

Open MPIにおけるメモリ削減機能の評価

住元 真司¹ 川島 崇裕¹ 志田 直之¹

概要: エクサスケールシステム向けの MPI においてはプロセス数が 100 万プロセス以上になると想定され、プログラム起動時間とメモリ利用量に大きく影響する。特にメモリ利用量はアプリケーションが利用可能なデータ量に大きく関係するために、抜本的な削減が必要な状況にある。本稿では Open MPI で採用しているメモリ削減機能の概要とその評価について述べる。

Evaluation of Memory Saving Techniques on Open MPI

SUMIMOTO SHINJI¹ KAWASHIMA TAKAHIRO¹ SHIDA NAOYUKI¹

1. はじめに

エクサスケール規模のシステム開発が 2020-2021 年を目標に進められている [1], [2], [3], [4], [5]。エクサスケール規模のシステムにおいては、現在稼働中のペタスケールシステムに比べて 100 倍規模のプロセッサコア数が必要とされているため、フルシステム構成での実行時には最大 100 倍規模のプロセス数のジョブ実行を想定する必要がある。

現在、我々は Open MPI ベースの富士通 MPI を次世代機向けに開発を進めている。ベースの Open MPI は Open MPI 開発コミュニティと共同で開発を進めている。エクサスケールのシステムにおいては、システム全体の電力制約が厳しいため総メモリ量の性能比での増加が見込めなくなる。しかし、従来の通信ライブラリは、並列プロセス数が増加した場合に、送受信バッファや通信制御用メモリがプロセス数に比例して増加するという課題がある [6], [7]。

我々はこの課題に対して Open MPI コミュニティと共に取り組み、各モジュールの開発メンバの協力の元、3 つの省メモリ機構が実現された。本論文では、この省メモリ機能化機構とその効果について述べる。

第 2 章において、MPI における省メモリ化の課題と解決の方向性、第 3 章、第 4 章でベースになる Open MPI と Open MPI 上に実装された省メモリ機構について述べ、第 5 章でこれらの省メモリ機構について評価する。

2. MPI における省メモリ化の課題と解決の方向性

2.1 MPI におけるメモリ使用

MPI 通信ライブラリにおけるメモリ利用には送受信バッファ用メモリと通信先情報などを格納する制御情報用メモリに分類できる。

送受信バッファに利用されるメモリ利用量は、通信プロトコルで大きく異なる。MPI 通信ライブラリでは通常 Eager プロトコルと Rendezvous プロトコルが用いられる。Eager プロトコルは送信先と同期を取らずに送ることができる分高性能である。しかし、送信分の受信側のバッファを確保する必要があるため、多くのメモリを必要とする。一方、Rendezvous プロトコルは送信先の受信バッファ確保の同期後に送信を始めるため、より少ないメモリで実現可能である。ユーザのメモリ間を直接ハードウェアにより転送する RDMA を用いることにより、送受信用のバッファを持たずに実現可能である。通常 MPI ライブラリ実装では短いメッセージを Eager プロトコルで一定以上のメッセージ長以上を Rendezvous プロトコルで実装している。

制御情報用のメモリは、通信用のバッファ以外の情報を格納する。通信先プロセス情報、通信ハードウェア制御用の情報のほか、MPI 仕様が提供するオブジェクト情報などが相当する。特に通信先プロセスの情報と通信ハードウェア制御用の情報は一般に通信先数に比例して必要になるた

¹ 富士通
川崎市中原区上小田中 4-1-1

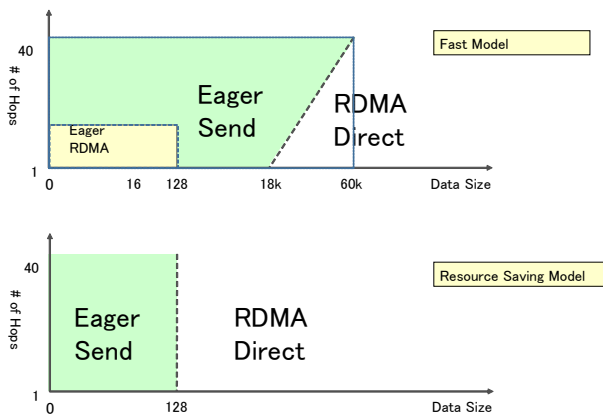


図 1 京の MPI の省メモリプロトコル

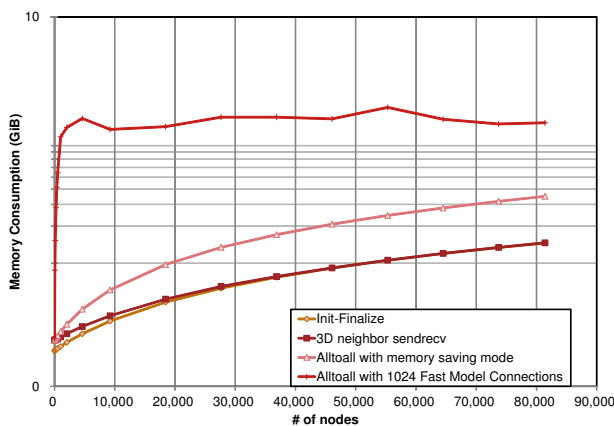


図 2 京におけるメモリ使用量の評価

めにメモリ使用量が大きくなる。

2.2 京向け MPI における省メモリ対策

10 万プロセス規模の京 [8] 向けの MPI 開発 [9] では、MPI 通信ライブラリの使用メモリを搭載メモリ量の 10% 未満に抑制する必要があった。この実現においては、特に前者の送受信バッファ使用量を削減することが課題となった。このため、通常のエーger プロトコルと Redezevous プロトコルの組み合わせた高速通信用プロトコル (Fast Model) の他に 128 バイトの 1 メッセージを受信する Eager プロトコルとそれ以上は Redezevous プロトコルを用いたメモリ資源節約型プロトコル (Resource Saving Model) を採用した (図 1)。

本方式を用いた場合の京におけるメモリ使用量の評価結果を図 2 に示す。図 2 中、Alltoall with memory saving mode が Resource Saving Model による測定結果であり、Alltoall with 1024 Fast Model Connections が Fast Model の通信数を 1024 に絞った場合の測定結果である。Alltoall のアルゴリズムは Simple Spread である。本結果より、Fast Model の通信先数を制限しないと 1024 ノード以上も比例したメモリ使用量が必要であるが、通信先を一定数に制限

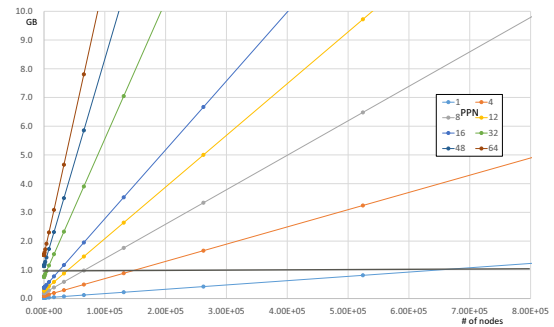


図 3 京におけるメモリ使用量推定

することにより一定量以下のメモリ利用量を実現している。

2.3 エクサスケール向け MPI における省メモリ化の課題

しかし、エクサスケールシステム向けの MPI においては、送受信バッファ用メモリだけでなく、制御情報用のメモリ使用量の削減も重要になってくる。我々は、JST/CREST の「ポストベタスケール高性能計算に資するシステムソフトウェア技術の創出」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」において、既存の MPI 通信ライブラリの使用メモリ量の調査並びに解析を行った。

この調査より、既存の MPI ライブラリである Open MPI[10]1.4.5 のメモリ使用量が、InfiniBand 上で最もメモリ使用量の少ない UD(Un-reliable Datagram) 通信を使用した場合においても、100 万プロセスの場合に 2.2GB ものメモリがプロセス毎に必要なことを示した (論文 [11])。これは、1 ノードあたり 1 プロセス実行であれば、全体のメモリ量に比べて大きなインパクトはないが、エクサスケール規模のシステムで想定される Many Core プロセッサベースのシステムで 1 ノードあたりのプロセス数が増加すると、これに比例して必要になる。例えば、8 プロセスの場合は 17.6GB のメモリ使用量を必要とするのが問題になる。

ここで具体的なシステムでのメモリ使用量を把握するために、京向けに開発した MPI においても、図 2 の Init-Finalize の結果を用いて MPI のメモリ使用量の推定を行った。MPI 初期化終了のグラフから外挿した結果をもとに、ノード内プロセス数 (Processes Per Node:PPN) を変化させた場合のメモリ使用量を推定した結果を図 3 に示す。

図 3 の計算結果よりノード内プロセス数が増えるにつれてプロセス数に比例したメモリ量が必要であることがわかる。ここで 1GB のメモリ内でのノード数について計算してみると、ノード内 1 プロセスの場合は 650K ノードまで対応可能であるが、ノード内 8 プロセス (8PPN) の場合は 68K ノードに制限される。ノード内プロセスの増加に比例してノード内メモリ量が増加するため今後のメニーコアシステムに対応するためにはノード内プロセス数に比例しな

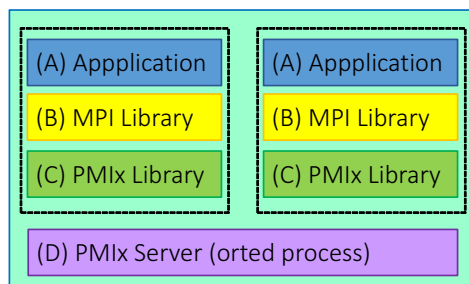


図 4 Open MPI の構成

い方式を考慮しなければならない。

このように、既存の MPI 通信ライブラリにおいてはプロセス数に比例したメモリ量を必要としているため、これを回避する方式の実現がエクサスケール向け MPI 実装の大きな課題である。

2.4 エクサスケール向け MPI における省メモリ化の指針

プロセス数に比例しない通信ライブラリ実装を進めるため、論文 [6], [7] において、MPI のメモリ使用の詳細を調査、整理した。整理の結果、MPI が割り当てるメモリは、MPI Init 時にすべてのプロセス分のメモリが静的に割り当てられることが判明した。更には実際に割り当てられるメモリは他のプロセスの情報と通信に利用する通信リソース情報が大きな割合を占め、かつ、これらのメモリがプロセス数に比例して必要なことがわかった。

この手法は性能を優先して安定して実行するには適している手法である。しかし、数千プロセスを超えるアプリケーションにおいて、すべてのプロセス間で通信する割合は多くない。このため、通信ライブラリ初期化時にすべてのプロセス情報と通信リソースをすべて割り当てると通信しないプロセス情報は無駄になる。よって、使用メモリ量を減らすためには、実際に通信を行うプロセス情報だけをメモリ上に確保し、通信する場合にのみ通信リソースを割り当てておくべきである。

更には、実際に通信を行うプロセスであっても、同一ノード内に格納される情報は、相手先プロセス情報他、複数プロセス間で共有可能な情報が多くを占め、プロセス毎に必ずしも独立して持つ必要のない情報が含まれる。このため、同一ノード内でのデータ共有を考えるべきである。

3. Open MPI について

Open MPI[10] は、George Bosilca, Jeff Squyres を主要リーダーとするオープンソース開発コミュニティである。MPICH と並ぶ MPI 開発コミュニティである。参加団体は 41 団体であり、ICL/UT, Cisco, Intel, AWS, Fujitsu, IBM, LANL, ORNL, SNL ほか多くの研究団体とベンダーが参加している。富士通も京コンピュータ開発において Open

MPI をベースにした富士通 MPI を開発したほか、最近では、独自発展を進めてきた IBM も Spectrum MPI と称し、Open MPI ベースに移行している。Spectrum MPI は米国のプレ・エクサスケールシステムである CORAL プロジェクトの Summit[3] と Sierra[4] で採用されている。

Open MPI 開発コミュニティにおいて、富士通は MPI ver 4.x 仕様のうち一部の機能を担当するほか、Arm/SPARC 対応、Java Binding, FP16, 省メモリ化、自動テストシステム MTT への協力や Bug Fix 等で貢献を続けている。

図 4 に Open MPI がプログラム実行を行う際の計算ノード内の実行モジュール構成を示す。2015 年後期以降の Open MPI は、プロセスマネージャとして PMIx が導入されている。図 4 において MPI プログラムの実行の手順について説明する。MPI プログラム実行時には PMIx Server により各計算ノードに並列プロセスが生成され、並列プロセス上で PMIx のクライアントライブラリにより MPI プログラム起動に必要な情報が整備され、MPI ライブラリがこの情報を元に MPI プログラムを起動する。

4. Open MPI における省メモリ機構

Open MPI コミュニティにおける省メモリ機構の導入は論文 [6], [7] の内容をベースにコミュニティ内での議論により、1) 相手先プロセス情報や通信資源の割り当ては必要とする相手に限定して割り当てるべき、2) 冗長データは排除すべき、という 2 つの方針により、3 つの機構が実装された。

Dynamic Add Procs: MPI Init 時に静的に全プロセス向けに割り当てていた相手先プロセス毎の通信リソースを初回通信時の動的割り当てとする機能

Shared Memory Data Storage(Dstore): MPI ライブラリ内で利用する情報データベースである Key Value Store(以降 KVS) の情報を同一ノード内では個々のプロセス単位で持たず、ノード内の共有メモリを用いて冗長なデータを排除する機構、

Direct modex: KVS データベース構築時、通常、全ノードでのデータ共有の完了時に同期を取り、全ノード間でデータを同期して各プロセスに KVS 情報をコピーする。Direct modex は、データ共有のための同期は取るが、実際のデータのコピーはあるプロセスが KVS をアクセスしたときに初めてデータコピーを実行する。利用する情報のみをプロセス内に格納する機能である。

5. 省メモリ化機構の評価

本章では前章で述べた Open MPI で導入された省メモリ化機構の評価について述べる。Dynamic Add Procs の評価と Dstore, Direct modex の評価は評価時期が異なるため、評価となる Open MPI と PMIx のバージョンと評価

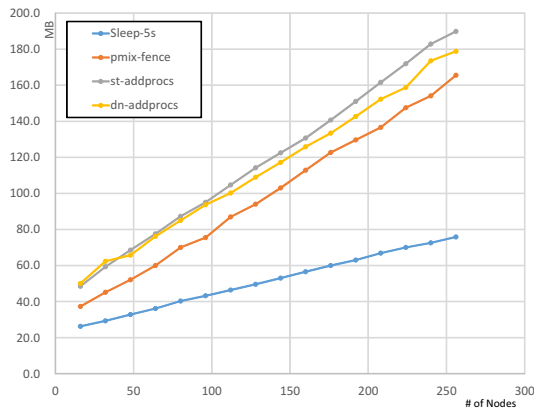


図 5 Dynamic Add Procs 機構のメモリ使用量 16PPN(Process Per Node)

システムが異なっている。それぞれの評価環境を表 1, 2 に示す。

表 1 評価環境: Dynamic Add Procs 機構

System	PRIMEHPC FX10 16 Cores/Node (256 node)
Open MPI	version 2.0.0
PMIx	version 1.1.2

表 2 評価環境: Dstore and Direct mode

System	PRIMEHPC FX100 32 Cores/Node (64 node)
Open MPI	version 3.0.1
PMIx	version 2.0.1

これ以降、各省メモリ化機構毎の評価について述べる。

5.1 Dynamic Add Procs 機構の省メモリ性評価

表 1 の評価環境を用いて Dynamic Add Procs 機構の省メモリ性評価を行った。具体的な Dynamic Add Procs 機構の評価のために、Open MPI を構成する階層毎にメモリ使用量を測定し、差分を取ることで効果を評価する。

図 4 の各モジュールの使用メモリ量を測定するために以下の 3 つのレベルのメモリ使用量を測定した。

1. 実行プログラム起動: MPI プログラムでない簡易なプログラム (sleep 5s) を実行し、PMIx Server のメモリ使用量を測定する。(グラフ凡例: Sleep-5s)
2. PMIx 初期化プログラム実行: PMIx の初期化/終了関数を呼ぶプログラムを実行することにより PMIx Client と Server の使用メモリ量を測定し、1. の測定値との差分により PMIx Client の使用メモリ量を測定する。(グラフ凡例: pmix-fence)
3. MPI 初期化プログラム実行: MPI の初期化/終了関数を呼ぶプログラムを実行することにより MPI ライブ

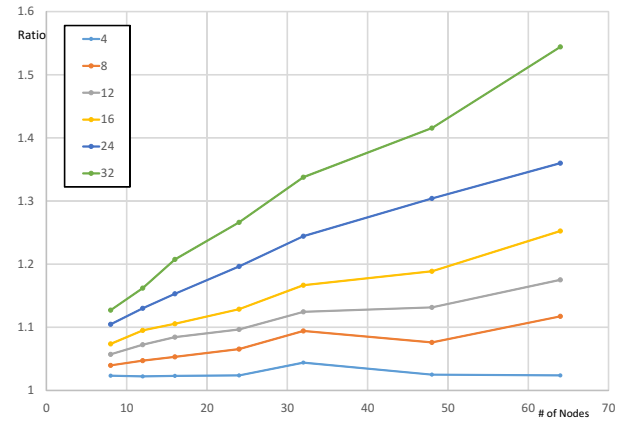


図 6 Dstore 機構の有無によるメモリ使用量の比

ラリ初期化時の使用メモリ量を測定し、2. の測定値との差分により Dynamic Add Procs 機構が有効でない MPI ライブラリの使用メモリ量を測定する。(グラフ凡例: st-addprocs)

4. MPI 初期化プログラム+Dynamic Add Procs 機構:

MPI の初期化/終了関数を呼ぶプログラムを Dynamic Add Procs 機構を有効化するオプションを追加して、実行することにより MPI ライブラリ初期化時の使用メモリ量を測定し、3. の測定値との差分により Dynamic Add Procs 機構が有効な場合のメモリ使用量削減の効果を測定する。(グラフ凡例: dn-addprocs)

図 5 に Dynamic Add Procs 機構のメモリ使用量 16 Process Per Node(16PPN) の測定結果を示す。図 5 の測定結果は積み上げグラフであり、差分がそれぞれの機構によるメモリ使用量に相当する。図 5 において、プロセスあたりのメモリ使用量の大きさはグラフの傾きで示されており、最も大きなものは PMIx Client のメモリ使用量で 256 ノードの時に全体の 47% を占め、それに続くのが PMIx Server のメモリ使用量の 42% になる。Dynamic Add Procs 機構自身のメモリ使用量は上記 3. と 4. の差分となり、その効果は 5.8% であった。メモリ使用量の評価において全体的には、PMIx Server と PMIx Client のメモリ使用量が支配的であった。PMIx ライブラリのメモリ使用量を削減する必要がある。

5.2 Dstore 機構の省メモリ性の評価

表 2 の評価環境を用いて Dstore 機構の省メモリ性評価を行った。図 6 に評価結果を示す。実行は Dstore の有る/無し状態でメモリ使用量を Process Per Node 数を変化させて測定し、Dstore 機構無しの結果を Dstore 有りの結果で割った比をプロットしている。測定結果で 1.0 より値が大きければ Dstore 機構を用いた場合の方がメモリ使用量が少ないことを意味している。

図 6 の評価結果より、すべてのケースにおいて Dstore 機

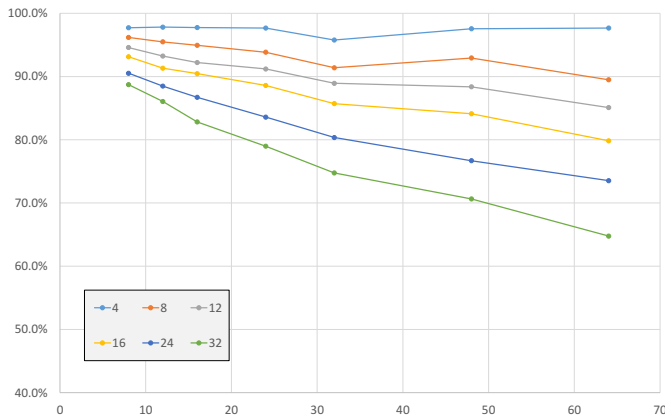


図 7 Dstore 機構未適用を 100%とした場合の適用時のメモリ使用量削減効果

構を用いる方がメモリ使用量が減るという結果になった。また、設計通りノード辺りのプロセス数が多いほど、またノード数が増えるほど Dstore 機構の効果が大きくなることが示された。

更に、実際のメモリ使用量削減効果を評価するために、図 6 の評価結果を、Dstore 機構未適用を 100%とした場合の適用時のメモリ使用量削減効果を図 7 に示す。図 7 より、64 ノード 32 プロセス時に 35.2%のメモリ削減効果があることがわかった。

5.3 Direct Modex 機構の省メモリ性の評価

Dstore と同様に表 2 の評価環境を用いて Direct Modex 機構の省メモリ性評価を行った。Direct Modex 機構の評価を行うために以下のオプションを用いて測定を行った。

- (1) オプション無し: Direct Modex 機構なし
- (2) pmix_base_async_modex=1: Direct Modex なし
- (3) pmix_base_async_modex=1, pmix_base_collect_data=0: Direct Modex あり

MPI.Init/MPI.Finalize に加えて、通信パターンによる変化を見るために MPI 上で Ring 通信を行うプログラムを用いて測定した。

図 8, 図 9 にそれぞれの測定結果を示す。図 8, 図 9 の凡例において、例えば、1-32-dstore の意味は (Direct Modex 機構なし)-32PPN-Dstore 機構ありを、2-32 の意味は (pmix_base_async_modex=1, Direct Modex なし)-32PPN-Dstore 機構無しを、3-32-dstore の意味は (pmix_base_async_modex=1, pmix_base_collect_data=0, Direct Modex あり)-32PPN-Dstore 機構ありを意味している。

図 8 の評価結果より、MPI.Init/Finalize 実行時の Dstore の効果は 32PPN, 64 ノード実行時 35.2%に対し、Direct Modex 機構の効果は、1%であった。図 8 の評価結果についても MPI.Init/Finalize+Ring 通信実行時は、32PPN, 64 ノード実行時、Dstore 機構の効果は変わらず、Direct

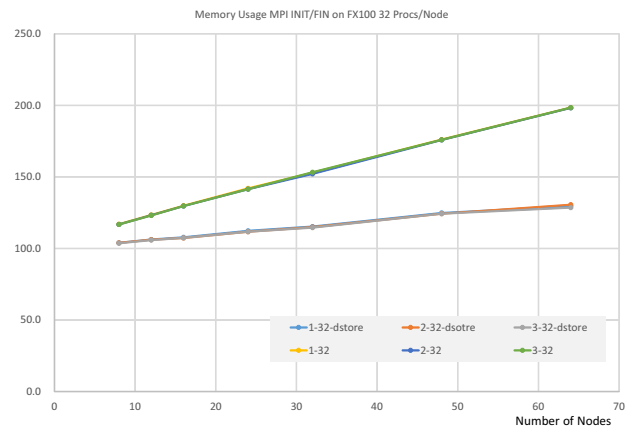


図 8 Direct Modex 機構の効果 MPI.Init/Finalize 32PPN (mca オプション)-(PPN)

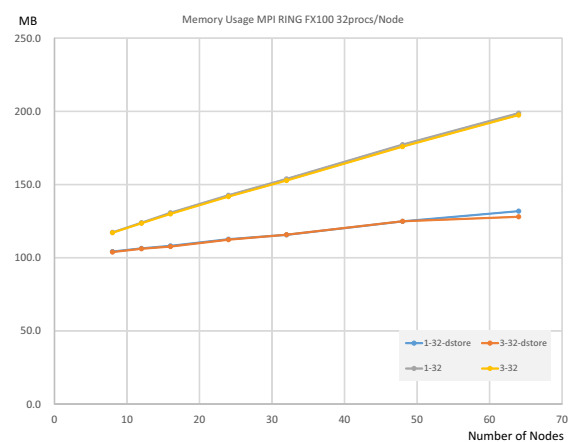


図 9 Direct Modex 機構の効果 MPI.Init/Finalize+Ring 通信 32PPN (mca オプション)-(PPN)

Modex 機構の効果は 3%と改善した。本来、Direct Modex 機構については、ノード数が多くなるほど効果が高くなると想定されるので、より大規模環境で評価する必要があると考えている。

5.4 各省メモリ化機構の 100 万プロセス時の使用量予測

本節では、これまでに評価した 3 つの省メモリ化機構の評価結果を元に 100 万プロセスでの各機構の効果を予測する。

Dynamic Add Procs 機構のメモリ使用量予測

図 5 で得られた結果から一次近似式を求めて外挿した結果を図 10 に示す。

図 10 において、100 万プロセスは 16PPN の場合 64K ノードの場合に相当する。この場合の静的割り当ての場合の使用メモリ量は 38.5GB であるのに対し、Dynamic Add Procs 機構を導入した場合 34.8GB と 3.7GB の削減となった。この値は、本測定を行った PRIMEHPC FX10 の主記憶の 32GB よりも大きな値となっているため抜本的に削減する必要がある。

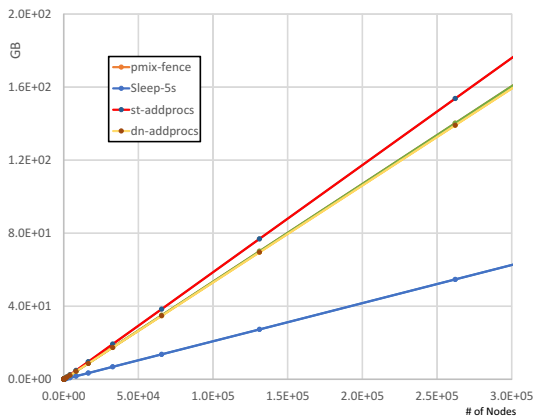


図 10 Dynamic Add Procs 機構のメモリ使用量予測 16PPN

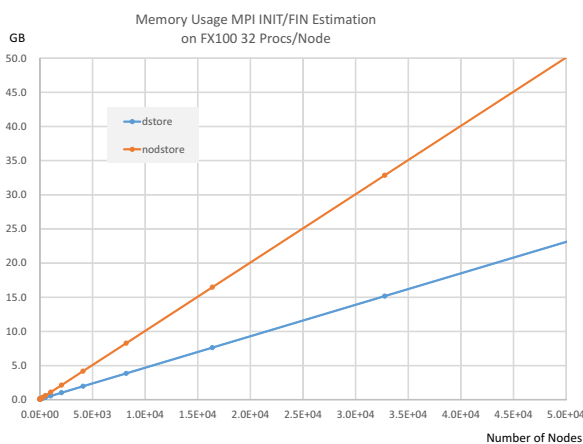


図 11 Dstore 機構の効果予測:MPI.Init/Finalize 32PPN

Dstore 機構のメモリ使用量予測

同様に図 8 で得られた結果から一次近似式を求めて外挿した結果を図 11 に示す。図 11 中、nodstore は Dstore 機構なし、dstore は Dstore 機構ありの結果を示す。図 11 の結果より、32PPN の場合に 100 万プロセスは 32K ノードの場合となるので、Dstore 機構導入時のメモリ使用量は 1 ノードあたり 15GB となった。これは、FX100 のメモリ使用量である 1 ノードあたり 32GB とするとその 47% に相当する。これは更なる削減が必要である。

Direct Modex 機構のメモリ使用量予測

図 8, 図 9 の評価結果からは、ノード数増加による削減の傾向が見えつつあるが、ノード数増加によるメモリ使用量の削減が継続的に観測できるかどうかを判断するのは困難である。従って、よりノード数を増加した環境での評価が必要であると考えている。

5.5 これからの課題

これまでに得られた評価結果より、Open MPI に導入された 3 つの機構により、一定のメモリ使用量削減の効果が得られることがわかった。一方、エクサスケールで想定される 100 万プロセス実行時のメモリ使用量見積りにおいて

は、更なる削減が必要なることも明らかになった。

本節では、これまでに得られた評価結果から、次のステップとして、どの部分のメモリ使用量の削減を考えるべきかについて明らかにしたい。

表 3 各省メモリ化機構のプロセス単位のパラメータ

1PPN	PMIx Server	MPI 静的	動的 Add procs	Dstore なし	Dstore あり
A. 傾き 増分	12.73 KB 0%	35.79 181%	32.39 154%	30.52 140%	14.05 10%
B. 切片	1,221 KB	2,441	2,441	3,052	3,052
100 万 プロセス	13.69 GB	38.47	34.82	32.77	15.09

表 3 に、前節で用いた近似式の傾き、増加分と切片、並びに 100 万プロセス時のノード辺りのメモリ使用量を示す。表 3 より、3 つの機構の導入により、MPI を実行しない PMIX Server だけの利用時のメモリ使用量に比べて 10% の増加分までに削減されていることがわかる。この 2 つのメモリ使用量は評価時期が異なるため、PMIx のバージョンも異なり PMIx Server/Client の実装が変わっている可能性があり、実測値として PMIx の切片の値が異なっている。しかし、大規模実行時のメモリ使用量の大部分を占める 1 プロセスあたり増加を示す傾きの値については有効な結果であり、今後のメモリ資料量削減の課題を探るには十分である。

さて、この傾き部分 10% までメモリ使用量の削減されたわけであるが、これ以上の削減を考えるには、PMIx Server 部分の削減並びに、今回 Btl として用いた TCP/IP 利用によるメモリ削減を考える必要がある。Btl 層は TCP/IP ではなく Tofu 向けの Btl となるため、PMIx Server と Tofu Btl のメモリ使用量を調べる必要があると考えている。

6. まとめ

本稿では Open MPI で採用しているメモリ削減機能の概要とその評価について述べた。Open MPI コミュニティにおいて、1. 相手先プロセス情報や通信資源の割り当ては必要とする相手に限定して割り当てるべき、2. 冗長データは排除すべき、という 2 つの方針により、Dynamic Add Procs 機構、Dstore 機構、Direct modex 機構という 3 つの省メモリ化機構が Open MPI 上に実現された。

実装された 3 つの省メモリ化機構を評価した結果、Dynamic Add Procs 機構においては、FX10 上で 16PPN, 256 ノード実行時に使用メモリ上全体のうち 5.8% のメモリ使用量削減、Dstore 機構機構については、FX100 上で 32PPN, 64 ノード実行時に使用メモリ上全体のうち 35.2% のメモリ使用量削減、Direct modex 機構については、同 FX100 上において Ring 通信時に 3% メモリ使用量削減の効果を

得た。

更に、測定結果より 100 万プロセスでのメモリ使用量予測を行った。32PPN, 32K ノード時において 15GB/ノードと導入された省メモリ機構により従来の 39% のメモリ使用量となったが、更なる削減が必要であることがわかった。

なお一層のメモリ削減を実現するためにはプロセスディスパッチを行う PMIx Server と Open MPI のデバイス依存の通信層である Btl 層を調査する必要がある。今度は、PMIx Server と Open MPI のデバイス依存の通信層である Btl 層を調査して更なるメモリ使用量削減を実施する。

謝辞 本論文の一部は、文部科学省「特定先端大型研究施設運営費等補助金（次世代超高速電子計算機システムの開発・整備等）」ならびに、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ェア技術の創出」の研究課題「省メモリ技術と動的最適化技術によるスケラブル通信ライブラリの開発」で実施された成果に基づくものである。

参考文献

- [1] Riken R-CCS Post K computer:
<http://www.r-ccs.riken.jp/jp/post-k>.
- [2] Coral Aurora: <https://www.alcf.anl.gov/alcf-aurora-2021-early-science-program-data-and-learning-call-proposals>.
- [3] Coral Summit: <https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/>.
- [4] Coral Sierra: <https://computation.llnl.gov/computers/sierra>.
- [5] China Tianhe-3: <https://www.straitstimes.com/asia/east-asia/china-builds-tianhe-3-the-worlds-first-exascale-supercomputer-says-scientist>.
- [6] 住元真司, 秋元秀行, 安島雄一郎, 安達知也, 岡本高幸, 三浦健一. DMATP-MPI を用いた MPI ライブラリの関数別メモリ使用量評価. 情報処理学会研究報告 13-HPC-138(14). 情報処理学会, Feb. 2013.
- [7] Shinji Sumimoto, Yuichiro Ajima, Kazushige Saga, Takafumi Nose, Naoyuki Shida, and Takeshi Nanri. The design of advanced communication to reduce memory usage for exa-scale systems. In *High Performance Computing for Computational Science*, Vol. 10150 LNCS of *Lecture Notes in Computer Science (including sub-series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 149–161, Germany, 1 2017. Springer Verlag.
- [8] Riken R-CCS K computer:
<http://www.aics.riken.jp/en/kcomputer/>.
- [9] 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. SACGIS 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.
- [10] Open MPI: <http://www.open-mpi.org/>.
- [11] 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.