロセスランク番号を返す関数が提供される．

2.4 グローバルメモリ管理関数

グローバルメモリ管理関数は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．グローバルアドレスを取得するには，事前にメモリの登録が必要である．登録するメモリ領域の先頭アドレスとサイズを指定してメモリ登録関数を呼び出すと，メモリ登録関数はアドレス変換キーを返す．すなわち，メモリを登録した時点では論理アドレスからグローバルアドレスへの変換は行われない．論理アドレスからグローバルアドレスへの変換は，グローバルアドレス取得関数が呼び出されるまで遅延される．グローバルアドレス取得関数はアドレス変換キーと論理アドレスを引数とし，グローバルアドレスを返す．ACP 基本層は近傍のメモリ領域が続けて登録されると内部でメモリ領域をマージし,同じアドレス変換キーを返す．我々はこの仕組みを遅延一括メモリ登録と呼ぶ．一般にハードウェアによるアドレス変換資源は有限であるので，遅延一括メモリ登録はアドレス変換資源を節約する．

メモリ登録の際にはメモリ領域の定義の他に，カラー番号を指定できる．カラー番号は複数のネットワークインタフェースが使用可能である際に，使用するネットワークインタフェースを振り分けるヒントとして使用される．カラー番号には 0 から最大カラー数-1 までの数値を指定する．最大カラー数は最大カラー数取得関数で取得できる．カラー番号はメモリ登録の際に指定し，グローバルアドレスにカラー情報が含まれる．そのため，グローバルメモリ参照関数の引数にはカラー番号指定はない．

アドレス変換機構へのメモリ登録を解除してグローバルアドレスを無効にするには，アドレス変換キーを指定してメモリ登録解除関数を呼び出す．同じアドレス変換キーが複数回発行されている場合，アドレス変換機構は該当アドレス変換キーの解除回数が発行回数と同数になるまで実際のメモリ登録解除を遅延する．

論理アドレスからグローバルアドレスへの変換は各プロセスでローカルに行われるため，あるプロセスが登録したグローバルメモリを他のプロセスが参照するには，プロセス間でグローバルアドレスを受け渡す手段が必要である．ACP 基本層では初期化後に最初にグローバルアドレスを受け渡す共有変数の配置場所として，グローバルメモリ参照が可能なスターターメモリを用意する．ACP 基本層は初期化時に各プロセスで一定サイズのスターターメモリを確保し,そのグローバルアドレスを全プロセス間で共有する．各プロセスはプロセスランク番号を指定してスターターアドレス取得関数を呼び出すことで，任意のプロセスのスターターメモリのグローバルアドレスを取得することができる．スターターメモリのサイズは環境変数もしくは ACP タスク起動時のオプションで指定する．

また，ACP 基本層のグローバルメモリ管理機構はグローバルアドレスから論理アドレス，プロセスランク番号，カラー番号を逆変換する機能も提供する．論理アドレスは論理アドレス取得関数，プロセスランク番号はランク番号取得関数，カラー番号はカラー番号取得関数で取得する．

2.5 グローバルメモリ参照関数

グローバルメモリ参照(GMA)関数はグローバルメモリ上でのコピーおよび不可分メモリ参照を行う．コピー関数は転送元と転送先に任意のグローバルアドレスを指定できる．コピーの読出しと書き込みは順不同に行われ，重複して読出し，書き込みが行われる場合もある．不可分メモリ参照でも転送元と転送先のグローバルアドレスを指定する．不可分メモリ参照は転送元で 1 回だけ行われ，不可分メモリ参照の読出し結果が転送先に書き込まれる．結果は複数回重複して書込まれる場合がある．不可分メモリ参照は 4 バイトまたは 8 バイトのデータ幅に対応し,演算は比較交換，交換，加算，排他的論理和，論理和，論理積に対応する．

GMA 関数は非ブロッキングであり，GMA の開始を待たずに GMA ハンドルを返す. GMA はバックグラウンドで実行される. GMA の終了は GMA 完了関数で待ち合わせることができる．ここで GMA の開始とは転送元において読み出し又は不可分メモリ参照を開始することであり，GMA

の終了とは転送先へ書込みが終了することであり，終了後は重複書き込みも行われない．GMA 完了関数は指定 GMA ハンドルおよびそれ以前の GMA が全て終了まで待機する．GMA 完了関数を呼び出した時に即座に戻るか待機するかを事前に知るには，GMA 照会関数を使用する．GMA 照会関数は指定 GMA ハンドルおよびそれ以前の GMA がすべて終了しているかどうかを表す値を返す．

GMA 関数はさらに，実行順序を指定することができる．GMA 関数呼び出し時に実行順序を GMA ハンドルで指定すると，該当 GMA は指定 GMA ハンドルおよびそれ以前の GMA が全て終了した後に開始される．ここで順序制御しない場合は GMA ハンドルに ACP_HANDLE_NULL を，先行する全ての GMA の終了を待ち合わせる場合は GMA ハンドルに ACP_HANDLE_ALL を指定する．

ここで GMA 関数のサンプルコードを示す．以下はスターターメモリ上で非ブロッキング，インプレイスの n バイト AllGather 集団通信を行う例である．

```
rank = acp_rank();
procs = acp_procs();
handle[rank] = ACE_HANDLE_NULL;
for ( i = 1 ; i < procs ; i++ ) {
    dst = (rank +  i    ) % procs;
    src = (rank + (i>>1)) % procs;
    handle[dst] = acp_copy(
        acp_query_startar_ga(dst) + n * rank,
        acp_query_startar_ga(src) + n * rank,
        n,
        handle[src]
    );
}
acp_complete(ACP_HANDLE_ALL);
acp_sync();
```

上記サンプルコードの for ループでは自プロセスのデータを二分木で全プロセスに Broadcast している．送信元でも送信先でもないプロセスが転送を制御する GMA を使用することで，元データを持つプロセスが Broadcast に必要な通信制御を全て行っている．また，各プロセスが受信したデータを送信する順序は GMA ハンドルで指定されている．このサンプルコードでは最終 2 行まで完了および同期待ちが無く，各プロセスが Broadcast を非ブロッキングに実行している．この結果，AllGather 通信全体が非ブロッキングとなっている．また，グローバルアドレスの計算は 64 ビット整数の計算として簡潔に記述されている．

ACP 基本層の不可分メモリ参照は任意のグローバルアドレスに対して実行できる．ただしデータ幅が 4 バイトならばデータが 4 バイト境界，データ幅が 8 バイトならばデータが 8 バイト境界に整列している必要がある．また，ACP 基本層の不可分メモリ参照はエクサスケール時代を想定してプロセッサの不可分メモリ参照と相互に不可分性を保証する．現在のインターコネクトデバイスは任意のアドレスに対する，プロセッサ命令と相互に不可分性を保証する不可分メモリ参照機能を持たない．よって，現在のハードウェアで動作する ACP 基本層実装では，プロセッサ命令によって不可分メモリ参照を実行する必要がある．

GMA 完了関数および GMA 照会関数では，終了した GMA のエラーハンドリングも行われる．ただし，コピー関数内もしくは不可分メモリ参照関数内で GMA の管理資源が不足している場合や，終了処理(acp_finalize)関数内でも GMA の終了待ち合わせとエラーハンドリングが行われるので，GMA 完了関数の呼び出しは必須ではない．なお GMA でエラーが発生すると，ACP 基本層はほとんどの場合で該当プロセスを強制終了する．

表 3　グローバルメモリ参照関数
Table 3　Global memory access functions

名称	定義
コピー	acp_handle_t **acp_copy**(acp_ga_t *dst*, acp_ga_t *src*, size_t *size*, acp_handle_t *order*);
4 バイト不可分比較交換	acp_handle_t **acp_cas4**(acp_ga_t *dst*, acp_ga_t *src*, uint32_t *oldval*, uint32_t *newval*, acp_handle_t *order*);
8 バイト不可分比較交換	acp_handle_t **acp_cas8**(acp_ga_t *dst*, acp_ga_t *src*, uint64_t *oldval*, uint64_t *newval*, acp_handle_t *order*);
4 バイト不可分交換	acp_handle_t **acp_swap4**(acp_ga_t *dst*, acp_ga_t *src*, uint32_t *value*, acp_handle_t *order*);
8 バイト不可分交換	acp_handle_t **acp_swap8**(acp_ga_t *dst*, acp_ga_t *src*, uint64_t *value*, acp_handle_t *order*);
4 バイト不可分加算	acp_handle_t **acp_add4**(acp_ga_t *dst*, acp_ga_t *src*, uint32_t *value*, acp_handle_t *order*);
8 バイト不可分加算	acp_handle_t **acp_add8**(acp_ga_t *dst*, acp_ga_t *src*, uint64_t *value*, acp_handle_t *order*);
4 バイト不可分排他的論理和	acp_handle_t **acp_xor4**(acp_ga_t *dst*, acp_ga_t *src*, uint32_t *value*, acp_handle_t *order*);
8 バイト不可分排他的論理和	acp_handle_t **acp_xor8**(acp_ga_t *dst*, acp_ga_t *src*, uint64_t *value*, acp_handle_t *order*);
4 バイト不可分論理和	acp_handle_t **acp_or4**(acp_ga_t *dst*, acp_ga_t *src*, uint32_t *value*, acp_handle_t *order*);
8 バイト不可分論理和	acp_handle_t **acp_or8**(acp_ga_t *dst*, acp_ga_t *src*, uint64_t *value*, acp_handle_t *order*);
4 バイト不可分論理積	acp_handle_t **acp_and4**(acp_ga_t *dst*, acp_ga_t *src*, uint32_t *value*, acp_handle_t *order*);
8 バイト不可分論理積	acp_handle_t **acp_and8**(acp_ga_t *dst*, acp_ga_t *src*, uint64_t *value*, acp_handle_t *order*);
GMA 完了	void **acp_complete**(acp_handle_t *handle*);
GMA 照会	int **acp_inquire**(acp_handle_t *handle*);

3. 関連研究

3.1 Universal Common Communication Substrate

Universal Common Communication Substrate (UCCS) [2]は米国オークリッジ国立研究所 Extreme Scale System Center (ESSC)が開発中のネットワークインタフェース抽象化ライブラリである. UCCS は Active Message, RDMA PUT/GET, Atomic, Collectives, Run-Time Environment の API を有する. 全ての通信 API は非ブロッキングであり，Collectives API には Cheetah Framework [3]を使用している．RDMA PUT/GET は short, large, medium で API が分かれており，通信モデルだけでなく実装プロトコルを意識した利用が想定されている．また put scatter，get gather など，メッセージレートの低いインターコネクトデバイスのオーバヘッドを隠蔽するための API も用意されている．UCCS の不可分メモリ参照である Atomic API は Fetch, Fetch and Add, Increment, SWAP, CSWAP の演算を提供する.

2014 年 1 月現在 UCCS のインタフェース仕様詳細は未公開だが，OpenSHMEM のリファレンス実装に組み込まれる予定であるので, 近日中に公開されると予想される. なお，現在公開されている OpenSHMEM v1.0e では下位層に GASNet [4]を使用している.

3.2 Parallel Active Messaging Interface

Parallel Active Messaging Interface (PAMI) [5][6]は IBM Blue Gene/Q [7]向けに開発された低レベル通信ライブラリである．PAMI 上に実装された ARMCI, MPI, UPC などの複数の上位層インタフェースを同時に使用できるようにするため，Client という仕組みを導入している．PAMI を使用するプログラムは最初に PAMI_Client_create サブルーチンを呼び出して Client 識別子を取得し，PAMI サブルーチンを呼び出す際は取得した Client 識別子を引数として渡す．PAMI サブルーチンは基本的に Client 識別子とパラメータ構造体へのポインタを引数とする.

PAMI サブルーチンの通信機能は Send, Put, Get, Rmw, Collective, Active Message である. Rmw は Read-modify-write の略であり，不可分メモリ参照を意味する．不可分メモリ参照の演算はセット，加算，論理和，論理積，排他的論理和であり, これらの演算に読出し値を返すかどうか(Fetch), 比較操作を行って一致した場合だけ実行するかどうか(Compare)のフラグを組み合わせる.

PAMI はバックグラウンドで通信プロトコルを進捗させるための通信スレッド機能を有する．当初，PAMI の通信スレッド機能はシングルスレッド時には 1.95 μ 秒だった MPI Ping-Pong 片道遅延を 8.7 μ 秒に増加させるなど，大幅な性能劣化を引き起こした．これは従来の Blue Gene では MPI のマルチスレッド対応のために MPI の全関数でグローバルロックを取得していたことが原因であった．そこで Blue Gene/Q の MPI ではマルチスレッド対応において排他制御粒度を縮小し，ロックフリー技術を利用した．この結果，PAMI が通信スレッド機能を使用する場合でも MPI Ping-Pong 片道遅延は 3.25 μ 秒まで改善した.

3.3 User-level Generic Network Interface

User-level General Network Interface (uGNI) [8]は Cray の Gemini および Aries インターコネクト向けに開発された低レベル通信インタフェースである．uGNI の通信は論理エンドポイント間で行われる．GNI_EpCreate 関数で生成された直後のエンドポイントは任意のエンドポイントとのデータグラム交換は可能だが，それ以外の通信はできない．エンドポイントは GNI_EpBind 関数で特定のリモートアドレスにバインドすると RDMA, Fast Memory Access (FMA), FMA Short Messaging (SMSG), Shared Message Queue (MSGQ)通信が可能になる．RDMA は連続データの Put, Get 通信，FMA は 64 バイトまでの Put, Get 通信と 8 バイトの Atomic Memory Operation (AMO)通信をサポートする．RDMA および FMA 通信は Gemini および Aries インターコネクトの通信機能をそのまま抽象化したものである．SMSG および MSGQ 通信は受信バッファを割り当て行う通信であり，利用には GNI_SmsgInit 関数もしくは GNI_MsgqInit 関数の呼び出しが必要になる．SMSG および MSGQ のデータ転送には Gemini および Aries インターコネクトの FMA 通信機能が使用される.

AMO は不可分メモリ参照であり，Compare and Swap, Fetch and Add, Fetch and Or, Fetch and Xor, Fetch and Masked Xor, Add, Or, Xor, Masked Xor の演算がサポートされる．Gemini および Aries インターコネクトのハードウェアは AMO キャッシュを有し，同じアドレスが繰り返し不可分メモリ参照される場合のスループットを向上する．しかしプロセッサが該当アドレスを参照しても AMO キャッシュ上の最新の値は読み出されない．これは標準の IO バスで接続されているために，インターコネクトがプロセッサのキャッシュとコヒーレントなキャッシュを持つことができないことが原因である.

4. 今後の課題

ACP 基本層に関しては現在 ACE プロジェクトで UDP 版, IB verbs 版，Tofu lib 版の開発を進めており[9]，ACP 基本層の現在のインターコネクトデバイス上での実装における課題解決に取り組んでいる．現在のインターコネクトデバイス上での実装方式の研究は，将来のインターコネクトデバイスで ACP 基本層をサポートする機能を検討する際にも役立つと考えられる.

課題の 1 つを挙げると，現在のインターコネクトデバイスの RMA 通信は一般的にノードアドレス，メモリ識別子，オフセットなど複数レベルのアドレスを必要とするので，ACP 基本層の 64 ビットグローバルアドレスに全ての情報は収まらない．この課題に対しては，何らかのアルゴリズ

ムでビット幅を圧縮する，全プロセスが変換テーブルのコピーを持ち RMA で更新する，各プロセスが変換テーブルを持ち RMA で相互に参照する，64 ビットグローバルアドレスに収まる範囲のメモリ識別子しか発行しないようにデバイスドライバを修正するなどのアプローチが可能である．

また別の課題として，遅延一括メモリ登録の実装方式も検討が必要である．ACP 基本層仕様が想定する，グローバルアドレス取得関数内でメモリを登録するユーザーライブラリレベル実装の他に，ランタイムによるメモリ割当て時にインターコネクトデバイスにもメモリを登録してしまうランタイムレベル実装, 論理アドレスと 64 ビットグローバルアドレスを一致させる OS レベル実装など，様々なレベルのアプローチが可能である．

今後の ACE プロジェクトに関しては，ACP ライブラリの基本層以外の部分が主な研究課題となる．過去の研究でグローバルデータ構造コレクションの基盤要素技術として非同期グローバルヒープの検討[10]などを行ってきたが，今後 ACP ライブラリ仕様として具体化する．具体化に際しては，基本層として実装すべき機能が無いか等の見直しが必要である．

5. まとめ

我々は Advanced Communication for Exa (ACE)プロジェクトにおいて，Advanced Communication Primitives (ACP)ライブラリを開発している．ACP は省メモリと低遅延を実現する通信プリミティブを提供するライブラリである．本稿では ACP ライブラリのうち, 基本層の設計思想とインタフェース仕様を紹介した．ACP 基本層は高度な通信プロトコル，アルゴリズムをポータブルに実装するためのハードウェア抽象化層である．

ACP 基本層の特徴は，エクサスケール時代に適したインターコネクトデバイスを想定した仕様である．System on Chip 実装によりプロセッサとインターコネクトデバイスが緊密に連携することを想定し，ACP 基本層が扱うグローバルアドレスは 64 ビット幅とした．また，エクサスケールの実現にはデータ移動の最小化による消費電力削減が重要であるので，データ移動最適化の自由度を上げるために，どのプロセスからでも任意のプロセス間のメモリコピーが可能なグローバルメモリ参照通信を導入した．

参考文献

1) 住元 真司, 安島 雄一郎, 佐賀 一繁, 三浦 健一, 野瀬 貴史, 高見 利也, 南里 豪志: エクサスケール通信向け ACP スタックの設計思想, 情報処理学会研究会報告 2014-HPC-143-8 (2014).
2) UCCS - Universal Common Communication Substrate: http://uccs.github.io/uccs/
3) Richard L. Graham, Pavel Shamis, et al.: Cheetah: A Framework for Scalable Hierarchical Collective Operations, IEEE/ACM CCGrid 2011, pp.73-83 (2011).
4) GASNet Communication System: http://gasnet.lbl.gov/
5) Sameer Kumar, Amith R. Mamidala, el al.: PAMI: A Parallel Active Message Interface for the Blue Gene/Q Supercomputer, IEEE 26th IPDPS, pp. 764-774 (2012).
6) PAMI Programming Guide Version 1 Release 1.0: http://publib.boulder.ibm.com/epubs/pdf/a2322730.pdf
7) The IBM Blue Gene Team: The Blue Gene/Q Compute Chip, HOT CHIPS 23 (2011).
8) Using the GNI and DMAPP APIs: http://docs.cray.com/books/S-2446-3103/S-2446-3103.pdf
9) 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, 情報処理学会研究会報告 2014-HPC-143-10 (2014).
10) 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, 情報処理学会研究会報告 2013-HPC-138-10 (2013).