Vol.2021-HPC-182 No.15
2021/12/7

スーパーコンピュータ富岳上でのMPI集団通信性能の評価

細野 七月[1,a)]　岩澤 全規[2,b)]　牧野 淳一郎[1,c)]

概要：スーパーコンピュータ富岳は，158976 ノードから構成される超大規模計算機である．故に，高い実行効率を得るためには MPI 集団通信の性能や富岳上での性質を知る事が重要である．そこで，集団通信の中から `Alltoall`，`Alltoallv` 及び `Bcast` の 3 つについて，富岳上でベンチマークを行った．測定にあたって，通信アルゴリズム，セグメントサイズやネットワーク・トポロジーの形状などを考慮した．本研究ではその結果と，これらの集団通信において高い性能を得るためのパラメータやネットワーク・トポロジーの設定の仕方について，報告する．

キーワード：MPI 集団通信性能, 富岳

Performance evaluation of MPI collective communications on the supercomputer FUGAKU

Abstract: The supercomputer "FUGAKU" is a massively parallel computer, which consists of 158976 nodes. The performance of MPI collective communication is important to achieve highly effective performance. In this article, we will report the performance evaluation of `Alltoall`, `Alltoallv` and `Bcast` on FUGAKU. We surveyed the effect of algorithms, segment size, and node topology. We will report the results and the optimal setting to achieve high efficiency.

Keywords: Performance of MPI collective communication, FUGAKU

1. はじめに

スーパーコンピュータ富岳は，158976 ノードから構成される超大規模計算機である．富岳は 2021 年 10 月現在，TOP500 の首位を取っており，その性能を活かしたアプリケーションの開発や成果の創出が期待されている．

一方で，この様な超大規模な計算機上において高い実行効率を出す事ができるプログラムを開発する事は容易ではない．特に，その性能を律速するのはノード間の通信によるコスト，その中でも `Alltoall`，`Alltoallv` や `Bcast` といった集団通信になる事が多い．そこで，富岳ではこの超大規模並列環境においても通信コストが削減できるよう，専用の TofuD[2] と呼ばれるトーラスネットワークを採用している．TofuD は 3 次元トーラスを拡張した 6 次元メッシュ構造をしている．各ノードは自分から見て $(\pm X, \pm Y, \pm Z)$ の 6 方向に物理的に別のノードと接続されており，最大でこの 6 方向に同時にデータを送受信する事が可能である．そして，ユーザーはジョブをこの 1 次元，2 次元及び 3 次元の任意の形状に割り当てて実行する事が可能である．富岳にはこの TofuD 向けにチューニングされた集団通信アルゴリズムが用意されており，これらを用いる事で TofuD の性能を最大限に活かした性能が出る事が期待される．

これらの集団通信のコストは，通信アルゴリズムだけでなく，ノードの物理的な割当形状 (ネットワーク・トポロジー) 及びパイプライン転送 [4] のセグメントサイズによっても決定される．したがって，富岳上で高い実行効率を出すためには集団通信の性能やその富岳上での性質を知り，その上でパラメータをセットする事が必要になる．実際，富岳と同じく Tofu ネットワークを用いていた前世代のスーパーコンピュータ「京」においても，パラメータ依存性の調査や，自動チューニングの性能評価などが積極的

[1] 神戸大学
Kobe University
[2] 松江工業高等専門学校
National Institute of Technology, Matsue College
[a)] natsuki.hosono@crystal.kobe-u.ac.jp
[b)] cc67803@gmail.com
[c)] makino@mail.jmlab.jp

になされており [6], [7], [8]，これらから Tofu ネットワーク上ではジョブの割当に 3 次元のトポロジーを用いる事が重要である事などがわかっている．しかしながら，こらの通信のパラメータ依存性などにに関して，ユーザー側が完全に学習し最適な物を設定をするのは必ずしも簡単ではない．そこで，富岳には自動チューニングが実装されており，ユーザー側がパラメータの選択などについて特に意識せずともある程度自動で最適なパラメータが選択される．

これらから，富岳では基本的に 3 次元トポロジーでのジョブの投入が推奨されるが，一方で 3 次元トポロジーにはジョブを投入してから実行までの時間が長いという問題もある．ユーザーがジョブの結果を得るまでの時間は，富岳上での通信性能のみならず，投入してから実行までの時間にも依存する．したがって，この投入から実行までの時間を最小化するには，通信の性能を 1 次元及び 2 次元に対しても最適化しつつ，よりスケジューリングが安易な形状指定なしでジョブを投入する事が望ましい．

そこで, 本稿では集団通信の中でも `Alltoall`, `Alltoallv` 及び `Bcast` について，通信性能のトポロジー依存性がどの様になるか，及びどう最適化するべきかに関して調べることを目的とした．具体的には，各集団通信関数に関して，送受信するメッセージサイズを変化させ，そのスループットを見積もる．その際，通信アルゴリズム，セグメントサイズ及びジョブの割当トポロジーの 3 つを変化させ，これらが通信性能にどの様な影響を与えるかを確認する．これらの結果から，1 次元および 2 次元でのジョブの割当の際でも通信の性能を最大限に発揮するためのパラメータなどに関して，しらべる．また，これらの結果を自動チューニングの結果を比較し，自動チューニングがどの程度最適な値を取るかどうかに関しても調べる．

本稿は，以下のような構成を取っている．二章では，まずは測定方法などに関して記述する．次に，三章では，本稿で調べるパラメータ，すなわちネットワーク・トポロジー，通信アルゴリズム及びセグメント分割について簡単に紹介する．四章では，結果を紹介する．最後に，本稿をまとめる．

2. 測定方法

本稿では，通信性能の評価の指標として，通信メッセージ長 M を通信にかかった時間 t で除算した，スループット T を用いる事とした．

`Alltoall` ないし `Alltoallv` では，全プロセスからの送受信が発生するため，メッセージ長に各ノードから送受信されるメッセージ長である，$(N_{\mathrm{P}} - N_{\mathrm{LP}})N_{\mathrm{LP}}$ を乗算した物を処理時間で除算した，

$$T = \frac{(N_{\mathrm{P}} - N_{\mathrm{LP}})N_{\mathrm{LP}}M}{t} \tag{1}$$

をスループットとした [5]．ただし，ここで N_{LP} は 1 ノードに存在するプロセス数である．このスループットは，メッセージ長が長い極限で実行的なバンド幅と一致する．

また，`Bcast` のスループット T は，以下の様に定義する．

$$T = \frac{M}{t} \tag{2}$$

この時，実行時間 t は，通信一回あたりのレイテンシ L と実際にメッセージ長 M を送受信するのに必要な通信時間である M/B からなる．

$$t = N_{\mathrm{R}}\left(L + \frac{M}{B}\right) \tag{3}$$

ここで，N_{R} は全体の通信を完了するのにかかった通信のラウンド数である．例えば，ルートプロセスが他のプロセス全てにメッセージをノンブロッキング送信する場合，$N_{\mathrm{R}} = N_{\mathrm{P}} - 1$ である．ただいし，ここで N_{P} はプロセス数である．

Name	1D	2D	3D
small	384	64x6	8x6x8
middle	3072	48x64	16x12x16
large	27648	144x192	48x12x48

表 1 ネットワーク・トポロジーのリスト．
Table 1 List for network topology used in this report.

3. パラメータ

富岳において，通信の性能を決定するのは，ネットワーク・トポロジー，通信アルゴリズム及びセグメントサイズである．以下では，個々について，簡単に紹介する．

3.1 ネットワーク・トポロジー

富岳では，TofuD 6 次元メッシュ/トーラスネットワークを採用しており，ユーザーがジョブを投入する際にノードの割当を 1 次元，2 次元及び 3 次元の中から指定する事が可能である．以降，この割当の際の物理的な座標軸を X，Y，Z と表記する．そして，各軸に配置されたプロセス数を N_{X}，N_{Y}，N_{Z} と表記する．そして，プロセス数を 1 次元の場合 N_{X}，2 次元の場合 $N_{\mathrm{X}}\mathrm{x}N_{\mathrm{Y}}$，3 次元の場合 $N_{\mathrm{X}}\mathrm{x}N_{\mathrm{Y}}\mathrm{x}N_{\mathrm{Z}}$ と表記する．

本稿では，3 種類のプロセス数に関して，計測を行った．**表 1** は，そのトポロジーのリストである．ただし，`Bcast` に関しては，small 及び middle での場合のみ行い，large に関しては行っていない．

また，`Alltoall` と `Alltoallv` に関しては，ノード毎に割り当てるプロセスの数に関しても調べた．具体的には，一般的によく用いられる 1 ノード 1 プロセスだけでなく，1 ノード 4 プロセスでの割当の場合も性能測定を行った．

3.2 通信アルゴリズム

富岳の MPI ライブラリは OpenMPI を元にして作られ

ており，OpenMPI 由来の通信アルゴリズムを用いることができる．この他に，富岳用にチューニングされたアルゴリズムを用いる事ができる．以下では，`Alltallv` と `Bcast` に関して，本稿で用いたアルゴリズムに関して簡単に説明する．

3.2.1 `Alltoallv`

富岳には，`Alltoallv` に 3 つのアルゴリズムが存在している．そのうち 2 つは OpenMPI にも実装されている Basic Linear と Pairwise であり，もう 1 つは富岳用に設計された Doublespread である．

Basic Linear 各プロセスが，他のプロセスに対してノンブロッキング通信で送信と受信を行う．このアルゴリズムは，1536 プロセス以下でないと使えない．

Pairwise 各ラウンド R において，プロセス P がプロセス $P-R$ からの受信と $P+R$ への送信を同時に行う．

Doublespread Simplespread アルゴリズムを用いて `Alltoallv` を実現するが，TofuD の 6 方向同時通信機能を用いたもの．このアルゴリズムは，送受信するデータ型が `MPI_PACKED` ではなく，隙間の無いデータ型である必要がある．

3.2.2 `Bcast`

富岳には，`Bcast` に 10 個のアルゴリズムが存在している．そのうち 6 つは OpenMPI にも実装されている Basic Linear，Chain，Pipeline，Split_binary_tree，Binary_tree，Binomial である [3]．残りの 4 つは富岳用に設計された Trinaryx6，Bintree3d，Trinaryx3，Bintree6d である [1]．

Basic Linear ルートプロセスが，他の全てのプロセスに対してノンブロッキング通信で送信を行う．

Chain 各ラウンド R において，プロセス $R-1$ がプロセス R へ送信を行う．

Pipeline Chain と同様であるが，各転送がパイプライン化されている．

Binary_tree ルートプロセスを根とし，プロセス全体で二分木を作る．この二分木に従って，ルートプロセスからメッセージを送信していく．

Split_binary_tree Binary_tree と同様であるが，ルートプロセスから二分木を二本作り，メッセージを二分割し，各二分木に沿ってメッセージを送信していき，最後に分割されたメッセージをマージする．

Binomial_tree 各プロセスが，各ラウンド R において，プロセス番号が $R-1$ よりも大きいランクに送信していく．

Trinaryx3 ルートプロセスを根とし，辺素なツリーを 3 次元ネットワーク上に 3 本作る．メッセージを 3 分割し，この 3 本の木を同時にたどってメッセージを送信し，最後にマージする．

Trinaryx6 Trinaryx3 と同様であるが，辺素なツリーを 6 本作り，メッセージを $(\pm\mathrm{X}, \pm\mathrm{Y}, \pm\mathrm{Z})$ 方向に同時に通信しながらツリーをたどり，最後にマージする．

Bintree3d 3 次元ネットワーク上において，まずルートプロセスを根とした二分木を X 軸方向に形成する．このルートプロセスからつながったプロセス達から，更に Y 軸の方向に二分木を形成する．更に同様に Z 軸方向の二分木も形成する．この二分木に従ってメッセージを送信する．

Bintree6d Bintree3d と同様であるが，$(\pm\mathrm{X}, \pm\mathrm{Y}, \pm\mathrm{Z})$ 方向に同時に通信する．

ここで，Basic Linear，Chain，Pipeline は，プロセス数に対して線形に通信コストが増加するアルゴリズムである．一方 Binary_tree，Split_binary_tree，Binomial_tree は，木構造を用いているため，通信コストは概ね $\log_2 N_\mathrm{P}$ に比例する．

Trinaryx3 及び Trinaryx6 は，富岳の 3 次元トポロジーを活かしたアルゴリズムとなっている．これらでは，通信時間は $N_\mathrm{P}(= N_\mathrm{X}N_\mathrm{Y}N_\mathrm{Z})$ ではなく，$N_\mathrm{X}+N_\mathrm{Y}+N_\mathrm{Z}$ に比例する [1]．これは 1 次元トポロジー上で用いてもプロセス数に応じて線形に通信コストが増加するが，3 次元トポロジーを用いる事で大幅な通信コストの削減が可能となる．一方で，Bintree3d 及び Bintree6d も富岳の 3 次元トポロジーを活かしたアルゴリズムであるが，これらは二分木構造を用いており，通信コストは $\log_2 N_\mathrm{X} + \log_2 N_\mathrm{Y} + \log_2 N_\mathrm{Z}$ となる．

3.3 パイプライン転送

パイプライン転送 [4] では，メッセージを m_S バイトのサイズを持ったセグメントに分割する．送信プロセスは，最初のセグメントを送信し，受信プロセスはそれを受信する．受信プロセスはその後，受け取った最初のセグメントを次の受信プロセスに送信するとともに，残りのセグメントを受信する．

このパイプライン転送を用いる事で，通信のコストを削減する事が可能であるが，最適なセグメント長のサイズに関しては，通信アルゴリズム及びメッセージ長に依存する事に注意する．例えば，Trinaryx6 を考えよう．Trinaryx6 は，辺素なツリー構造を作り，その構造に従ってメッセージを送信していくアルゴリズムである．その際，メッセージを 6 分割し，$(\pm\mathrm{X}, \pm\mathrm{Y}, \pm\mathrm{Z})$ 方向に同時に通信する．したがって，その通信コストは，Trinaryx3 の通信コスト t[1], [8] を 3 方向から 6 方向にしたものとなるので，以下の様に見積もられる．

$$t = \left(N_\mathrm{X} + N_\mathrm{Y} + N_\mathrm{Z} + \frac{M}{6m_\mathrm{S}} - 3\right)\left(L + \frac{m_\mathrm{S}}{B}\right) \qquad (4)$$

ただし，ここで m_S はセグメント長である．この通信コストを m_S で微分し，それが 0 になる m_S を求める事で，m_S の最適値 $m_\mathrm{S}^\mathrm{opt}$ を求めることができる [4]．

$$m_{\mathrm{S}}^{\mathrm{opt}} = \sqrt{\frac{LBM}{6\left(N_{\mathrm{X}} + N_{\mathrm{Y}} + N_{\mathrm{Z}} - 2\right)}} \tag{5}$$

ここで，通信のレイテンシ L 及びバンド幅 B は，用いたアルゴリズム及びノードのトポロジーの影響を受けるため，測定値をフィットするなどして値を得る必要がある．

また，Bintree3d の通信コストは以下のように見積もられる [1].

$$t = \left(\lfloor \log_2 N_{\mathrm{X}} \rfloor + \lfloor \log_2 N_{\mathrm{Y}} \rfloor + \lfloor \log_2 N_{\mathrm{Z}} \rfloor + \frac{M}{m_{\mathrm{S}}} - 1\right) \times \left(L + \frac{m_{\mathrm{S}}}{B}\right) \tag{6}$$

最適セグメント長も同様に，

$$m_{\mathrm{S}}^{\mathrm{opt}} = \sqrt{\frac{LBM}{\lfloor \log_2 N_{\mathrm{X}} \rfloor + \lfloor \log_2 N_{\mathrm{Y}} \rfloor + \lfloor \log_2 N_{\mathrm{Z}} \rfloor - 1}} \tag{7}$$

となる．

また, このパイプライン転送は `Alltlall` 及び `Alltlallv` ではサポートされていないため，以降では `Bcast` のみこのセグメント長の影響を考える事にする．

4. 結果

本節では，結果を報告する．まずは `Alltoall` 及び `Alltoallv` の性能のノード・トポロジー及び通信アルゴリズム依存性を調べた後，`Bcast` のノード・トポロジー依存性，通信アルゴリズム依存性及びセグメント長への依存性を調べる．

4.1 `Alltoall`/`Alltoallv`

まずは, `Alltoall` 及び `Alltoallv` の性能のノード・トポロジー依存性を調べる．**図 1** は, `Alltoall` 及び `Alltoallv` を small，middle 及び large の各トポロジーにおいて実行した際のスループットを表示した物である．アルゴリズムはすべて Doublespread を用いた．また，ノード毎のプロセス数は 1 とした．これらから，いずれのノード数及びトポロジーに関しても，メッセージ長が大きい極限では，1 GB/s 程度の性能が出る事が読み取れる．また，ノード数があまり多くない small では，ノードのトポロジーによらず性能は概ね同じであった．一方，middle の場合は，1 次元でのトポロジー設定は，2 及び 3 次元での場合に比べて性能が劣る場合が見られ，特に `Alltoallv` ではメッセージ長によらず 2 及び 3 次元でのトポロジーが高い性能となった．同様の傾向は large での場合にも見られ，`Alltoall` に関してはメッセージ長が短い場合にはわずかに 3 次元形状が 1 及び 2 次元に対して低い性能を出す事も見られるが，メッセージ長が長い場合は 3 次元が最も良い性能となった．

図 2 は，図 1 と同様の計測を，1 ノード 4 プロセスで行ったものである．図 1 と同様に，small の場合ノード・トポロジーによる結果の違いはほぼ見られなかった．まだ，同様に middle での場合，3 次元トポロジーが最も良い性能になり，1 及び 2 次元の間に大きな性能の違いは見られなかった．しかしながら，large の場合，図 1 と違い各トポロジー間での性能の違いはほぼ見られない結果となった．

これらから，`Alltoall` 及び `Alltoallv` に関しては限定的な状況，すなわちプロセス数が大きくメッセージ長が短い状態で `Alltoall` を使う場合を除いては，3 次元のトポロジーを明示的に指定するのが最も良い選択であると言える．ただし，ノード数があまり大きくない場合 (small など) や，メッセージサイズが大きい場合などでは，トポロジーの影響をあまり強く受けず，1 次元又は 2 次元を選んでも問題無い場合もある．そのため，これらトポロジーにより性能に大きな変化が無い場合では，ノード形状指定無しでジョブを投入しても大きな問題は無い．

次に，通信アルゴリズムがどの様に性能に影響するかを調べる．**図 3** は，$N_{\mathrm{LP}} = 4$ の場合において，`Alltoallv` の通信アルゴリズムを変更した際のスループットの値である．ただし，Basic Linear は，middle 及び large の場合適用可能なプロセス数を超えているため，表示されていない．まずノード数が小さい small の場合，メッセージ長が短い場合 ($M \leq 128$ [KiB]) は Basic Linear が最適となるが，それ以上のメッセージ長では Doublespread が最も良い性能となった．また，ノード数が middle 及び large の場合，メッセージ長によらず Doublespread が最適なアルゴリズムとなった．

これらから, 通信アルゴリズムに関しては, Doublespread を選択するのが概ね最適であると言える．ただし，Basic Linear が使えるノード数であり, メッセージ長が短い場合は Basic Linear を用いたほうが良い．富岳では，`Alltoallv` の通信アルゴリズムを特に指定しない場合，Doublespread が用いられる．しかしながら，上述の通り Doublespread には「データ型が `MPI_PACKED` ではなく，隙間の無いデータ型である」という適用条件があり，アプリケーション側で通信しているデータ型がこの条件に適合していると判断されたかどうかに関しては注意する必要がある．もし適用できないと判断された場合，Doublespread は破棄されて代わりに Pairwise が適用されるため，通信の性能がメッセージ長によっては数倍〜一桁程度落ちる可能性がある．

4.2 Bcast

次に，`Bcast` の性能に関して調べる．

まずは，`Bcast` において，自動チューニングを用いた際に選択される通信アルゴリズムとパイプライン分割のセグメント長に関して調べる．**図 4** は，3072 ノード 1 次元トポロジーにおいて，`Bcast` のスループットを計測したものである．その結果，富岳の自動チューニングでは，メッセージ長が短い場合 ($M \leq 512$ [B]) は Binary_tree が，中程度の場合 (1 [KiB] $\leq M \leq$ 32 [KiB]) は Bintree3d が，大

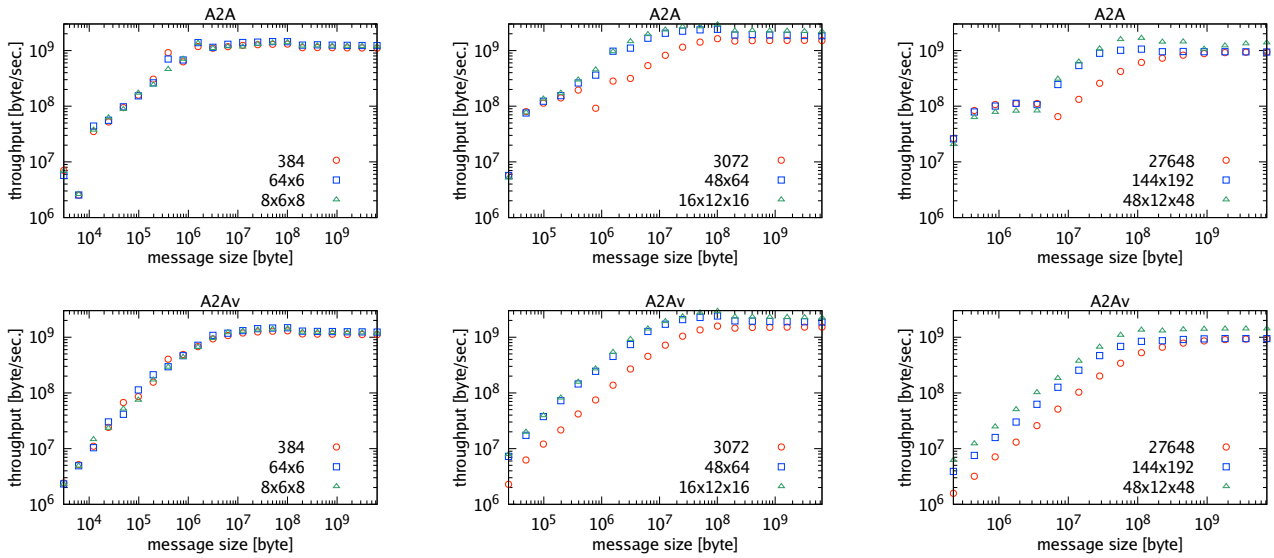

図 1 Alltoall(v) のスループットを測定した結果．横軸は通信されたメッセージのサイズで，縦軸はスループット．1 ノードあたりのプロセス数は 1 である．左から順番に small，middle，large ノード数での結果で，赤い丸が 1D，青い四角が 2D，緑の三角が 3D でのトポロジーでの結果．上段は `Alltoall` の結果，下段は `Alltoallv` の結果である．

Fig. 1 Throughput vs. message size are shown. The number of processes per each node are set to 1. The top row shows the results of `Alltoall`, whereas the bottom row shows those for `Alltoallv`. From left to right, small, middle and large models are shown. The red circles, blue squares, and green triangles indicate the results with 1D, 2D, and 3D topology, respectively.

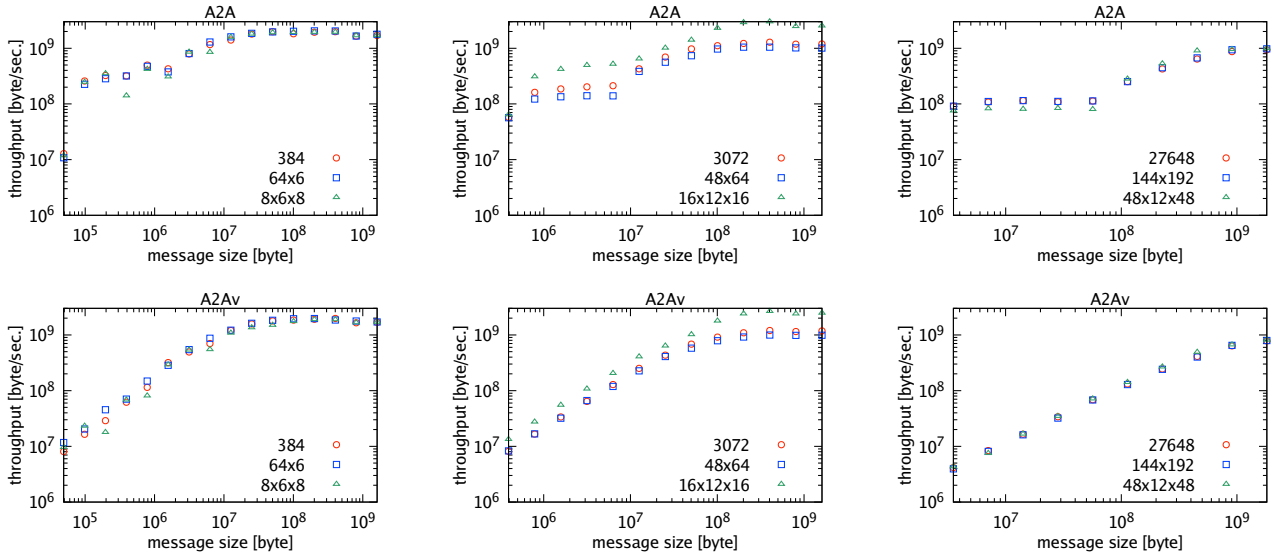

図 2 図 1 と同様であるが，1 ノード 4 プロセスで行った結果．

Fig. 2 Same as Figure 1, but with 4 processes per each node.

きい場合 ($M \geq 64$ [KiB]) は Trinaryx6 が選択された．また，アルゴリズムが Bintree3d から Trinaryx6 に変わる際に，スループットの大きな低下が起きる事がわかる．

次に，通信性能のアルゴリズム依存性に関して，調べる．**図 5** 及び**図 6** は，`Bcast` のスループットを small 及び middle ノード数に対して通信アルゴリズムを変えて計測したものである．ただし，セグメント長に関しては指定しておらず，パイプライン分割は行われていない結果である．Trinaryx3 及び Trinaryx6 は，木構造を用いないアルゴリズムであるため，1 次元形状の場合は N_{P} に比例する汎用アルゴリズム (Basic Linear，Chain，Pipeline) と概ね変わらない性能となった．一方で，2 次元，3 次元形状になる

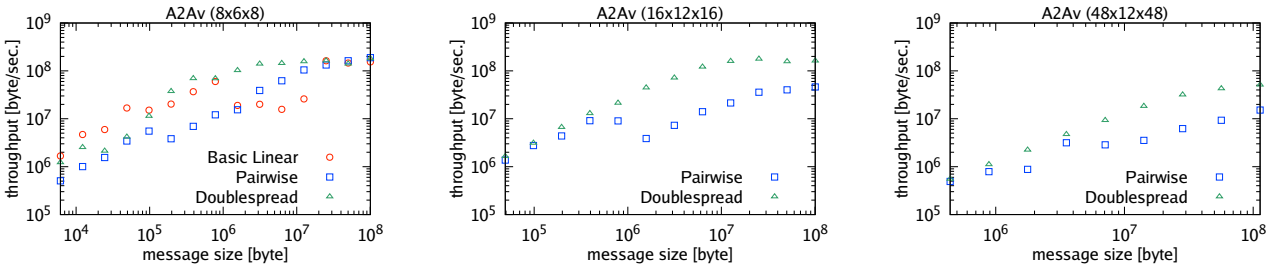

図 3 `Alltoallv` のスループットを，Basic Linear(赤い丸)，Pairwise(青い四角) 及び Doublespread(緑の三角) で比較した図．左から順番に，small，middle，large の 3 次元トポロジーを指定した．1 ノードあたりのプロセス数は 4 である．

Fig. 3 Throughput vs. message size for each algorithm are shown. The number of processes per each node are set to 4. From left to right, small, middle and large models with 3D node topology are shown. The red circles, blue squares, and green triangles indicate the results with Basic Linear, Pairwise, and Doublespread, respectively.

につれ，TofuD の物理的 3 次元構造による通信コストの削減が効き，性能が大幅に上昇する．同様に，木構造ベースである Bintree3d も，他の汎用木構造ベース (Binary_tree, Split_binary_tree, Binomial) のアルゴリズムと同様の通信コストとなったが，こちらは形状を多次元にしても大きな変化は無かった．これから，1 次元形状の場合はメッセージ長が大きくなるところ，特にアルゴリズムが Trinaryx6 に変化する所で自動チューニングにより選ばれた Trinaryx6 よりも Bintree3d などの方が性能が良くなる事が見て取れる．これは，Trinaryx6 が 3 次元形状を前提に作られているアルゴリズムで，1 次元形状ではその性能を十分に活かすことができないためである．一方，2 次元及び 3 次元では，多次元形状特有の多方向同時通信から，Trinaryx6 はその性能を大きく上げるため，アルゴリズムが変化する部分でスループットの低下を招かない．

自動チューニングで選ばれた Trinaryx6 と，明示的に Trinaryx6 を指定した場合では，スループットに違いが出ている．これは，セグメントサイズの違いの効果である．**図 7** は，3072 ノードで rinaryx6 を用いた際のスループットで，セグメント長を変化させた際の物である．明示的に Trinaryx6 を指定し，かつセグメント長を自動チューニングが選んだ値と一致させたものは，自動チューニングでの結果と一致する．

これらから，メッセージ長が長く，1 次元形状を用いる際は自動チューニングを用いるのではなく，明示的にアルゴリズムとセグメント長を指定する事が重要である．また，トポロジーによらず，メッセージ長が短い場合は自動チューニングは Bintree3d や Bintree6d に比べてわずかにスループットが低くでる．したがって，3 次元形状を用いる際も，アルゴリズム及びセグメント長の明示的な指定を行う事で，コストの削減を行う事が可能である．

図 8 は，各通信メッセージ長 M に対し，m_S を変えていった際の通信コスト (式 4 及び 6) をプロットした物である．レイテンシ L やバンド幅 B の値は，実際の計測から得られた値をフィットして得られた値を用いた．具体的には，Trinaryx6 の場合は $L = 1.73$ [μs] 及び $B = 6.34$ [GB/s]，Bintree3d の場合は $L = 4.29$ [μs] 及び $B = 6.64$ [GB/s] とした．Trinaryx6 の場合，式 5 は通信コストを最小化するセグメント長を良く予想できている事がわかる．一方で，Bintree3d の場合，実際に通信コストを最小化するセグメント長は，式 7 よりも小さい値になった．しかしながら，このセグメント長での Bintree3d の通信コストへの依存性は大きくなく，式 7 での値を指定しても大きくつ通信コストが増える事も無いため，実用上は式 7 での値を指定して問題ないと考えられる．

5. まとめ

本稿では，富岳上において `Alltoall`，`Alltoallv` 及び `Bcast` に関して性能を測定し，最適な設定に関して調査した．その結果，いずれの場合でも，ノードのトポロジーは 3 次元にすると最も性能が出た．しなしながら，富岳のリソースを有効に使うという観点からは，より柔軟なスケジューリングが可能な形状指定なしでジョブを投入する事が望まれる．したがって，1 次元及び 2 次元の場合にも性能が最適化されるようなパラメータ設定をする事が重要である．

富岳の集団通信には openMPI で実装されている汎用アルゴリズムに加え，富岳専用の通信アルゴリズムが存在する．この通信アルゴリズムを明示的に指定しない場合，自動的に最適な物が選択される．実際，`Alltoallv` の場合，通信アルゴリズムに自動的に富岳用のアルゴリズムである Doublespread が選択された．ただし，Doublespread には適用条件が存在し，通信する型が MPI_PACKED では無く，隙間のないデータ型である必要がある．自動チュー

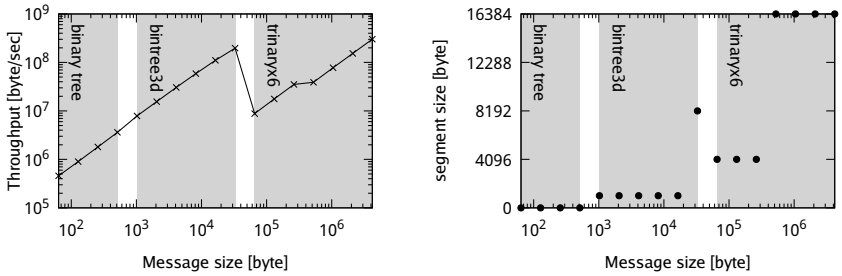

図 4 左のパネルは Bcast を，富岳の自動チューニングを用いて動かした際のスループットである．また，右側の図はそれぞれのメッセージ長において，用いられたセグメントサイズを示している．ただし，セグメントサイズが 0 とは，セグメント分割を行わない事を意味する．ノード数は small でトポロジーは 1 次元を用いた．各グレーのハッチは動作しているアルゴリズムであり，一番左の領域が Binary_tree，中央が Bintree3d，右が Trinaryx6 で動作している．

Fig. 4 The left panel shows throughput vs. message size and the right panel shows the segment size vs. message size on the auto-tune. The number of nodes and topology are middle and 1D. From left to right, Binary_tree, Bintree3d, and Trinaryx6 work on each hatched region.

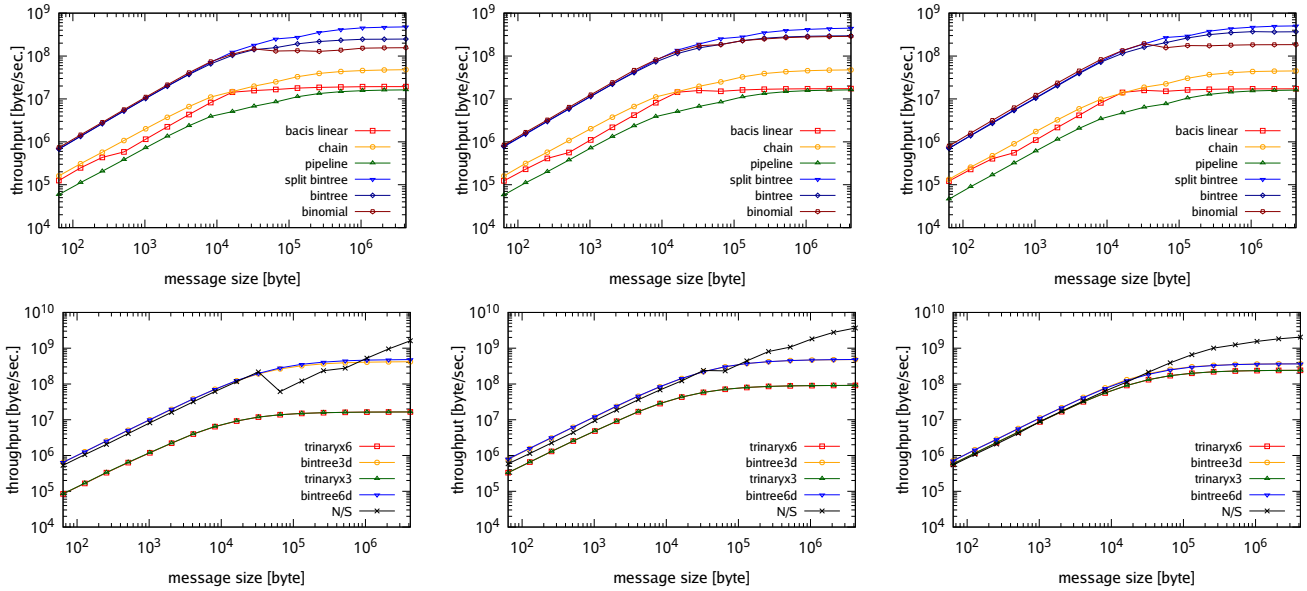

図 5 Bcast のスループットを測定した結果．横軸は通信されたメッセージのサイズで，縦軸はスループット．1 ノードあたりのプロセス数は 1 である．ノード数は small で，左から順番に 1 次元，2 次元及び 3 次元での結果であり，点の種類は通信アルゴリズムを示している．上段は汎用アルゴリズムの結果であり，下段は TofuD 専用アルゴリズムの結果である．

Fig. 5 Throughput vs. message size of Bcast on various algorithms. The number of process per node is 1. From left to right, results of small nodes with 1D, 2D, and 3D topology are shown.

ニングの際に，Doublespread が適用出来ないと判断され Pairwise で通信が行われると，スループットの大きな低下を招く．そのため，実際に自分のプログラムがどのアルゴリズムを用いて通信しているのか，確認しておく必要がある．

また，Bcast 場合は，自動チューニングにより通信アルゴリズム及びセグメント長が自動的に選ばれる．ノードトポロジーが 3 次元及び 2 次元の場合は自動チューニングでも概ね問題の無い性能が出たが，1 次元になると自動チューニングはメッセージ長の長い場合にスループットの不要な低下を招く事がわかった．これは，選択されるアルゴリズムの一つである Trinaryx6 が，トポロジーの選択に強く影響を受け，1 次元と 3 次元では性能が大きく違うからである．したがって，1 次元でのトポロジーで計算を行う際は，

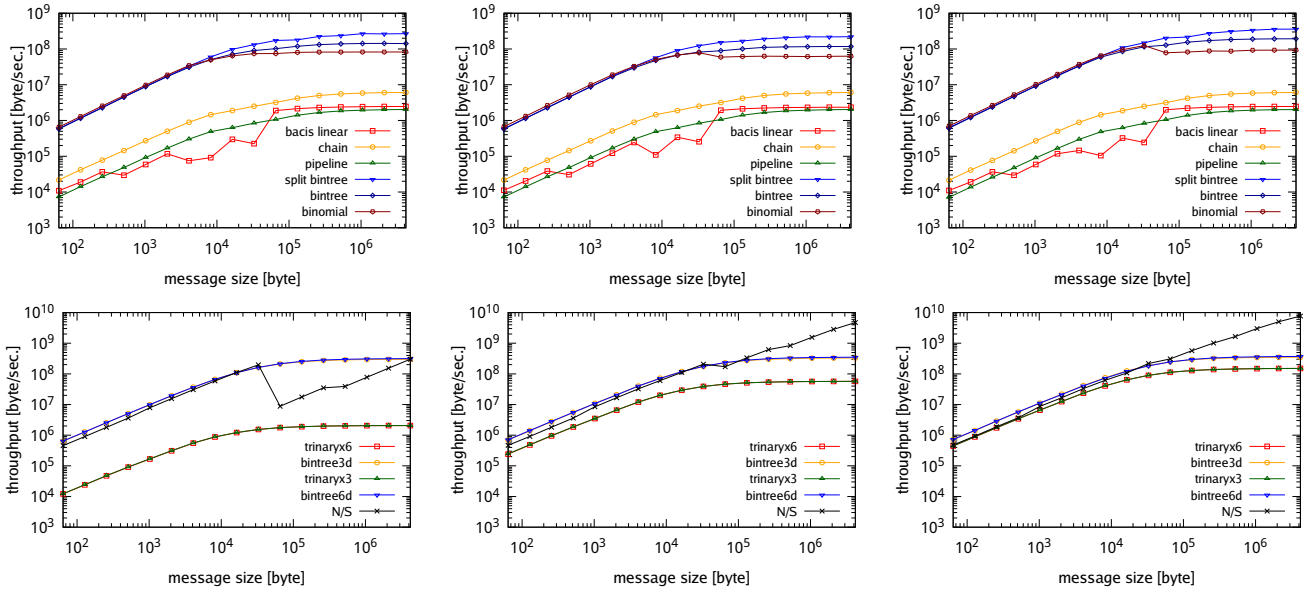

図 6 図 5 と同じであるが，プロセス数が middle になったもの．
Fig. 6 Same as 図 5, but with middle model.

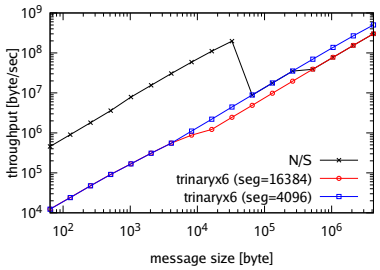

図 7 `Bcast` のスループットを，セグメント長を変えて計測したもの．プロセス数は middle でトポロジーは 1D である．黒いクロスはアルゴリズム及びセグメント長を指定しなかった際の結果，すなわち自動チューニングの結果である．その他の点のアルゴリズムは全て Trinaryx6 であり，赤い丸はセグメント長 $m_S = 16384$，青い四角は $m_S = 4096$ の結果である．
Fig. 7 Throughput vs. message size of `Bcast` on various segment size. The number of processes is middle and the topology is 1D. The black crosses indicate the results with auto-tune. The red circles and blue squares are those with $m_S = 16384$ and $m_S = 4096$ and Trinaryx6, respectively.

Bintree3d といった 1 次元でも速いアルゴリズムを明示的に適用しておく事が性能を出す上では重要である．また，1 次元の場合に限らず，2 次元または 3 次元の場合でも，通信コストの予測式から m_S^{opt} を導出し，明示的にアルゴリズム及びセグメント長の値をセットする事で，より良い性能を出す事が可能であると予想される．

本原稿では，`Alltoall` と `Alltoallv`，及び `Bcast` の 3 つを集中的に調べたが，他の通信アルゴリズムに関しても同様の事が言えると予想される．したがって，富岳で計算を行う際は，ノードトポロジーを 3 次元にして行う事が推奨されるが，リソースなどの観点から 1 次元または 2 次元で行う場合は，個々の通信のアルゴリズムやセグメント長などへの依存性を十分に理解した上で明示的にこれらの要素を設定する事が求められる．

謝辞 本研究は，HPCI システム利用研究課題 (課題番号：hp120286) を通じて，理化学研究所計算科学研究センター所有のスーパーコンピュータ「富岳」を用いて行われたものです．

参考文献

[1] Adachi, T., Shida, N., Miura, K., Sumimoto, S., Uno, A., Kurokawa, M., Shoji, F. and Yokokawa, M.: The design of ultra scalable MPI collective communication on the K computer, *Computer Science - Research and Development*, Vol. 28, pp. 147–155 (2012).

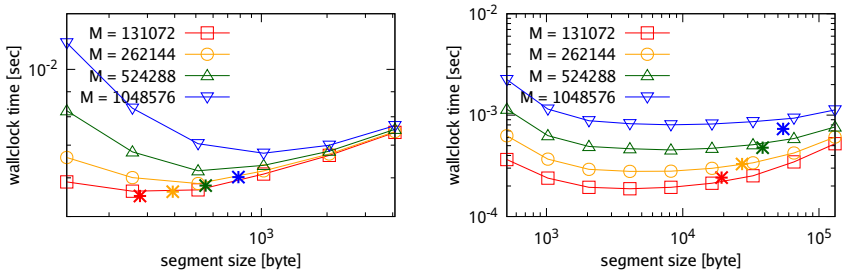

図 8 `Bcast` の通信時間を，セグメント長 m_{S} を変えてプロットした物である．用いたプロセス数は 3072 である．左のパネルはアルゴリズムが Trinaryx6 もので，で右のパネルは Bintree3d ある．点と色の違いは送受信するメッセージ長の違いである．アスタリスクは，対応する色の $m_{\mathrm{S}}^{\mathrm{opt}}$ と，その際の通信時間の理論値である．

Fig. 8 Wall-clock time vs. segment size is shown. The number of processes is 3072. In the left and right panel, Trinaryx6 and Bintree3d are used, respectively. The colors and points correspond to the message size. The asterisks correspond to $m_{\mathrm{S}}^{\mathrm{opt}}$ and theoretical prediction for communication cost at $m_{\mathrm{S}} = m_{\mathrm{S}}^{\mathrm{opt}}$.

[2] Ajima, Y., Kawashima, T., Okamoto, T., Shida, N., Hirai, K., Shimizu, T., Hiramoto, S., Ikeda, Y., Yoshikawa, T., Uchida, K. and Inoue, T.: The Tofu Interconnect D, *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 646–654 (online), DOI: 10.1109/CLUSTER.2018.00090 (2018).

[3] Nuriyev, E. and Lastovetsky, A.: Efficient and Accurate Selection of Optimal Collective Communication Algorithms Using Analytical Performance Modeling, *IEEE Access*, Vol. 9, pp. 109355–109373 (online), DOI: 10.1109/ACCESS.2021.3101689 (2021).

[4] Watts, J. and van de Geijn, R. A.: A Pipelined Broadcast for Multidimensional Meshes, *Parallel Process. Lett.*, Vol. 5, pp. 281–292 (online), DOI: 10.1142/S0129626495000266 (1995).

[5] 彰　成瀬，耕太中島，真司住元，耕一久門：マルチコア PC クラスタ向け All-to-all 通信アルゴリズムの提案と評価，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 3, No. 3, pp. 166–177 (2010).

[6] 南里豪志，深沢圭一郎：Tofu ネットワークにおけるプロセス配置形状による集団通信アルゴリズムの性能解析，技術報告 25，九州大学情報基盤研究開発センター／ JST CREST, 九州大学情報基盤研究開発センター／九州大学国際宇宙天気科学・教育センター／ JST CREST (2012).

[7] 好人北澤，明義黒田，直之志田，知也安達，一生　南：スループットを用いた「京」における MPI 通信性能の評価，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 10, No. 3, pp. 1–11 (2017).

[8] 好人北澤，明義黒田，一生　南，文由庄司：スーパーコンピュータ「京」における MPI 通信性能の評価，技術報告 34，理化学研究所, 理化学研究所, 理化学研究所, 理化学研究所 (2016).