Vol.2016-HPC-153 No.31
2016/3/3

ACP ライブラリによる MPI_Comm_spawn の置き換えと OpenFMO への適用

本田 宏明[1,3,a)]　森江 善之[1,3]　南里 豪志[1,3]　稲富 雄一[2,3]　高見 利也[1,3]

概要：Advanced Communication for Exa (ACE) プロジェクトでは，将来のエクサスケール環境でも省メモリでありスケーラブルに利用可能な Advanced Communication Primitives (ACP) ライブラリを開発中である．本研究ではこの ACP ライブラリを利用し複数 MPI プログラムを接続する方法を提案実装した．次に MPI_Comm_spawn 関数等を利用する MPI の動的プロセス管理機構を利用して実装されている OpenFMO プログラムにこの ACP ライブラリを利用する方法を適用可能であることを示した．さらに複数 MPI プロセスグループを MPI ならびに ACP ライブラリにて接続し，全体のマスタプロセスから全てのにワーカプロセスに 4 バイト Bcast するテストプログラムを作成し，全 33 プロセスのテスト環境にて全てのプロセス上のヒープ領域メモリ消費量を測定した．その結果， MPI の動的プロセス管理機構を利用する全体のマスタプロセスで 11.8MB と最大のヒープ領域メモリ使用量となった．これに対し ACP によって MPI プロセスグループを接続した場合の全体マスタプロセスの使用量は 2.32MB, ワーカルートプロセスは 10.8MB であった．その結果，本研究で対象とした小規模な通信の場合にも ACP を利用することでの優位性が現れることが分かった．

1. はじめに

現在，ポストペタスケールやエクサスケールクラスのスーパーコンピュータに向けた研究開発が進められている．エクサスケールクラスコンピュータでは，おおよそ 10^6 ～ 10^7 プロセスの計算が可能な超高並列環境となり，プロセスあたりに利用可能となるメモリサイズは 10GB 以下となるのでは，との予想がされている．これに対し，大規模並列アプリケーションの多くで利用されている MPI ライブラリは，1. Unexpected Message 処理，2. 通信用と管理制御用のバッファ，3. 他のデバイス非依存ならびにデバイス依存バッファのそれぞれの内部実装の特徴により，通信遅延の点で実用的な MPI ライブラリ実装を利用する場合では各プロセスにおける MPI ライブラリの要求メモリ量は 10^6 プロセス数で 10GB を越えると試算されている．そのため，起動プロセスの増加とともにいずれプログラムの実行が困難になるのでは，との指摘がなされている [1]–[2].

この問題に対し，Advanced Communication for Exa (ACE) プロジェクトでは，エクサスケール環境においてもスケーラブルな性能を持つ通信ライブラリを目指し省メモリな Advance Communication Primitives(ACP) ライブラリを開発中である [3]．この ACP ライブラリは RDMA に基づく片側通信技術を採用し，メモリアロケーションについてはライブラリ使用者に明示することで，省メモリ化を実現している．ライブラリは，ハードウエア依存部分を吸収する Basic Layer ならびに Basic Layer を利用した Middle Layer から構成され，すでにリリース版が公開されており，UDP や Infiniband, Tofu, Tofu2 のネットワーク環境にて利用可能となっている [4]-[10].

一部の超高並列向け MPI アプリケーションではプログラム中のコミュニケータを分割してサブコミュニケータとし，複数のサブコニュニケータにそれぞれ異るタスクを割り振り計算する場合がある．また，MPI が提供する MPI_Comm_spawn 関数ならびに付随する一連の関数からなる動的プロセス管理機構を利用し，全体プログラムが内部にて複数種類の MPI_COMM_WORLD からなる MPI プロセスグループを複数個束ねて動作するよう実装されている場合がある．例として，超高並列向け量子化学計算計算プログラム OpenFMO では，既に MPI サブコミュニケータ実装と動的プロセス管理による 2 種類の実装が存在している [12], [13].

[1] 九州大学 情報基盤研究開発センター
RIIT, Kyushu University
[2] 九州大学 システム情報科学研究院
ISEE, Kyushu University
[3] 独立法人 科学技術振興機構 戦略的創造研究推進事業
JST-CREST
[a)] honda.hiroaki.971@m.kyushu-u.ac.jp

このサブコニュニケータを利用する場合ではプログラム全体の MPI_COMM_WORLD の利用が必要なため，上述のメモリ使用量の問題が発生すると考えられる．しかしながら，動的プロセス管理機構を利用する場合では，MPI プロセスグループ内通信とプロセスグループ間通信が独立に実施されると考えられるためにメモリ使用量が低減されるのではないか，と期待される．しかしながら既存の MPI 実装のメモリ使用に関する実装では，プロセスグループ間通信についてもメッセージパッシングに基づいており，通信バッファを利用した暗黙的アロケーションが行われるために，どの程度のメモリ使用量となるかが明らかではないといえる．これに対し，複数の MPI プロセスグループを ACP により接続することにより，RDMA による通信を行うことで少なくとも複数 MPI プロセスグループ間の通信については必ず省メモリとすることが可能であると期待される．

そこで本研究では，複数 MPI プロセスグループの ACP ライブラリによる接続の方法を提案し，初期実装ならびにヒープメモリ使用量測定を行なうことを目的とした．

2 節では ACP ライブラリの全体構成ならびに複数のプロセスの接続方法について説明する．3 節では ACP ライブラリを利用した複数 MPI プロセスグループの接続方法について説明する．4 節では本方法の OpenFMO プログラムへの適用可能性について検討する．5 節では本方法と MPI の動的プロセス管理機構方法とのヒープメモリの使用量評価実験について説明し，最後にまとめる．

2. ACP 通信ライブラリ

2.1 ライブラリ概略

図 1 に全体構成を示す ACP 通信ライブラリは，エクサスケールの超高並列環境においてもスケーラブルな性能を持つ通信ライブラリを目指して開発が進めてられており，通信に関わるメモリ使用量を低く抑えることを目標としている．これまで広く用いられている MPI ライブラリでは，MPI_Init 関数呼び出し後では，どのタイミングにおいても Send/Recv や集団通信による通信が可能である．これに対し我々の ACP ライブラリでは，通信に必要な通信元ならびに通信先情報の管理やバッファ領域等の確保ならびに解放をライブラリ使用者側に明示させることで，ライブラリによる暗黙のメモリ使用量を抑えている．

現状では Basic Layer と Middle Layer の 2 層で構成されており，下層に位置している Basic Layer では Ethernet や Infiniband，Tofu, Tofu2 の通信ネットワークデバイスを抽象化している．また，RDMA かつ片側通信のモデルに基づいており，大規模並列計算機環境の全ての分散メモリを単一のグローバルメモリとして表現することで，プロセス間の通信の送信先と送信元を指定可能としている．グローバルメモリ上でユニークに決定されるグローバルアドレス (GA) を指定することにより，各種ネットワークデバイスにおける片側通信ならびにアトミックな処理を可能としている．この GA はそれぞれのデータを管理するプロセス上にて，Basic Layer が提供するローカルメモリ領域の登録関数により取得可能である．

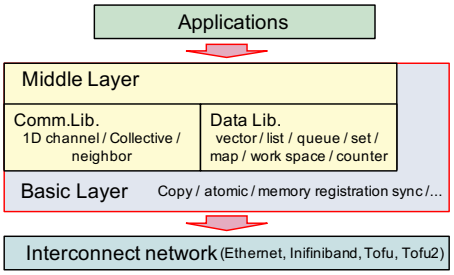

図 1　ACP 通信ライブラリ全体構成

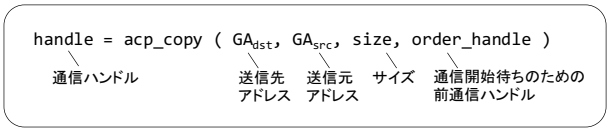

図 2　ACP_copy 関数

図 2 に GA 間のデータコピーに対応する基本的な関数を示す．この関数以外にも各種の atomic 通信関数が用意されている．データの送信は non-blocking を基本としており，データ送信終了の情報を保持するハンドルを関数の返り値として持つ．また，関数の引数として，データ送信元 GA，データ送信先 GA，通信サイズ，通信開始待ちのための前通信ハンドルを指定する．

一方 Middle Layer は，Communication Library と Data Library の 2 種類からなり，現在までのところ，Communication Library ではメッセージパッシング型のチャネルインタフェース [8] と集団通信が可能な命令列型通信 [9], [10]，Data Library では分散メモリ環境にて C++ の STL に似た機能を提供するデータストラクチャ [5], [6] と Global Arrays に似たワークスペース，グローバルカウンタライブラリが開発されている．

2.2 ACP プログラムの起動方法

ACP ライブラリを利用するプログラムは，MPI の mpirun プログラムに対応する acprun プログラムによっても実行可能であり，ACP ライブラリに同梱され配布されている．図 3 に acprun に実装されている 32 プロセスの場合の ACP タスクの起動方法を示す．図に示す様に，ssh コマンドにより複数の ACP プログラムのリモートプロセス起動を行ない，その際にランク番号やランク数，ポート番号，通信先プロセスの IP アドレスを環境変数 (もしくはプログラム引数) により指定する．起動された各プロセスでは指定された情報に従い，ACP 初期化関数である acp_init 関数内にて他プロセスとの接続を行なう．また，計算機セ

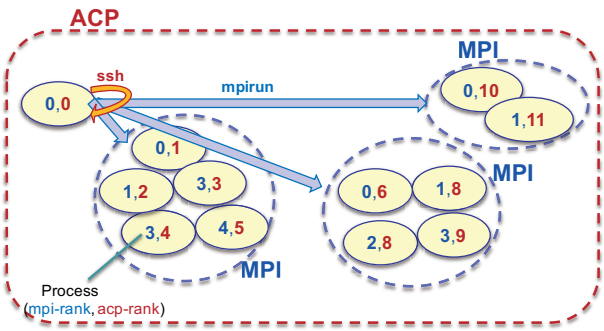

図 4　複数 MPI プログラムの接続例

ンターのバックエンド等の環境では ssh の代替コマンドを利用することも可能である．

3. ACP 通信ライブラリによる 複数 MPI プログラムの接続

3.1 方法の概略

図 4 に，本研究で提案する複数 MPI プロセスグループを ACP により接続する場合の例を示す．最初のプロセスから，ACP のみのプロセスグループを ssh により起動する．この後，MPI と ACP が両方利用可能な複数のプロセスグループは，MPI の mpirun を複数回利用することにより起動される．mpirun により起動された MPI プロセスグループはそれぞれ MPI_COMM_WORLD を持ち，プロセスグループ内にてランク番号とプロセス数が取得可能である．これらのプロセスグループは同時に ACP ライブラリも利用しており，ACP のランク番号とプロセス数も保持しており，全てのプロセスにおける通番に対応している．このランク番号は MPI のランク番号に MPI プロセスグループのプロセス数をオフセット値とする和として求められる．また，ACP のみのプロセスグループの個数は 0 もしくは 1 であり，0 の場合には全てのプロセスを MPI プロセスグループのみとすることも可能である．

3.2 複数 MPI プログラムの接続方法

本方法では，前節で説明したように複数の MPI プロセスグループを起動したのち，これらの MPI プロセスグループを ACP により接続する．図 5 に，全 12 プロセスにて，ACP のみ 1 プロセス，MPI プロセスグループ数 3，それぞれ 5,4,2 のプロセス数とする場合のプログラム起動方法を示す．1 行目は ACP のみのプログラムを 1 プロセス起動している．もしも，残りの 11 プロセスを ACP のみにより接続するのであれば，節 2.2 にて説明した ssh にてプロセスを起動する方法を用いる．しかしながら，本方法では MPI プロセスグループの起動をする必要があるため，mpirun を複数回利用している．その際に，ジョブスケジューラから取得するジョブ起動ノード情報ファイルを予めプロセス数に分割して Nodefile.1~3 に保存しておき，これを参照する．この mpirun にて起動されたプロセスを ACP ライブラリにより接続するため，ユーザの MPI プログラム内の MPI_Init 関数呼び出し直後に acp_init 関数を呼び出しておく．acp_init にて他プロセスと接続するためには，ローカルプロセスのポート番号ならびに接続先プロセスの IP アドレス情報とポート番号が必要であるため，図の mpi_prog の引数にこれらの情報を予め保存してあったファイル名 “Portfile” を指定している．acp_init 関数は ユーザプログラムの MPI_Init 関数後に呼び出されているため，MPI_Comm_rank 関数が利用可能であり，MPI のランク番号と mpi_prog の最後の引数，ランク番号オフセット値から，ACP のランク番号が算出され，“Portfile” のランク番号行から接続先情報を取得可能である．この macprun コマンドは次回以降の ACP ライブラリにて配布予定であり，既に開発リポジトリにはコミットされている．

現実装では macprun は mpirun にて起動されたプロセス同士を ACP により接続するために Portfile なるファイルを利用している．しかしながらこの場合では，マスタプロセスのノードと複数の mpirun にて起動されるプロセスグループのノード間にて NFS 等のファイル共有のしくみがサポートされている必要がある．そのため，今後はファイルにて情報を渡さない，デーモンプロセスを利用する実装を予定している．

3.3 MPI と ACP による複数 MPI 接続との違い

MPI-2 以降で採用された動的プロセス管理のインターフェースである MPI_Comm_spawn ならびに一連の関数を利用することにより，ユーザは複数の MPI_COMM_WORLD プログラムを接続可能である．最初に起動した 1 つのプロセス (全体マスタ) から順次 MPI プロセスグループを起動し，全体マスタと各プロセスグループの間の通信ならびにプロセスグループ間の通信を可能とする．これに対し，本方法では，複数 MPI プロセスグループならびに ACP ライブラリのみのプロセスからなる全てのプロセスを起動後に，一旦 acp_init により全てのプロセスを接続する必要がある．そのため現状の acp_init の仕様では，最初に起動したプロセスのみによる接続に限定される．

4. ACP を利用した OpenFMO プログラムの複数 MPI プロセスグループの接続

量子化学におけるフラグメント分子軌道法計算 (FMO 計算) は，分子全体をフラグメントと呼ばれる分子の部分系に分解し，各部分系 (モノマー) と部分系のペア (ダイマー) の Hartree-Fock 計算を独立に行うことで，部分系単位での分子軌道ならびに全エネルギーを計算する．この際，各モノマーとダイマーは並列計算が可能である．この FMO 計算を超高並列計算機環境にて利用可能な OpenFMO プ

```
ssh rhost_00 "cd dir ; MYRANK=0 NUMPROCS=32 LPORT=44256 RPORT=44257 RHOST=rhost_01 acp_prog" &
ssh rhost_01 "cd dir ; MYRANK=1 NUMPROCS=32 LPORT=44257 RPORT=44258 RHOST=rhost_02 acp_prog" &
...
ssh rhost_32 "cd dir ; MYRANK=2 NUMPROCS=32 LPORT=44259 RPORT=44256 RHOST=rhost_00 acp_prog"
```

図 3 ACP タスクの起動方法

```
ssh rhost_00 "cd dir ; MYRANK=0 NUMPROCS=12 LPORT=44256 RPORT=44257 RHOST=rhost_01 acp_prog" &
mpirun --mca btl openib,self -machinefile Nodefile.1 -np 5 mpi_prog --acp-multrun Portfile 1 &
mpirun --mca btl openib,self -machinefile Nodefile.2 -np 4 mpi_prog --acp-multrun Portfile 6 &
mpirun --mca btl openib,self -machinefile Nodefile.3 -np 2 mpi_prog --acp-multrun Portfile 10
```

```
 0  12   44256    44257    rhost_00   rhost_01
 1  12   44257    44258    rhost_01   rhost_02
...
11  12   44259    44256    rhost_11   rhost_00
```

図 5 複数 MPI プログラムの起動のためのコマンド (上), ならびに ACP ライブラリが利用する接続情報ファイル (下)

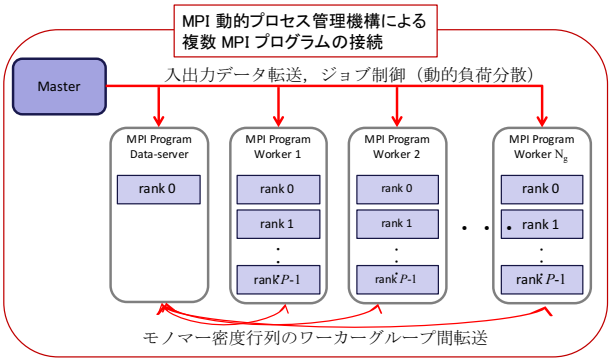

図 6 MPI 動的プロセス管理機構を利用した OpenFMO プログラムの現状実装

ログラムが九州大学の稲富らにより開発されており，2 万コアまでのスケーラブルな計算が可能であることが示されている [12], [13].

図 6 に OpenFMO プログラムの現状実装を示す．OpenFMO はモノマーやダイマーを計算する複数の MPI プロセスグループからなるワーカプログラムが主に計算を行ない，全体のマスタープログラムが複数のワーカの入出力転送や動的負荷分散を考慮したジョブ制御を行う．また，1 つのワーカプログラムは他のワーカグループの計算結果を利用するため，ワーカグループ間通信が必要である．そのため，データサーバプログラムを別にマスタが起動し，ワーカはデータサーバプログラムとのデータの送受信をする構成となっている．この複数ワーカプログラム群とデータサーバプログラムは，MPI の動的プロセス管理機構を利用し全体マスタが起動しており，各ワーカとデータサーバの接続を行なっている．

MPI の動的プロセス管理機構を利用する際には全体マスタを起動後に複数の MPI プロセスグループを動的に起動するが，OpenFMO プログラムの現実装では，プログラム実行の途中で新たなプロセスグループの生成を必要としない．そのため，本研究で提案している複数の MPI プロセスグループを ACP にて接続する方法で実装することも可能である．この際，全体マスタは ACP ライブラリのみで実装することが可能である．

5. メモリ使用量の評価

5.1 実験概要

複数 MPI プログラムの動的プロセス管理機構による接続と，本論文で実装した ACP による接続手法について，実ヒープメモリ使用量の初期評価を行なった．MPI の universe サイズに対応する全体のプロセス数を 33 とした．1 プロセスを全体のマスターに割り当てることで，残りの 32 プロセスを複数の MPI プロセス (ワーカプロセス) として利用した．このワーカープロセスのプロセスグループ数 Ng と各グループのプロセス数 Np をそれぞれ
$(Ng, Np) = (1, 32), (2, 16), (4, 8), (8, 4), (16, 2), (32, 1)$
とし，6 通りの分割について考慮した．

プログラム内容は全体マスターから 4 バイト整数を 1 つ Bcast するテストプログラムとした．MPI のみのプログラムでは，全てのワーカグループを起動後にマスタから各ワーカに順番に intercommunicator を通して Bcast を行なった．この際，全体マスタは 1 プロセスで 1 グループとした．ACP を利用したプログラムでは，acp_init 後に全体マスターから各ワーカグループのマスタに acp_copy による 4 バイトの送信を順次行ない，プロセス全体における acp_sync による同期処理の後に，各ワーカグループ内のみで Bcast を行なった．

ヒープメモリ使用量の測定には DMATP[11] を利用した．このライブラリプログラムにより，各プロセスに対しプログラム中のライブラリや関数単位にてデータを取得可能である．

本実験は PRIMERGY RX200 S7 にて行なった．計算ノード数は 16 基で，各ノードに Intel Xeon E5-2609(2.40 GHz, 4 コア) が搭載されている．メモリは 8GB, 計算ノー

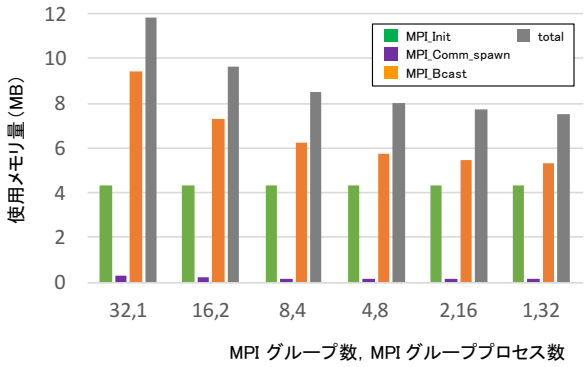

図 7　MPI-spawn 利用における全体マスタープロセスのメモリ使用量

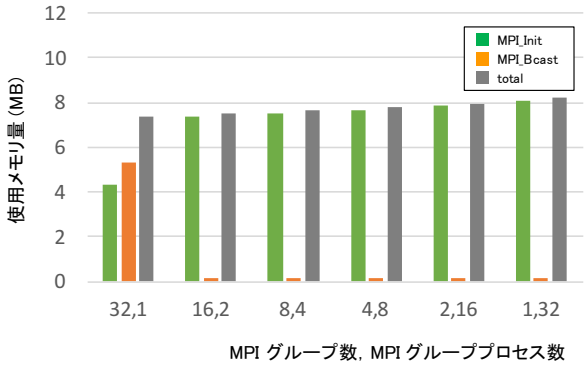

図 8　MPI-spawn 利用における MPI プロセスグループのルートプロセスのメモリ使用量

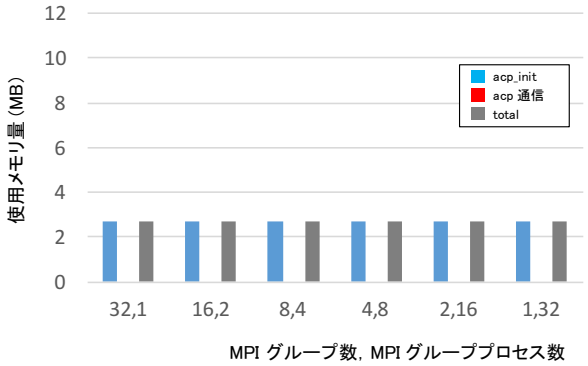

図 9　ACP ライブラリ利用における全体マスタープロセスのメモリ使用量

ド間は Infiniband QDR スイッチで接続されている．MPI ライブラリは OpenMPI 1.6.3 を使用し，ACP ライブラリは 1.2.0 リリース版を変更して使用した．計算 8 ノード上の 4 コアに 32 のワーカプロセスを，残りの全体マスター 1 プロセスを他のノード 1 つに割り当てプログラムを実行した．

5.2　実験結果

図 7，8 に MPI 動的プロセス管理機構 (MPI-spawn) を利用する全体マスタと MPI プロセスグループのルートプロセスの各関数単位における最大メモリ使用量を示す．また，図 9，10 に ACP ライブラリ利用時における全体マスタと MPI プロセスグループのルートプロセスの各関数単位における最大メモリ使用量を示す．Total はプログラム全体を通しての最大メモリ使用量を示している．

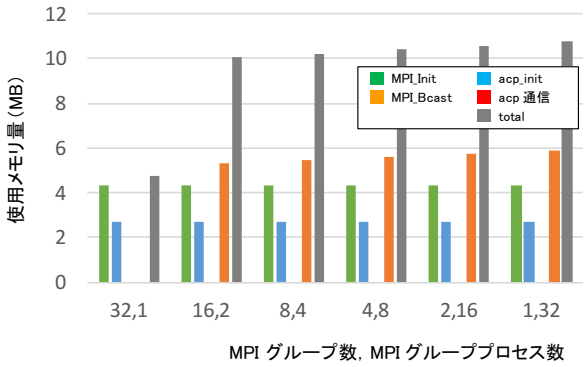

図 10　ACP ライブラリ利用における MPI プロセスグループのルートプロセスのメモリ使用量

測定全体でのメモリ使用量の最大値は MPI-spawn による全体のマスタプロセスであり，$(Ng, Np) = (32, 1)$ において 11.8MB 使用されていた．また，MPI グループ数が減少することで使用メモリ量も減少していた．このメモリ使用量の特徴はグラフから Bcast 由来であるといえる．これは全体マスタからの複数の MPI プロセスグループへの逐次的な Bcast に対応している．

これに対し，図 9，10 から，ACP を利用する場合のメモリ使用量は全体マスタプロセスにて 2.32MB，ワーカルートプロセスにて最大 10.80MB である．MPI_init のメモリ使用量は MPI-spawn の場合と同じ 4.30MB であることが読み取れる．acp_init については図 9 と同様にワーカプロセス数に依存しないメモリ消費量であり，全体マスタプロセスと同じである．これは ACP ライブラリのレイヤからは全てのプロセスは同じ階層で接続されていることに対応する．

前述の MPI-spawn のみを利用する場合の全体マスタプロセスの逐次的な複数の Bcast に比較し，ACP を利用する場合では，ワーカルートプロセスにて 1 回のみ Bcast を実行しており，グラフ全体のプロファイルは MPI グループのプロセス数が変化した際の Bcast メモリ使用量変化に由来している．そのため，例えば図 7，10 の同じ MPI グループプロセス数 32 の Bcast を比較した場合では，どちらも 1 回の Bcast を行なっているために 5.32 MB と 5.90MB と同様のメモリ使用量となっている．

また，この MPI-spawn の全体マスタプロセスと ACP を利用する MPI プロセスグループは MPI の mpirun から起動されている．そのために本実験の小規模なテストでは，それぞれ 1 と 1~32 と起動プロセス数が異るが MPI_init のメモリ使用量は両者ともに使用メモリ量のデフォルト値

の 4.30MB となっているのでは，と考えられる．これに対し，図 8 の MPI_Init のメモリ消費量は約 8MB と異なるのは，これが MPI_Comm_spawn 由来であるからと考えられ，通常の MPI_Init によるメモリ使用と別アルゴリズムを利用していると推測される．

ACP ライブラリはエクサスケール時にも省メモリにてスケーラブルとなるよう開発されているが，本実験のような小規模な場合にも MPI に比較して省メモリとなる条件が有ることが分った．そのため，計算機センター規模の並列計算において，MPI-spawn に比較しメモリ使用量において大きく優位になる場合もあると期待され，種々の条件にて測定する予定である．

6. まとめ

Advanced Communication for Exa (ACE) プロジェクトでは，将来のエクサスケール環境でもスケーラブルに利用可能な Advanced Communication Primitives (ACP) ライブラリを開発中である．

本研究ではこの ACP ライブラリを利用して複数の MPI プロセスグループを接続することを提案し，具体的な実装方法を示した．複数の MPI プログラムを接続する方法は MPI にも動的プロセス管理機構として MPI_Comm_spawn 関数と一連の関数により利用可能であるが，ACP を利用する方法は動的プロセス生成ではないものの，同様の機能を提供可能である．そのため，MPI の動的プロセス管理機構 (MPI-spawn) を利用している OpenFMO プログラムの置き換えに利用可能であることを示した．

また，複数 MPI プログラムを MPI-spawn と ACP それぞれを利用する場合により接続した両者のプログラムに対し，全体マスタとワーカプロセスの上のヒープメモリ使用量を 33 プロセスの場合について測定した．その結果，MPI の動的プロセス管理機構を利用する全体のマスタプロセスで 11.8MB と最大のヒープ領域メモリ使用量となった．これに対し ACP によって MPI プロセスグループを接続した場合の全体マスタプロセスの使用量は 2.32MB，ワーカルートプロセスは 10.8MB であった．このように，エクサスケールに向け開発されている ACP ライブラリについて，小規模な通信においても省メモリであることを示すことが出来た．

今後はさらなる大規模並列計算や，OpenFMO での通信パタンを利用した現実のアプリケーションにおける通信について，MPI のサブコミュニケータを利用するプログラムを含めて測定する予定である．

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) の研究領域「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エア技術の創出」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」の一部として実施された．

参考文献

[1] 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, 情報処理学会研究報告 2014-HPC-143-8(2014).
[2] Shinji Sumimoto, Yuichiro Ajima, Kazushige Saga, Takafumi Nose, Naoyuki Shida and Takeshi Nanri, ACP: Advanced Communication Primitives for Exa-scale Systems, 11th International Meeting High Performance Computing for Computational Science (VECPAR2014), Eugene, Jun.2014.
[3] ACE Project, http://ace-project.kyushu-u.ac.jp/index.html.
[4] 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 2014-HPC-143-10(2014).
[5] 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ーフェース, 情報処理学会研究報告 2014-HPC-143-9(2014).
[6] 安島雄一郎, 佐賀一繁, 野瀬貴史, 志田直之, 住元真司, ACP の分散動的データ構造インターフェース, 情報処理学会研究報告 2014-HPC-146-18(2014).
[7] Yuichiro Ajima, Takafumi Nose, Kazushige Saga, Naoyuki Shida and Shinji Sumimoto, ACPdl: Data-Structure and Global Memory Allocator Library over a Thin PGAS-Layer, ESPM2 2015.
[8] Takeshi Nanri, Takeshi Soga, Yuichiro Ajima, Yoshiyuki Morie, Hiroaki Honda, Taizo Kobayashi, Toshiya Takami, and Shiji Sumimoto, Channel Interface: A Primitive Model for Memory Efficient Communication, PDP 2015, 2015.
[9] 本田宏明, 山田博厚, 森江善之, 南里豪志, 高見利也, ACP ライブラリの集団通信インターフェース, 情報処理学会研究報告 2015-HPC-148(2015).
[10] Hiroaki Honda, Development of Applications on ACP Library, Language, Network and System Software 2015 (LENS2015), Oct.2015.
[11] 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 2013-HPC-138-14 (2013).
[12] 稲富雄一, 眞木淳, 本田宏明, 高見利也, 小林泰三, 青柳睦, 南一生, 京コンピュータでの効率的な動作を目指した並列 FMO プログラム OpenFMO の高性能化, *J.Comp.Chem.Japan*, Vol.12, pp.145–155 (2013).
[13] OpenFMO, http://www.openfmo.org/OpenFMO/.