Vol.2013-HPC-138 No.15
2013/2/21

DMATP-MPIを用いたMPIライブラリの関数別メモリ使用量評価

住元 真司[1]　秋元 秀行[1]　安島 雄一郎[1]　安達 知也[2]　岡本 高幸[1]　三浦 健一[1]

概要：ポストペタスケール規模での通信ライブラリでは、省メモリ性の実現が必須となる。省メモリ手法を考えるため、既存の MPI 通信ライブラリの実装の問題点を洗い出し、その問題点を解決することで省メモリ性実現を検討している。本論文では、MPI 向け動的メモリ割当分析ツール DMATP-MPI を用いて、プロセス数に対するライブラリ内の個別関数毎の動的メモリ使用量の変化について、Open MPI を対象に評価を行った。評価の結果、MPI_Init 関数がメモリ使用量のプロセス数依存性が高いことがわかった。

Memory Consumption Usage Analysis and Evaluation of MPI Library Functions Using DMATP-MPI Tool

Abstract: It is important to reduce memory consumption of communication library for post peta-scale systems. To study the method of reduction of memory consumption of communication library, we are trying to clarify issues of existing MPI libraries and fixing the issues. This paper presents the evaluation of memory consumption usage of Open MPI using DMATP-MPI dynamic memory consumptioin analysis tool. The evaluation results show that the memory consumption of MPI_Init function increases by increasing number of processes.

1. はじめに

我々は、ポストペタスケールシステムとして 100 万-1000 万ノード規模並列の省メモリを実現する通信ライブラリの研究開発を進めている。ポストペタスケール規模での通信ライブラリでは、通信性能劣化を最小限に抑えながら，既存研究 [1] に見られる通信用バッファの削減に対する対策を一層進めると共にプロセス数に比例して確保している管理用構造用メモリの削減が必要である。このための初期検討として、既存の MPI[2] 通信ライブラリの実装の問題点を洗い出し、その問題点を解決することで省メモリ性実現を検討を進めている。

既存の MPI 通信ライブラリの実装の問題点を洗い出すため、論文 [3] では、MPI プロセス全体のメモリ使用量調査を実施し報告した。しかし、論文 [3] における評価は MPI プロセス全体のメモリ使用量での評価であり、それぞれのメモリの使用用途については明らかでない。特にアプリケーション実行中の動的なメモリ確保は静的なソースコード解析は容易ではなく、これらを定量的かつ正確に把握することが省メモリ性実現を検討に不可欠である。このために、MPI 向け動的メモリ割当分析ツール DMATP-MPI を開発した [4]。

本論文では、アプリケーション実行中の動的なメモリ確保をも検知可能なメモリ割当分析ツール DMATP-MPI を用いて、プロセス数に対するライブラリ内の個別関数毎の動的メモリ使用量の変化の Open MPI[5] 上で評価を行った。評価の結果、MPI_Init 関 数がメモリ使用量のプロセス数依存性が高いことがわかった。本論文の構成は、第 2 章で、MPI ライブラリにおける省メモリ化の課題、第 3 章で動的メモリ割当分析ツール DMATP-MPI の概要について述べる。第 4 章で評価、これをうけて第 5 章で MPI の省メモリ化の進め方を言及する。そして、第 6 章で関連研究、第 7 章でまとめを述べる。

[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/(独) 科学技術振興機構　戦略的創造研究推進事業
Fujitsu Limited., Next Generation Technical Computing Unit Japan Science and Technology Agency (JST), Core Research for Evolutional Science and Technology (CREST)
[2] 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. MPI ライブラリにおける省メモリ化の課題

本章では、通信ライブラリにおけるメモリ使用方針について述べた後、既存 MPI 通信ライブラリのメモリ使用についてまとめ、省メモリ化の課題を整理する。

2.1 通信ライブラリにおけるメモリ使用方針

アプリケーション実行には、それぞれのアプリケーションプログラムが必要とするメモリを割り当てる必要がある。必要なメモリを割り当てることができない場合には、プログラム実行の中断、もしくは、メモリスワップ等の性能劣化が伴う。

メモリ割り当ては、プログラムの起動時に静的に割り当てられるものと、プログラム実行中に動的に割り当てられるものがある。それぞれのメモリ割り当て方式がプログラム実行に与える影響は次のようになる。

静的割当てメモリ: プログラム起動時にメモリ割り当てができない場合、プログラム実行自体が不可能になる。

動的割当てメモリ: プログラム実行中にメモリ割り当てができない場合、プログラムの実行途中でのプログラムが終了する場合がある他、メモリスワップやメモリ割り当てが待たされることによる性能劣化が発生する。さらには、空きメモリ量の増減と動的な割り当ての動作が複合的に重なることにより、同じプログラムにも係わらず性能差がでたり、プログラム実行が途中で終了するなど、プログラムの実行安定性と性能安定性に影響がある。

よって、プログラムの実行安定性と性能安定性を確保するためのメモリ使用方針としては，プログラムの下で稼働する通信ライブラリ等のシステムソフトウェアは、静的に割り当てるメモリ使用量を最小限度に抑え、かつ、動的メモリ割り当て量の増減を最小限に抑えるべきである。

2.2 既存 MPI 通信ライブラリのメモリ使用と既存の省メモリ化研究

本節では既存の MPI 通信ライブラリが使用するメモリの用途と既存の MPI ライブラリへの省メモリ化研究について整理する。ここでは、メモリ使用の分類として、通信用バッファ、管理制御用バッファ、デバイス依存か非依存かという視点で分類する。

通信用バッファ、管理制御用バッファによる分類

通信用バッファ： デバイス向けの送受信バッファ、間接バッファ (集団通信向け、 unexpected message 用バッファ等) がこれに相当する。

- **Eager 通信向け：** 通信相手先毎にバッファが必要
- **ランデブ通信向け：** RDMA を用いる場合は、ランデブの制御通信用のバッファが通信相手先毎に必要

管理制御用バッファ： 全ノードに対し通信するために必要な情報、デバイス向け送受信要求キュー、完了キュー等がこれに相当する。

デバイス依存、デバイス非依存による分類

デバイス依存： 例えば，Socket、 InfiniBand[6]、Tofu[7] 向けがあり、デバイス毎に異なる。

- **Socket 通信用：** Socket 用の送受信バッファ（カーネル内バッファ）が必要
- **InfiniBand 通信用：** 通信主体として Queue Pair(QP) 用メモリ、送受信要求用のキュー、完了キュー、メモリ管理テーブルであるメモリリージョン用メモリが必要
- **Tofu 通信用：** 送受信要求用のキュー、完了キュー、Tofu 用のメモリ管理テーブルである STAG、座標管理用の配列が必要

デバイス非依存： 集団通信の中間バッファ，コミュニケータ等の MPI オブジェクト情報、相手ノード毎の管理情報がこれに相当する。

本問題に対して既存 MPI 通信ライブラリにおいて次のように省メモリ化を進めている。

Open MPI、 MVAPICH[8]: InfiniBand のデバイス依存部について対応するため、次の対応を実施している。

- 通信を実施している相手先のみ QP を割り当て
- コネクション毎に必要な通信バッファを削減するため、共有受信バッファキュー、さらには，Unreliable Datagram(UD) 通信の採用
- 管理制御バッファとしてノード数に比例したバッファが必要

「京」の MPI: Tofu の RDMA 通信に対応するために、デバイス依存部のメモリ削減の他，集団通信時の間接通信バッファの削減を実施している [1]。

- 通信を実施している相手先のみ通信ハードウェアリソースとバッファを割り当て
- ランデブ方式による RDMA ベースの 1 対 1 通信の適用を拡大することにより、Unexpected Message 用バッファ利用を抑制
- Tofu は通信する相手毎にコネクションを必要としないためコネクション管理用バッファは不要
- 管理制御バッファとしてノード数に比例したバッファが必要

以上のように、既存の MPI 通信ライブラリではデバイス依存の通信用バッファの削減が主体で、制御管理バッファについては省メモリ化の取り組みは少ない。

2.3 ポストペタスケールシステムにおける既存 MPI 通信ライブラリの課題

HPCI 技術ロードマップ白書 [9] による、ポストペタスケールシステムにおいては、システムの電力対性能比を高めるため算術計算処理に比べてメモリ量が相対的に減少すると想定される。これは、ポストペタスケールシステムでは現状のシステムにも増して通信ライブラリ全般的な省メモリ化技術が必須となることを意味している。

しかし，既存の通信ライブラリが採用している通信プロトコルは、通信性能や品質を確保するために通信相手数に応じたメモリ確保が必要な構造となっている。相対的に空きメモリが減少することによる第 2.1 節で述べたプログラム実行安定性と性能安定性への影響が懸念される。さらには，一般に資源 (メモリ使用量) と性能・品質はトレードオフの関係にあるため、性能・品質の劣化をいかに最小にした省メモリ化技術を実現するかが課題である.

このため、ポストペタスケールシステムにおける既存 MPI 通信ライブラリの課題として

- Unexpected Message 用バッファ使用の抑制
- ランデブー型通信の高速化
- デバイス非依存部の管理制御バッファの削減

に取り組む必要があると考える。

3. 関数別メモリ使用評価と DMATP-MPI メモリ解析ツール概要

我々は第 2.3 節で述べた課題に取り組むべく、遠隔 Atomic 通信を用いた省メモリの取り組みを進めている [3], [10], [11], [12]。

この取り組みの初期検討として、既存の MPI 通信ライブラリの実装の問題点を洗い出し、その問題点を解決することで省メモリ性実現を検討を進める。この検討を進めるには，既存の MPI 通信ライブラリのどの関数がどのようにメモリを使用しているかの詳細の解析が不可欠であるため関数別メモリ使用評価を実施する。これを実現するために、我々は MPI 向け動的メモリ割当分析ツール DMATP-MPI を開発した [4]。

DMATP-MPI は次に述べる特徴がある。

(1) ユーザプログラムやライブラリの変更なしに簡便に実動作状況におけるメモリ使用量の測定が可能
(2) 細粒度、すなわちスレッド、ライブラリおよびライブラリ内の関数毎のメモリ使用量を分類・集計可能
(3) 動的メモリを対象に、OS が割り当てる実際のメモリ量を集計可能
(4) 集計値として、最大・最小を含むメモリ使用量、malloc 系関数の呼び出し回数を集計可能
(5) ソース修正すれば、任意指定区間のメモリ使用情報を取得可能

これらの機能的な特徴を利用し、既存の MPI 通信ライブラリのどの関数がどのようにメモリを使用しているかの詳細の解析を実施する。

4. DMATP-MPI による MPI の消費メモリ量測定と評価

本章は、DMATP-MPI を用いた MPI の消費メモリ量測定と評価を行う。主として Open MPI を扱うが、MPI 実装毎の違いも調査するため MVAPICH についても取り扱う。

4.1 評価項目と測定環境

本評価においては、MPI の全般的なメモリ消費の傾向を見るため，IMB(Intel MPI Benchmark) を用いて測定を行った。本稿では次の 4 つの評価について述べる。

- IMB における Open MPI のメモリ使用量評価
- Unexpected Message によるメモリ使用量評価
- Open MPI と MVAPICH のメモリ使用量比較
- MPI_Init のメモリ使用量解析

評価環境は、Opteron プロセッサを搭載した PC クラスタである。PC クラスタ仕様の概要を次のとおりである。

(1) x86_64 Cluster （Quad Core AMD Opteron 8354 2.2 GHz x 4CPUs、 16GB RAM）
(2) Mellanox Connect X DDR InfiniBand x 4HCA 測定は 1HCA を使用
(3) Cent OS 6.0（Kernel: 2.6.32-71.29.1.el6.x86_64)、
(4) MPI: Open MPI 1.6、 MVAPICH2 1.8-r5471
(5) メモリ使用量の測定は rank = 0 のみで実施

4.2 IMB を用いた Open MPI のメモリ使用量評価

本節では，IMB を用いた Open MPI のメモリ使用量について評価する。IMB の MPI 関数については、一通り評価を行ったが，評価結果の中で典型的な関数、MPI_Init、MPI_Recv、MPI_Alltoallv、 MPI_Alltoall の結果を示す。

4.2.1 MPI_Init

図 1 に IMB 実行における MPI_Exchange 実行時 (iter=32, msg_size=128B) の MPI_Init 関数を呼び出した際の Rank0、1、2、3 のメモリ使用量をプロセス数を増加させてプロットした結果を示す。

図 1 の結果より，Rank 番号によらず、ほぼプロセス数に正比例の消費メモリ量の傾向を示している。傾きは 1 プロセスあたり 4.5KB(4.5KB/Proc) であった。MPI_Init については、他の MPI 関数やメッセージサイズによらす同じ傾向であった。

4.2.2 MPI_Recv

図 2 に IMB 実行における MPI_Exchange 実行時 (iter=32, msg_size=128B) の MPI_Recv 関数を呼び出し

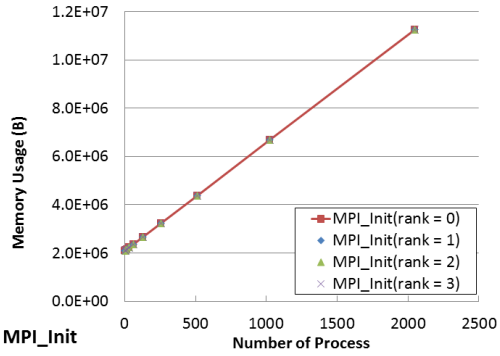

図 1 IMB における MPI_Init のメモリ消費量

た際の Rank0、1、2、3 のメモリ使用量をプロセス数を増加させてプロットした結果を示す。

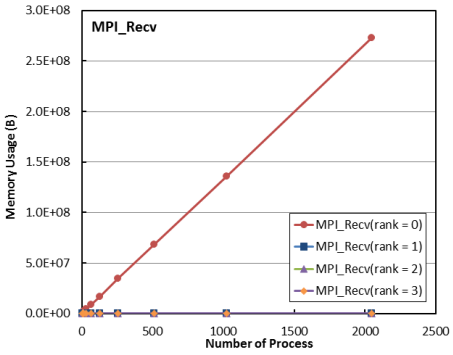

図 2 IMB における MPI_Recv のメモリ消費量

図 2 より、MPI_Recv の Rank 毎のメモリ消費量は、Rank0 とそれ以外で異なる結果となった。Rank0 のメモリ消費量はプロセス数に正比例しており、傾きは 135KB/Proc であった。これに対して Rank1、2、3 の結果はメモリ消費量は無視できる程度であった。この Rank0、1、2、3 のメモリ消費の傾向は，他の MPI 関数やメッセージサイズによらず同様であった。これは、IMB プログラムの実行開始時に各ノードから Rank0 に情報を収集するため、MPI_Send、MPI_Recv を利用しており、他ノードからの送信に対して Unexpected Message として処理されているためである。

4.2.3 MPI_Alltoallv

図 3 に IMB 実行における MPI_Alltoallv 実行時 (iter=32, msg_size=128B) の Rank0、1、2、3 のメモリ使用量をプロセス数を増加させてプロットした結果を示す。搭載メモリ量の制限で 512 プロセスまでの結果である。

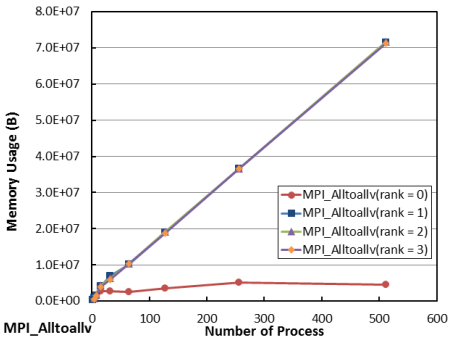

図 3 IMB における MPI_Alltoallv のメモリ消費量

図 3 の結果より，Rank0 のついては、プロセス数に対し対数比例する結果であり、Rank1、2、3 はプロセス数に正比例する結果であった。

4.2.4 MPI_Alltoall

図 4 に IMB 実行における MPI_Alltoall 実行時 (iter=32、msg_size=128B) の Rank0、1、2、3 のメモリ使用量をプロセス数を増加させてプロットした結果を示す。

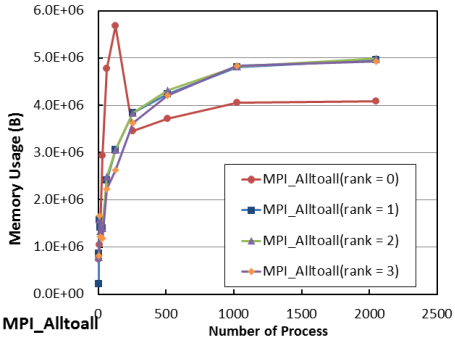

図 4 IMB における MPI_Alltoall のメモリ消費量

図 4 の結果より，Rank0 のついては、プロセス数の小さなところで、消費メモリ量が多いが、Rank0、1、2、3 共にプロセス数に対し対数比例する結果であった。Open MPI はメッセージサイズ毎に集団通信のアルゴリズムを変更しているため、Rank0 のメモリ使用量については、適用アルゴリズムを含め調査する必要がある。

4.2.5 MPI_Init vs. MPI_Alltoall

本節では，メモリ使用量が増加傾向にある、MPI_Init と MPI_Alltoall について、プロセス数の増加によるメモリ使用量について比較する。

図 5 に IMB 実行における MPI_Alltoall 実行時 (iter=32, msg_size=128B) の MPI_Init と MPI_Alltoall の結果を比較する。

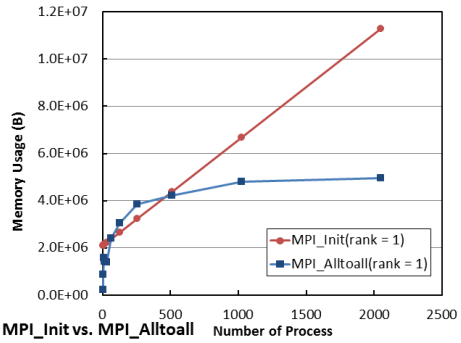

図 5 IMB における MPI_Init vs. MPI_Alltoall のメモリ消費量

図 5 の結果より，プロセス数が 500 プロセス未満の場合は、MPI_Init よりも MPI_Alltoall が仕様するメモリ量が多い。しかし、プロセス数が 500 以上になると MPI_Init のほうが MPI_Alltoall よりメモリ使用量が大きい結果となった。

この結果は, MPI の省メモリ化を考える上では, MPI_Init の内部についてメモリ使用量の削減を考えなければならないことを意味している。

4.3 Unexpected Message によるメモリ使用量評価

本節では、4.2.5 節で述べた MPI_Recv の評価で各ノードから Rank0 へのメッセージ送信により Unexpected Message となり、メモリ利用量が増加していることがわかった。本節では，この Unexpected Message によりどれくらいメモリ割り当てが行われるのかを調査する。

調査に用いたテストプログラムを図 6 に示す。図中、UM は Unexpected Message を意味する。評価では、意図的に Unexpected Message が生成される環境を作り出すために次のような処理を行う。

(1) Rank0-1 間で pingpong 転送を 100 万回実行する。
(2) Rank0-1 間の pingpong 転送の間 Rank2-59 の 58 ノードから Rank0 に 10KB のメッセージを 10,000 回転送する。
(3) Rank0 は pingpong 転送の終了後,Rank2-59 のメッセージの受信処理を行う。

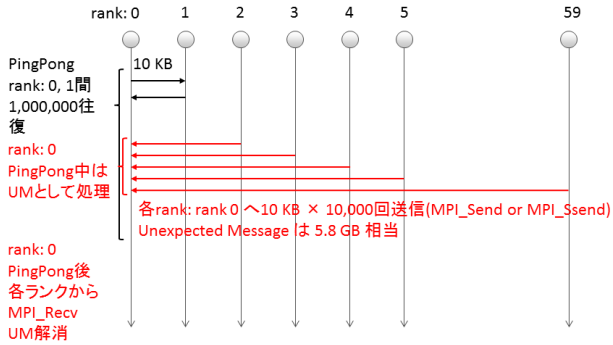

図 6 Unexpected Message のテストプログラムの動作

総計で Rank0 には 5.8GB のメッセージが届くことになる。Unexpected Message によるメモリ消費であることを示すため，同時に同期した上で転送する MPI_Ssend を用いた結果も示す。

MPI_Send送信: Unexpected Message として処理

```
---- Statistics of individual library memory usage -------
Library: /home/akimoto/OpenMPI/lib/libmpi.so.1
mem_size = 933304, mem_min = 0, mem_max = 10198882832
malloc: 630673, realloc: 832, memalign: 2074, free: 628654
  ---- Statistics of individual function memory usage ----
  Function: MPI_Init
  mem_size = 8761800, mem_min = 0, mem_max = 8777336
  malloc: 24456, realloc: 831, memalign: 38, free: 15420
  Function: MPI_Send
  mem_size = 10188099848, mem_min = 0, mem_max = 10188100488
  malloc: 603541, realloc: 0, memalign: 2036, free: 4114
  Function: MPI_Recv
  mem_size = 1952728, mem_min = 0, mem_max = 1952728
  malloc: 113, realloc: 0, memalign: 0, free: 0
  Function: MPI_Finalize
  mem_size = -10197949056, mem_min = -10197949056, mem_max = 472
  malloc: 1131, realloc: 1, memalign: 0, free: 608832
  ------------------------------------------------------------
```

MPI_Ssend送信: Unexpected Message なし

```
---- Statistics of individual library memory usage -------
Library: /home/akimoto/OpenMPI/lib/libmpi.so.1
mem_size = 928544, mem_min = 0, mem_max = 76597232
malloc: 41417, realloc: 832, memalign: 2042, free: 39371
  ---- Statistics of individual function memory usage ----
  Function: MPI_Init
  mem_size = 8756528, mem_min = 0, mem_max = 8772064
  malloc: 24448, realloc: 831, memalign: 38, free: 15417
Function: MPI_Send
  mem_size = 30822896, mem_min = 0, mem_max = 30823536
  malloc: 3852, realloc: 0, memalign: 530, free: 1070
  Function: MPI_Recv
  mem_size = 36949280, mem_min = 0, mem_max = 36949824
  malloc: 10576, realloc: 0, memalign: 1474, free: 3022
  Function: MPI_Finalize
  mem_size = -75668144, mem_min = -75668144, mem_max = 472
  malloc: 1109, realloc: 1, memalign: 0, free: 19574
  ------------------------------------------------------------
```

図 7 MPI_Send と MPI_Ssend のメモリ消費量比較

図 7 にテストプログラムを実行した際の、DMATP-MPI ツールによる出力結果を示す。図 7 の結果より、MPI_Ssend の場合は，30,822,896 バイト (0.028GB) の使用メモリ量であったのに対し、MPI_Send の場合は、10,188,099,848 バイト (9.488GB) の使用メモリ量であった。MPI_Send の場合の場合に想定の 5.8GB より多いのは 10KB のバッファに対して、メモリ割り当て時に 16KB に切り上げたメモリ (総計 9.155GB) が割り当てられているからと考えられる。

また、注目すべき点として，Unexpected Message 向けに割り当てられたメモリは MPI_Finalize が呼び出されるまで開放されないことがあげられる。詳細な調査が必要であるが，Unexpected Message により MPI ライブラリ内に大量のメモリが確保されたままになると、アプリケーションがメモリ不足で中断する可能性がある。実行安定性確保のため一定量以上のメモリ確保は制限する必要がある。

4.4 Open MPI と MVAPICH のメモリ使用量比較

本節では，Open MPI と MVAPICH のメモリ使用量の違いについて調査した結果を述べる。

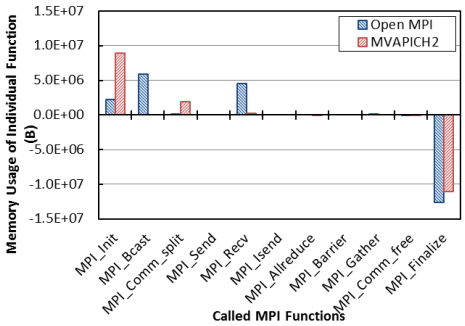

図 8 Open MPI と MVAPICH のメモリ消費比較

図 8 に、Open MPI と MVAPICH のメモリ消費比較として IMB の実行結果を示す。図 8 より、Open MPI は MPI_Recv 時にメモリを割り当てているのに対して，MVAPICH では MPI_Init 時にまとめてメモリを割り当てていることがわかった。

4.5 MPI_Init のメモリ使用量解析

現在、MPI_Init のメモリ使用量解析をすすめているところであるが、プロセス数に依存してメモリ使用量が増加している関数モジュールの一部を示す。

デバイス非依存：

- proc 系関数： ompi_proc_all, ompi_proc_world,
- opal 系関数： opal_bitmap_init, opal_hash_table_set_value_uint64, opal_hwloc132_hwloc_bitmap_alloc etc.
- orte 系関数： orte_grpcomm_base_get_proc_attr, orte_util_decode_pidmap etc.
- mca 系関数： bml 系、coll 系、pml 系

デバイス依存：

- mca 系関数： btl 系

5. MPI の省メモリ化の進め方

ここでは、これまでの評価に基づき、MPI の省メモリ化を進めるにあたりすべきことについて考察する。

MPI_Init： プロセス数の増加により、通信バッファだけでなく、制御用の構造体においても使用メモリ量が増加することが観測された。また、通信デバイスに依存するもの、しないものがあることがわかった。これら、4 つの面から省メモリ化に取り組む必要がある。

Unexpected Message： 評価の結果、Unexpected Message 向けのメモリ使用は無制限に行われ、かつ、プログラムの終了時まで開放されない結果となった。割り当てメモリ量の制限，不要時に開放すると言った対策の他、Unexpected Message を使用しない方式の採用を含めて考えるべきである。

集団通信： 集団通信に用いるアルゴリズムの違いによるメモリ使用量の違いがあるので、メモリ使用量についても考慮した集団通信アルゴリズムを検討すべきである。

6. 関連研究

Open MPI や MVAPICH における InfiniBand 実装では通信用のハードウェアリソース量の制限に加え、宛先毎の送受信バッファと送受信要求と完了構造体が必要であるため。その削減のため、ハードウェアリソース割り当てを動的に行う。信頼性のないプロトコル (UD) の採用による削減などの研究が行われている。ただし、我々が進めている，制御用メモリと Unexpected Message については考慮していない点が異なる。

「京」の MPI 通信においては、本プロジェクトと同様にプロセス数の増加により増える消費メモリ量の削減のための工夫が実施しているが、ランデブープロトコル自体の高速化は未対処である点と管理用のメモリは o(N) で必要であり削減していない。制御用メモリと Unexpected Message については考慮していない点が異なる。

7. まとめ

ポストペタスケール規模での通信ライブラリでは、省メモリ性の実現が必須となる。省メモリ手法を考えるため、既存の MPI 通信ライブラリの実装の問題点を洗い出し、

その問題点を解決することで省メモリ性実現を検討している。本論文では、MPI 向け動的メモリ割当分析ツール DMATP-MPI を用いて、プロセス数に対するライブラリ内の個別関数毎の動的メモリ使用量の変化について、Open MPI を対象に評価を行った。この結果、既存の MPI においては、MPI_Init、Unexpected Message、 集団通信の観点から省メモリ化を進める必要があることがわかった。

今後、調査をすすめ、既存 MPI の省メモリ化を進める。

参考文献

[1] 住元真司, 川島崇裕, 志田直之, 岡本高幸, 三浦健一, 宇野篤也, 黒川原佳, 庄司 文由, 横川三津夫. 「京」のための MPI 通信機構の設計. SACSIS 2012 - 先進的計算基盤システムシンポジウム. 情報処理学会, May 2012.

[2] The Message Passing Interface (MPI) standard: http://www.mpi-forum.org.

[3] 三浦健一, 秋元秀行, 安島雄一郎, 岡本高幸, 住元真司. エクサスケールコンピューティングに向けた省メモリ通信ライブラリの検討. 情報処理学会研究報告 12-HPC-133(14). 情報処理学会, Mar. 2012.

[4] 秋元秀行, 三浦健一, 安達知也, 岡本高幸, 安島雄一郎, 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 13-HPC-138(14). 情報処理学会, Feb. 2013.

[5] Open MPI: http://www.open-mpi.org/.

[6] InfiniBand Trade Association: http://www.infinibandta.org/.

[7] Yuichiro Ajima, Shinji Sumimoto, and Toshiyuki Shimizu. Tofu: A 6d mesh/torus interconnect for exascale computers. In *IEEE Computer*, pp. 36–40, Nov. 2009.

[8] MVAPICH: http://mvapich.cse.ohio-state.edu/.

[9] HPCI 技術ロードマップ白書: http://open-supercomputer.org/wp-content/uploads/2012/03/FutureHPCI-Report.pdf.

[10] 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. 情報処理学会研究報告 12-HPC-133(7). 情報処理学会, Mar. 2012.

[11] 秋元秀行, 三浦健一, 岡本高幸, 安島雄一郎, 住元真司. InfiniBand Atomic Operation の性能評価. 情報処理学会研究報告 12-HPC-133(8). 情報処理学会, Mar. 2012.

[12] 住元真司, 安島雄一郎, 安達知也, 岡本高幸, 秋元秀行 and 三浦健一. 遠隔 Atomic 通信を用いた省メモリ性実現のための方式検討. 情報処理学会研究報告 13-HPC-138(13). 情報処理学会, Feb. 2013.