Vol.2015-HPC-151 No.9
2015/9/30

ACP グローバルメモリ管理アルゴリズムの改善

安島雄一郎[†1,†2]　野瀬貴史[†1,†2]　佐賀一繁[†1,†2]　志田直之[†1,†2]　住元真司[†1,†2]

本論文では Advanced Communication for Exa (ACE)プロジェクトで開発している Advanced Communication Primitives (ACP)ライブラリにおける，グローバルメモリアロケータのメモリ管理アルゴリズムを改善する．既存実装では各プロセスの空きグローバルメモリをアドレスでソートした片方向リスト，フリーリストで保持しており，メモリ割当は典型的には計算量が O(1)で高速である一方，解放は隣接する空き領域をマージするため，計算量が O(*n*)と空きメモリ領域数に比例していた．ACP では使用しないメモリを解放することによる省メモリ化を推奨しているので，メモリ解放のコストが高いことは望ましくない．そこでメモリ解放時はフリーリストではなく隣接領域を直接調べて空き領域であるかどうか判定するアルゴリズムに改良し，計算量を O(1)に削減した．この際，メモリ割当の計算量は O(1)を維持するだけでなくメモリ転送回数を元のアルゴリズムと同等に抑えるために，空き領域の遅延マージを導入した．実装したアルゴリズムを UDP 版，Tofu 版および Tofu2 版の ACP 基本層を使用して評価し，メモリ割当関数の実行時間を維持しつつ，メモリ解放関数の実行時間をメモリ割当関数と同等に削減したことを示した．提案したグローバルメモリ管理アルゴリズムは，使用しないグローバルメモリの解放による省メモリ化を推奨する ACP に適したデータ構造とアルゴリズムであることが確認できた．

1. はじめに

ポストペタスケール時代の High Performance Computing (HPC)では，メニーコア・プロセッサと広帯域の三次元積層メモリがキーテクノロジとして期待されている．しかしながら広帯域の三次元積層メモリは容量が既存のメモリモジュールと同程度の水準に留まるため，今後のメニーコア化の進展に伴ってコアあたりのメモリ容量が減少することが課題である．ポストペタスケール時代のシステムソフトウェアにおいてはメモリ消費量の削減が重要となる．

従来のメッセージパッシング型の通信ライブラリでは，通信バッファは自動的に割当てられ，解放せずに割り当てられ続ける．これは解放，再割当てとも処理コストが高いためである．メモリ使用量削減のためには通信バッファを解放，再割当てすべきであるが，アプリケーション性能の劣化も回避すべきである．この問題を解決するには，利用者がアプリケーションの通信パターンを深い理解し，明示的にメモリ消費量を制御する必要がある．

Advanced Communication for Exa (ACE) プロジェクト[1]ではプロセスあたりのメモリ消費量を抑制しつつ，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，利用者がメモリ消費量を意識したプログラミングが可能であるように，明示的にメモリを使用するインタフェースを備える低レベル通信ライブラリ Advanced Communication Primitives (ACP)を開発している[2].

ACP は低レベル通信を抽象化し，インターコネクトデバイスの違いを吸収する基本層[3]と，基本層の上にポータブルに実装される中間層で構成される．中間層にはコミュニケーションライブラリおよびデータライブラリ[4]という 2 つのサブライブラリが含まれる．コミュニケーションライブラリは既存のメッセージパッシングで記述されるアプリケーションの移行を想定したインタフェースであり，明示的な通信バッファ割当・解放が可能なチャネルインタフェースを中核とする．データライブラリは PGAS 言語ランタイム，ビッグデータ処理などの新しい用途を想定したインタフェースであり，大域的なデータ配置を最適化するための分散動的データ構造インタフェースを中核とする．

本論文では ACP データライブラリの中核であるグローバルメモリアロケータのメモリ管理アルゴリズムを改良し，メモリ解放の実行時間を短縮する．また，提案したアルゴリズムを実装評価する．以降では，2 章で ACP の基本層とデータライブラリ，既存のグローバルメモリ管理アルゴリズムを紹介し，3 章で新しいグローバルメモリ管理アルゴリズムを提案する．4 章で評価，5 章で今後の課題について記述し，最後に 6 章でまとめる．

2. Advanced Communication Primitives

2.1 ACP 基本層

Partitioned Global Address Space (PGAS)プログラミングモデルは，メモリアドレスを明示的に指定してデータを転送する Remote Direct Memory Access (RDMA)ハードウェアの通信機能を抽象化する．しかし RDMA ハードウェアのプロセス間メモリ（グローバルメモリ）管理方式は標準化されていないので，既存実装では言語ランタイムで RDMA ハードウェア仕様を隠蔽し，並列言語仕様として PGAS プログラミングモデルを組み込むものが多い．これに対して ACP では直接グローバルメモリ管理をインタフェース化する．ポータビリティ確保のために利用が煩雑になるため，アプリケーションプログラマの直接使用を想定しない低レベルのインタフェース層と位置付け，基本層と名付ける．

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

†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)

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データを転送するグローバルメモリ参照関数で構成される．代表的なインフラストラクチャ関数，グローバルメモリ管理(GMM)関数，グローバルメモリ参照(GMA)関数の定義を表 1，表 2，表 3 に示す．

GMM 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．グローバルアドレス取得関数はアドレス変換キーとローカルの論理アドレスを引数とし，グローバルアドレスを返す．ACP 基本層ではプロセッサとインターコネクトデバイス間で不可分メモリ参照を介して低遅延で同期するために，グローバルアドレスを 64 ビット整数とする．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る．登録したメモリ領域はグローバルアドレス空間でも連続であることが保証されるので，登録したメモリ領域内の一ヶ所のグローバルアドレスを取得すれば，残りの部分のグローバルアドレスを計算可能である．アドレス変換機構へのメモリ登録を解除するには，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．

ACP 基本層では複数プロセスに跨って連続したメモリ領域を定義できない．また，CPU が直接参照可能なのは該当プロセスで登録されたメモリだけである．このようなグローバルアドレス空間は Partitioned Global Address Space (PGAS)に分類される．アドレス変換キーはメモリを登録したプロセスでのみ有効であり，論理アドレスとグローバルアドレスの相互変換は各プロセスでローカルに行われる．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必要がある．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．

GMA 関数は任意のグローバルアドレスを送信元および宛先に指定してプロセス間でデータをコピーする．また，プロセス間で同期するための不可分参照もサポートする．GMA と仮想共有メモリ(VSM)の違いは，GMA は専用のグローバルアドレス空間を扱うが，VSM は CPU から直接参照可能な仮想メモリ空間をグローバルアドレスとして扱う点である．

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);
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	void* **acp_query_address**(acp_ga_t *ga*);

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*);
GMA 完了	void **acp_complete**(acp_handle_t *handle*);
GMA 照会	int **acp_inquire**(acp_handle_t *handle*);

2.2 ACP データライブラリ

ACP データライブラリ(ACPdl)は ACP 基本層の上に構築される分散データ構造のライブラリである．ACPdl はデータ構造を操作するアルゴリズム自体を変えずに，配置指定の変更だけでグローバルなデータ配置の最適化を可能にすることを目標とする．そのため，データ生成時に配置を明示的に制御することで，データ構造を複数プロセスに分散させる．データの生成，操作，破棄は非同期に，配置するプロセスと同期せずに行う．さらに，データがローカルに配置されている場合，通常のローカルなデータ構造の操作と比較して遜色のない性能を目指す．インタフェースは，上位層の言語処理系などでラップされることを想定して，データ構造間でインタフェースの直交性が高い仕様とした．データ構造の型は C++言語の標準テンプレートライブラリ(Standard Template Library, STL)[5]を参考にして，これまでにベクタ，リスト[4]および連想配列[6]を導入した．

2.3 グローバルメモリアロケータ

グローバルメモリ割当関数 acp_malloc およびグローバルメモリ解放関数 acp_free の定義はほぼ C 言語の標準 C ライブラリの malloc，free 関数と同様であるが，acp_malloc 関数にはメモリ割り当てを行うプロセス番号の指定がある点が異なる．割り当てたメモリ量などの管理情報は利用者からは隠されて管理され，割り当てたメモリを解放する際に使用される．

リモートプロセスのメモリ上に動的にデータを生成するためにはリモートプロセスのメモリを確保する必要がある．しかし基本層のインタフェースでグローバルアドレスを取得するために登録できるメモリはローカルプロセスのメモリだけである．ここで非同期グローバルヒープ[7]を使用すれば，リモートプロセスのプロセッサを介在せずにメモリを確保できる．非同期グローバルヒープとは各ノードが空きメモリを登録しておき，他プロセスが端から順に空き領域を取得できるようにしたデータ構造である．非同期グローバルヒープの割当済み領域と空き領域の境界はブレークポインタで示され，ブレークポインタを操作することでメモリの割当，解放を行う．しかし 1 つの非同期グローバルヒープから複数プロセスがメモリを取得した場合，メモリの解放が困難になる問題がある．

そこで ACPdl では，より高度なメモリ管理アルゴリズムの実装を想定し，グローバルメモリアロケータを導入する．グローバルメモリアロケータではブレークポインタの操作でメモリ割当，解放を行うのではなく，メモリ割当と解放に別関数を用意する．表 4 にグローバルメモリアロケータ関数の定義を示す．インタフェース関数の引数は C 言語の標準ライブラリに含まれる malloc，free と同様であるが，メモリ割当関数にプロセスランク番号の引数が追加されている点が異なる．

従来の ACPdl 実装では，空き領域をアドレスでソートされた片方向リストで管理する Kernighan and Richie (K & R) [8]のアルゴリズムを採用している．メモリ管理のデータ構造と操作例を図 1 に示す．空き領域は循環フリーリストでアドレス順に連結され，フリーリストの検索先頭ポインタが別に記録される．空き領域，割当済み領域とも先頭にサイズを格納し，空き領域はサイズに続いてフリーリストのポインタを格納する．

図 1(a)の状態からメモリ領域を 1 つ割当てた後の状態が図 1(b)である．先頭ポインタは移動しているが空き領域の数は増えていない．また，割当に成功した空きメモリ領域は再度割当成功する可能性が高いので，検索ポインタは割当てた残りの空きメモリ領域を指すように更新される．このようにしてメモリ割当に要する典型的な計算量は O(1)となる．

図 1(b)の状態からメモリ領域を 1 つ解放した後の状態が図 1(c)である．新しくメモリ領域を解放する場合，まずアドレス順のフリーリストを検索して挿入するべき位置を探す．次に，フラグメンテーション回避のために連続する空き領域はマージする必要があるので，前後の空き領域と新しい空き領域が連続しているか調べる．連続していればマージして新しい空き領域とし，連続していなければ単独の空き領域としてリストに挿入する．新しく解放されたメモリ領域は再割当される可能性が高いので，割当のコストを下げるために検索ポインタは新しい空きメモリ領域を指すように更新される．ここでフリーリスト検索の計算量は空き領域数に比例するので，O(n)とコストが高い．ACP では使用しないメモリの解放による省メモリ化を推奨しているので，高コストなメモリ解放は問題である．

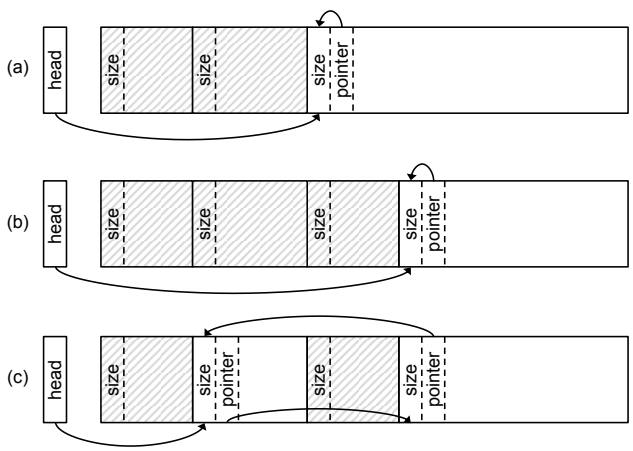

図 1　従来のグローバルメモリ管理アルゴリズムのデータ構造と操作の例

Figure 1　Examples of the data structure and operations of the existing global memory management algorithm

表 4　グローバルメモリアロケータ関数
Table 4　Global memory allocator functions

名称	定義
グローバルメモリ割当	acp_ga_t **acp_malloc**(size_t *size*, int *rank*);
グローバルメモリ解放	void **acp_free**(acp_ga_t *ga*);

3. 新アルゴリズムの提案

従来の ACPdl に実装されたグローバルメモリ管理アルゴリズムではメモリ解放のコストが高い問題があった．そこで本論文では，メモリ解放時はフリーリストではなく隣接領域を直接調べて空き領域であるかどうか判定し，計算量を O(1)に削減するアルゴリズムを提案する．

新しいグローバルメモリ管理構造を図 2 に示す．空き領域，割当済み領域とも先頭だけでなく末尾にサイズを記録し，さらにサイズは 8 バイトの倍数であることから使用されていない下位 3 ビットに領域の種類を格納した．この管理構造により，新しい空き領域を追加する際は前後の領域が空き領域か直接調べることにより空き領域のマージが可能になる．なお，割当て可能領域の両端に 0 バイトの割当済み領域を示す番兵を置き，判定を単純化している．

図 2(a)の状態から 1 つのメモリ領域を割当てた後の状態が図 2(b)である．新しい管理構造ではフリーリストのポインタを領域の先頭から末尾に移し，なおかつポインタは次の領域の先頭ではなくポインタを直接指すものとした．これにより空き領域の一部を割当てた際もポインタの位置は不動となり，直前のポインタを付け替える必要がなくなる．また，空き領域を連結するフリーリストはアドレス順にソートせず，さらに循環せずに終端するものとした．割当に成功した空き領域の残りはリストの元の位置から削除され，先頭に挿入される．新しい管理構造においても，メモリ割当に要する典型的な計算量は O(1)であり，コストも旧アルゴリズムと同等である．

図 2(b)の状態から 1 つのメモリ領域を解放した後の状態が図 2(c)である．前後の領域を直接調べて，どちらも空き領域でなければフリーリストの先頭に挿入する．後ろの領域が空き領域であれば，後ろの領域にマージする．新しい管理構造では前方に領域が拡張される際はポインタの位置が変わらないので，後ろの空き領域のサイズを変更するだけで処理が完了する．後ろの領域が空き領域でなく，前の領域が空き領域である場合，フリーリストに連結されない拡張空き領域として処理を完了し空き領域のマージは割当時に発見されるまで遅延する．この遅延マージアルゴリズムにより解放時のフリーリストの検索を回避し，メモリ解放に要する典型的な計算量を O(1)に削減する．

遅延マージアルゴリズムでは割当時に空き領域を検索する際，後ろの領域も調べる必要がある．新しい管理構造ではリストポインタ，空き領域のサイズ，後続領域のサイズは連続して配置されるので，1 回のデータ転送で読み出すことができ，後ろの領域を調べることはコスト増にはならない．ただし，割当時に空き領域の次が拡張空き領域ではなく通常の空き領域が見つかった場合，該当空き領域をリストから削除して後ろの空き領域にマージし，フリーリストの検索を先頭からやり直すため，メモリ割当のコスト増要因となる．このような連続した空き領域は，新しくメモリ領域を解放する際に前後とも空き領域だった場合に残される．

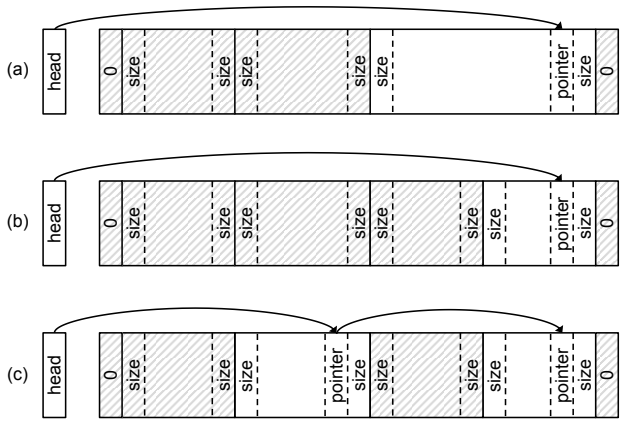

図 2　提案するグローバルメモリ管理アルゴリズムのデータ構造と操作の例
Figure 2　Examples of the data structures and operations of the proposed global memory management algorithm

4. 評価

提案アルゴリズムを評価するため，オリジナルのアルゴリズムと提案アルゴリズムでグローバルメモリ割当関数 acp_malloc および解放関数 acp_free の実行時間を計測した．acp_malloc については 100 回連続で呼び出し，平均実行時間を求めた．割当サイズは 1 バイト以上 32768 バイト以下の乱数を使用し，毎回変更した．割当てた 100 個のグローバルメモリは acp_free で解放し，その平均実行時間も求めた．解放する順番は割当て順ではなく，ランダムに定めた．

計測プログラムは 2 プロセスで実行し，acp_malloc の割当先のプロセスはローカルとリモートの 2 通りを評価した．ローカルプロセスが割当先の場合，インターコネクトを介した通信は発生せず，全て CPU で処理される．

4.1 評価環境

評価は PC クラスタとスーパーコンピュータ，計 3 機種で実施した．表 5，表 6 および表 7 に評価環境の詳細を示す．PC クラスタのインターコネクトは Gigabit Ethernet であり，UDP[9]版の ACP 基本層[3]を使用して評価した．スーパーコンピュータ 1 のインターコネクトは Tofu インターコネクト[10][11]であり，Tofu 版の ACP 基本層[12]を使用して評価した．スーパーコンピュータ 2 のインターコネクトは Tofu インターコネクト 2[13]であり，Tofu2 版の ACP 基本層[14]を使用して評価した．

表 5　評価環境 1: PC クラスタ
Table 5　Evaluation environment1: PC Cluster

Node	Fujitsu PRIMERGY RX200 S5
CPU	Intel Xeon E5520 (4 cores, 2.27 GHz), 2 sockets
Memory	DDR3 SDRAM 48GB, 51.2 GB/s
Network	Gigabit Ethernet (125 Mbyte/sec)

表 6　評価環境 2: スーパーコンピュータ 1
Table 6　Evaluation environment2: supercomputer

Node	Fujitsu Supercomputer PRIMEHPC FX10
CPU	Fujitsu SPARC64^{IM} IXfx (16 cores, 1.848 GHz)
Memory	DDR3 SDRAM 64GB, 85 GB/s
Network	Tofu interconnect (5.0 Gbyte/sec)

表 7　評価環境 3: スーパーコンピュータ 2
Table 7　Evaluation environment3: supercomputer2

Node	Fujitsu Supercomputer PRIMEHPC FX100
CPU	Fujitsu SPARC64^{IM} XIfx (34 cores, 1.975 GHz)
Memory	HMC 32GB, 480 GB/s
Network	Tofu interconnect 2 (12.5 Gbyte/sec)

4.2 評価結果

図 3 にグローバルメモリアロケータ関数の実行時間評価結果を示す．割当先ローカルプロセスの acp_malloc 関数と acp_free 関数，割当先リモートプロセス acp_malloc 関数と acp_free 関数の平均実行時間が棒グラフで示されている．それぞれの実行時間は新旧アルゴリズムと UDP 版，Tofu 版，Tofu2 版 ACP 基本層の組合せ計 6 種類が計測された．

acp_malloc 関数ではローカル，リモート，UDP，Tofu，Tofu2 のいずれの組合せでも新旧アルゴリズムの平均実行時間がほぼ同等であった．このことから，提案アルゴリズムによって割当コストが増加していないことが確認できた．

acp_free 関数ではローカル，リモート，UDP，Tofu，Tofu2 のいずれの組合せでも新アルゴリズムの平均実行時間は旧アルゴリズムからほぼ半減した．具体的には UDP 版はローカルが約 23 μ 秒から約 10 μ 秒，リモートが約 0.89m 秒から約 0.40m 秒，Tofu 版はローカルが約 22 μ 秒から約 10 μ 秒，リモートが約 58 μ 秒から約 29 μ 秒，Tofu2 版はローカルが約 27 μ 秒から約 12 μ 秒，リモートが約 59 μ 秒から約 24 μ 秒に削減された．

また新アルゴリズムに注目すると，ローカル，リモート，UDP, Tofu, Tofu2 いずれの場合も acp_malloc 関数と acp_free 関数の平均実行時間がほぼ同等になった．具体的には UDP 版はローカルが約 12 μ 秒と約 10 μ 秒，リモートが約 0.42m 秒と約 0.40m 秒，Tofu 版はローカルが約 10 μ 秒と約 10 μ 秒，リモートが約 31 μ 秒と約 29 μ 秒，Tofu2 版はローカルが約 12 μ 秒と約 12 μ 秒，リモートが約 24 μ 秒と約 24 μ 秒であった．

以上の結果から，提案したグローバルメモリ管理アルゴリズムによりグローバルメモリ解放のコストをグローバルメモリ割当と同等に削減できた．提案したグローバルメモリ管理アルゴリズムは，使用しないグローバルメモリの解放による省メモリ化を推奨する ACP に適したデータ構造とアルゴリズムであることが確認できた．

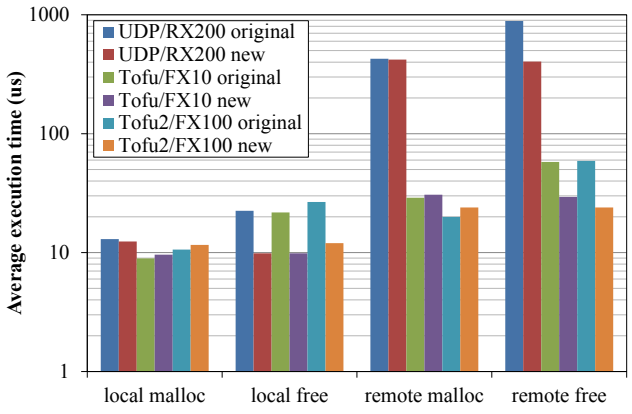

図 3　グローバルメモリアロケータ関数の平均実行時間評価結果
Figure 3　Evaluation results of average execution time of global memory allocator functions

5. 今後の課題

ローカルの場合でも関数実行時間は 10 μ 秒前後に達している．これは現状の ACP 基本層の実装では，データの転送元，宛先ともにローカルプロセスであった場合でも ACP の通信スレッドがデータ転送を処理するためであり改善の余地がある．また，通信スレッドのコマンドキューがスレッドセーフになっている点もオーバーヘッドを増やしていることが確認されており[15]，ACP のスレッドモデルについても検討の余地がある．

著者らは非同期グローバルヒープの提案[7]において，RDMA とアトミック通信のリモート側での順序制御が可能であれば通信遅延の隠ぺいにより，ヒープ操作関数の実行時間を半減できることを示した．しかし，現在のグローバルメモリアロケータ実装ではリモート側での順序制御は

行っていない. これは ACP 基本層の仕様としてはリモート側順序制御を定義してあるが，現在の実装は UDP 版，Tofu 版，Tofu2 版とも送信元で順序を保証する簡易実装になっているためである．今後 UDP 版，Tofu 版，Tofu2 版 ACP 基本層でリモート側順序制御が実装されれば，グローバルメモリアロケータ関数の実行時間が半減することが期待される．

6. まとめ

本論文では ACE プロジェクトで開発している ACP ライブラリにおける，グローバルメモリアロケータのメモリ管理アルゴリズムを改善した．既存実装では各プロセスの空きグローバルメモリをアドレスでソートした片方向リスト，フリーリストで保持しており，メモリ割当は典型的には計算量が O(1)で高速である一方，解放は隣接する空き領域をマージするため，計算量が O(n)と空きメモリ領域数に比例していた．ACP では使用しないメモリを解放することによる省メモリ化を推奨しているので，メモリ解放のコストが高いことは望ましくない．そこでメモリ解放時はフリーリストではなく隣接領域を直接調べて空き領域であるかどうか判定するアルゴリズムに改良し，計算量を O(1)に削減した．この際，メモリ割当の計算量は O(1)を維持するだけでなくメモリ転送回数を元のアルゴリズムと同等に抑えるために，空き領域の遅延マージを導入した．実装したアルゴリズムを UDP 版，Tofu 版および Tofu2 版の ACP 基本層を使用して評価し，メモリ割当関数の実行時間を維持しつつ，メモリ解放関数の実行時間をメモリ割当関数と同等に削減したことを示した．提案したグローバルメモリ管理アルゴリズムは，使用しないグローバルメモリの解放による省メモリ化を推奨する ACP に適したデータ構造とアルゴリズムであることが確認できた．

参考文献

1) ACE Project, http://ace-project.kyushu-u.ac.jp/index.html
2) 住元真司, 安島雄一郎, 佐賀一繁, 野瀬貴史, 三浦健一, 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)
3) 安島雄一郎, 佐賀一繁, 野瀬貴史, 三浦健一, 住元真司: ACP 基本層の設計思想とインタフェース, 情報処理学会研究会報告 2014-HPC-143-9 (2014)
4) 安島雄一郎, 野瀬貴史, 佐賀一繁, 志田直之, 住元真司: ACP の分散動的データ構造インタフェース, 情報処理学会研究会報告 2014-HPC-146-18 (2014)
5) Containers library, http://en.cppreference.com/w/cpp/container
6) 安島雄一郎, 野瀬貴史, 佐賀一繁, 志田直之, 住元真司: ACP の分散連想配列インタフェース, 情報処理学会研究会報告 2014-HPC-148-26 (2015)
7) 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)
8) Brian W. Kernighan and Dennis M. Ritchie: The C Programming Language (2nd edition), Prentice Hall (1988)
9) Jonathan B. Postel (editor): User Datagram Protocol, RFC 768 (1980)
10) Yuichiro Ajima, et al: Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers, Computer, 42(11), pp. 36-40 (2009)
11) Yuichiro Ajima, et al.: The Tofu Interconnect, Micro, 32(1), pp. 21-31 (2012)
12) 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)
13) Yuichiro Ajima, et al.: The Tofu Interconnect 2, 22nd IEEE Annual Symposium on High-Performance Interconnects, pp. 57-62 (2014)
14) 野瀬貴史, 安島雄一郎, 佐賀一繁, 志田直之, 住元真司: Tofu インターコネクト 2 上での ACP 基本層の実装と性能評価, 情報処理学会研究会報告 2015-HPC-148-32 (2015)
15) 野瀬貴史, 安島雄一郎, 佐賀一繁, 志田直之, 住元真司: ACP ライブラリの性能最適化に関する検討, 情報処理学会研究会報告 2015-HPC-150-39 (2015)