Vol.2020-HPC-175 No.10
2020/7/30

MPI環境下での動的マルチキャストのための配信アルゴリズムのパラメータ最適化

山田　竜輝[1]　福間　慎治[1]　森　眞一郎[1]

概要：大規模並列環境下で計算資源を有効活用する手段として, マスター・ワーカー型の並列処理を採用している. 並列処理を行う際に, マスターノードからある特定のワーカーノード群に対してメッセージの送信を行う動的マルチキャストを実装している. 並列処理の性能向上のために, 動的マルチキャストの高速化が必要となる. そこで, 配信アルゴリズムに注目し, 新たな配信アルゴリズムを含めた様々な実行環境下で性能を比較し, アルゴリズム broadcast hybrid d の有効性を確認した. さらに, 配信アルゴリズムごとの通信コストを計算することで, 実行環境下によって変化する最適な配信アルゴリズムの推定も行い, 一定の成果が得られたと考える.

An Implementation of Dynamic Multicast Function under MPI Environment and its Parameter Optimization

1. はじめに

近年, プロセッサのメニーコア化やシステムの大規模環境並列化が進み, 潤沢な計算資源が提供されるようになった. 我々は, その潤沢な計算資源を有効活用するために, MPI[1] をベースとしたマスター・ワーカー型の並列処理機構を採用し, マスターノードが特定のワーカーノード群に対してタスクを割り当てることで並列処理を行っている. この時, マスターノードからある条件が成立する一部のワーカーノード群のみに対してマルチキャストが必要となる. MPI では集団通信手法としてブロードキャスト (一対全通信) のメカニズムが提供されており, あらかじめ指定したグループに属す全てのノードにブロードキャストする機能を利用してマルチキャストを実装することが可能である. しかしながら, グループに分割する際にグループの分割に関係ないノードも含め全ノードでの同期処理が必要である. このような、本質的には不必要な全体同期は, 我々が想定するグループの構成が頻繁に変動する環境下では並列処理効率の著しい低下を招いてしまう. MPI を用いたマスター・ワーカー型の大規模並列処理環境下で, 実行時の動的かつ頻繁なグループ変更に対応可能な動的マルチキャストの実装がおこなわれた (図 1). また, この先行研究では, 配信先ノード数やメッセージサイズを考慮して配信アルゴリズムを切り替えることによる高速化の可能性が示唆された.

本論文では, 先行研究で実装された複数の配信アルゴリズムに新たな選択肢を加えるとともに, 実行時の計算機環境に依存するパラメータも考慮した配信アルゴリズムの自動選択のための予測モデルを設計し, それらの有効性の検証を行った結果を報告する.

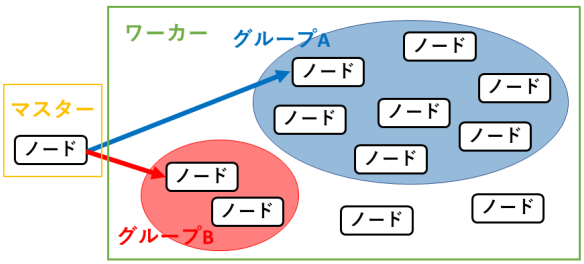

図 1　動的マルチキャスト

2. 研究関連

2.1 動的マルチキャスト

本研究で使用する MPI 通信ライブラリでは, 並列処理を行う全ノードグループのサブセットとなるグループを定義したのち, 当該グループ内の全ノードに対してブロードキャストを行うことでマルチキャストを実現することが可

[1] 福井大学大学院工学研究科知識社会基礎工学専攻

能である．しかしながら, この手法ではグループに分割する際にグループの分割に関係ないノードも含め全ノードでの同期処理が必要であり, 我々が想定している環境に適用するとことは不可能である．

そこで, 先行研究 [2] ではビットマップを用いた手法により実行時の動的かつ頻繁なグループ変更に対応可能な動的マルチキャストの実装が提案された．実装方法として, マスターノードはマルチキャストを行うワーカーをリストアップする際, マルチキャスト内での通し番号 (以下より論理番号) を各ノードに割り当てる．MPI により設定された各ノードに対応するノード番号（ランク）とマルチキャスト内の論理番号は異なるため, これらを関連付けするためのリストを作成する．このリストは, マルチキャストの受信対象となるノードのランクに対応した bit に 1 を, それ以外に 0 を記録して表現されるものであり, このリストをビットマップと呼ぶ (図 2)．ビットマップを受け取ったノードは, 自身が属するグループのメンバ構成（木構造のトポロジー）を把握するとともに, グループ内の論理番号と MPI のランク番号との対応表を構築する．自らがグループ内で木構造の葉ノードでないことを認識したノードは, 受け取ったビットマップデータを木構造の子ノードに転送する．全メンバがビットマップを受け取った時点で論理的なマルチキャストグループの構築が完了する．これ以降は, 構築されたグループ内の全ノードへのブロードキャストを行うことで動的マルチキャストを実現する．この, グループ内全ノードへのブロードキャストのアルゴリズムを本論文では「配信アルゴリズム」と呼ぶ．

先行研究では, この配信アルゴリズムとして, メッセージサイズにより binomial-tree[2] と binomial-tree_scatter_ring_allgather[2] の 2 種のアルゴリズムを実装し, それらを切り替えて使用する方式が提案されていた．なお, 先行研究ならびに本研究では, 1) マルチキャストの受信対象となり得るワーカーノードからの MPI メッセージを受信可能な状態で待機中, 2) 一度に所属するグループは 1 グループのみ, 3) グループ変更は数 10ms から数 100ms の頻度で発生, ならびに, 4) 異なるマルチキャストグループへのマルチキャストはオーバーラップ可能, という前提のもと行っている．

図 2 ビットマップデータ

2.2 実装した配信アルゴリズム

本節では, 先行研究で実装を行った配信アルゴリズムである 1)binomial-tree および 2)binomial-tree_scatter_ring_allgather と本研究で新たに実装を行った 3)binomial-tree_scatter_doubling_allgather および 4)broadcast hybrid d の内, 1) と 4) のアルゴリズムについて通信コストのモデリングを行う．この際, 相互通信する上で 1byte のデータ送信に要する時間を M [sec]，MPI 関数の起動コストを含むノード間の遅延を S [sec] とし，集団通信に参加するノード数を p，送信するメッセージ長を n [byte], hybrid d でのメッセージ分割数を d(d ≦ p, p(mod d) = 0 , d は 2 の累乗数) とする．また, 木構造ベースの配信アルゴリズムでの木の高さや通信ステップを stage と表現する．また, 実システムではノードペア毎に通信遅延が異なる可能性が高いが, これを考慮するためには実行時の物理的なノード間トポロジー推定等が必要となるとともに, コストモデル自体が複雑になりすぎるため, 本研究では均質環境 (いかなるノード間でも通信遅延は同じ) を仮定する [3].

2.2.1 binomial-tree_scatter_ring_allgather

このアルゴリズムは, 配信すべきデータをあるルールに従って分割し配信先ノードに分配する scatter 部と, 全ノードが他ノードに分配されたデータをあらためて収集することで, 元データを入手する allgather 部の 2 部で構成される．このとき allgather を ring アルゴリズムによって行うものが binomial-tree_scatter_ring_allgather(以下, ring と呼ぶ) である．scatter 部では, 分配すべきデータを p 個に等分割したデータの p 番目のデータが, 論理番号 p のノードに配置されるような分配を行う．次に, allgather 部では p 台のノードで論理的な ring ネットワークを構築し p 台のノード全てが自らが所有するデータを ring 内で循環シフトさせながら必要なデータを収集することで全ノードで元データを復元する．scatter 部ならびに ring アルゴリズムによる allgather 部の通信コストは以下のとおりである．

【scatter 部】 2 分木構造に従って, stage 1 では最上階となる root からサイズが n/2 のメッセージがひとつ下の階層へと送信される．stage i ではサイズ $n/2^i$ に分割されたメッセージがひとつ下の階層へとそれぞれ分配されていく．メッセージを受信した中継ノードは受信後は送信ノードとなり, 同様に自身の配下となるノードに分配する．この手順をすべてのノードが行うことでサイズ n/p のメッセージが各ノードに分配されたことになる (図 3)．よって, scatter 部での通信コストは

$$\frac{(p-1)}{p}nM + S\log_2 p \qquad (1)$$

となる．

【allgather（ring）部】各ノードにメッセージの一部を分配した後，ring アルゴリズムによって収集しメッセージを復元する．ring アルゴリズムでは, まず最初に自分が所有する n/p [byte] のメッセージを隣接ノードのいずれか一

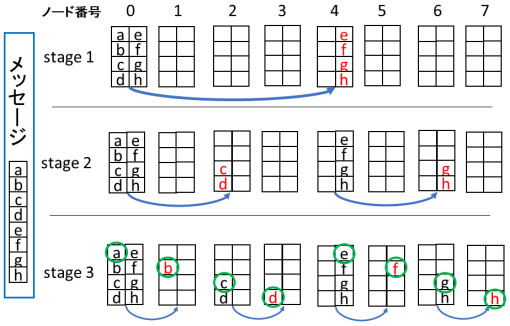

図 3 ring【scatter 部】

方のノードに送信する (図 4 では, node i が node (i+1)%p に送信). それ以降 p - 2 回, 各ノードは隣接ノードから受け取った n/p [byte] のメッセージを反対側の隣接ノードへ配信する. したがって, allgather 部の通信コストは

$$\frac{(p-1)}{p}nM + S(p-1) \tag{2}$$

となる.

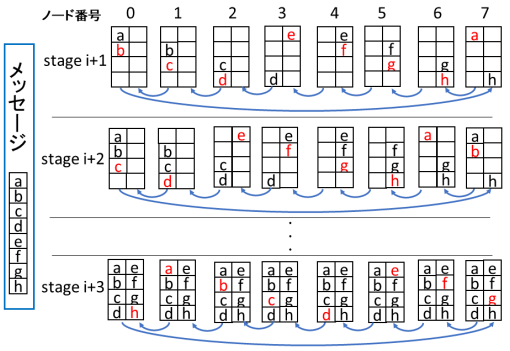

図 4 ring【allgather 部】

上記で記した scatter 部 (式 1) と allgather 部 (式 2) を合算した binomial-tree_scatter _ring_allgather の通信コストは

$$\frac{2(p-1)}{p}nM + S(\log_2 p + (p-1)) \tag{3}$$

となる.

2.2.2 broadcast hybrid d

broadcast hybrid d(以下, hybrid d と呼ぶ) は, 文献 [4] で提案されていたアルゴリズムである. hybrid d は, scatter 部と broadcast 部と collect 部の 3 部で構成されている. ring ではメッセージを 1/p に分割し配信を進行していくが, hybrid d では分割数 d(通常は p の約数) を指定し, 交換されるメッセージサイズが小くなり過ぎることを回避して速度向上を目指したアルゴリズムである.

まず最初に、p ノードを d 台毎にグルーピングしたグループ G_i ($0 \leqq i < (p/d)$) のうち, まず最初に G_0 の d 台のノードに, 元データの 1/d のデータを分配する (Scatter 部). ここでは ring アルゴリズムの scatter 部と同様の方法で分配を行う. 次に, G_0 の $j(0 \leqq j < d)$ 番目のノードに分配されたデータを, 他の全てのグループの j 番目のノードにコピーする (broadcast 部). これにより, 全てのグループにおいてグ元データを復元するために必要な d 個の分割データがループ内の d 台ノードの何れかに配置されることとなる. 最後に, 各グループ内で doubling アルゴリズムを用いてデータの収集を行う. doubling アルゴリズムは同じデータを持っていないノード同士でペアをつくりデータの相互交換を行うことで, 毎 stage ごとに送受信するデータサイズを倍々に増やすことを繰り返すことで元データを全ノードで復元する. scatter 部, broadcast 部ならびに doubling アルゴリズムによる collect 部の通信コストは以下のとおりである.

【scatter 部】分割数 d と同じノード数毎にグループを作成する. メッセージを所有するグループだけで 2 分木を作成する. 作成した 2 分木に従い, stage 1 では最上階となる root からサイズが n/2 のメッセージがひとつ下の階層へと送信される. stage i ではサイズ $n/2^i$ に分割されたメッセージがひとつ下の階層へとそれぞれ分配されていく. メッセージを受信した中継ノードは受信後は送信ノードとなり, 同様に自身の配下となるノードに分配する (図 5). この手順をノード数 d だけで行うことで, ノード数 d にそれぞれ 1/d のメッセージが分配された状態となる. stage 数は, 2 分木の高さであるため $\log_2 d$ となる. したがって, scatter 部の通信コストは

$$\frac{(d-1)}{d}nM + S\log_2 d \tag{4}$$

となる.

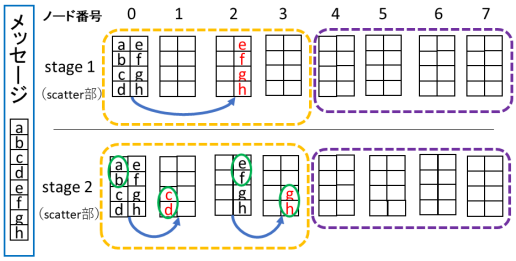

図 5 hybrid d (d=4 の例)【scatter 部】

【broadcast 部】グループ単位での 2 分木を作成し, root となるグループ 0 を最上階として下の階層へ 2 分木的に分配を行っていく. 送信するメッセージサイズは一定であり, n/d[byte] のメッセージを所有するグループ内のノードがそれぞれ 2 分木に従って他のグループのノードに対して送信を行う (図 6). グループ数は, p/d となる. よって stage 数は, グループ単位での 2 分木の高さ $\log_2(p/d)$ となる. したがって, broadcast 部の通信コストは

$$(\frac{1}{d}\log_2\frac{p}{d})nM + S\log_2\frac{p}{d} \tag{5}$$

となる.

【collect 部】グループ内で doublilng アルゴリズムによりメッセージの収集を行う. doubilng アルゴリズムでは,

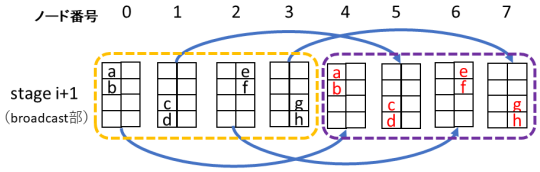

図 6 hybrid d（d=4 の例)【broadcast 部】

stage(i+j+1) のとき自身が所有する n/p[byte] をグループ内の隣接ノード間でペアをつくりノード間でメッセージの送受信を行う (図 7). stage i+j+k($1<k \leqq \log_2 d$) のとき受信したメッセージを含めて (n/d) * 2^{k-1}[byte] サイズのメッセージを 2^{k-1} 離れたノードと双方向通信を行う. stage 数は $\log_2 d$ となる. したがって, collect 部の通信コストは

$$\frac{(d-1)}{d}nM + S\log_2 d \qquad (6)$$

となる.

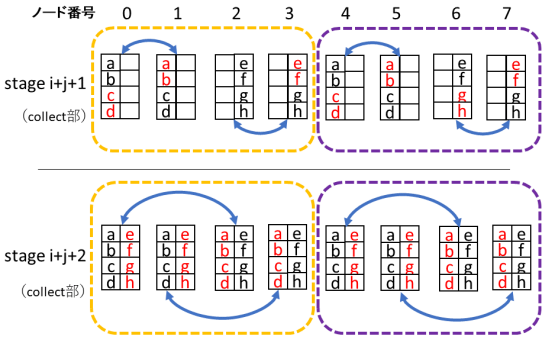

図 7 hybrid d（d=4 の例)【collect 部】

上記の scatter 部 (式 4), broadcast 部 (式 5) ならびに, collect 部 (式 6) を合算した hybrid d の通信コストは

$$(\frac{2(d-1)}{d} + \frac{1}{d}\log_2\frac{p}{d})nM + S(\log_2 d + \log_2 p) \qquad (7)$$

となる. 表 1 は配信アルゴリズムとそれぞれの通信コストをまとめたものである.

表 1 配信アルゴリズムと通信コスト

ring	$\frac{2(p-1)}{p}nM + S\{\log_2 p + (p-1)\}$
hybrid d	$(\frac{2(d-1)}{d} + \frac{1}{d}\log_2\frac{p}{d})nM + S(\log_2 d + \log_2 p)$

2.3 等価なアルゴリズム

hybrid d は, メッセージを d 分割しながら通信を行うアルゴリズムと 2.2.2 節で説明した. d の値を 1(メッセージの分割なし) と設定した hybrid 1 は, binomial-tree(以下, bino と呼ぶ) と等価なアルゴリズムとなっていた. また, d の値を p(集団通信に参加するノード数) と設定した hybrid p は, binomial-tree_scatter_doubling_allgather[5](以下, doubling と呼ぶ) と等価なアルゴリズムとなっていた. このことより, hybrd d は, bino と doubling を包括するアルゴリズムであると言える. よって, 以下では ring と hybrid d についてのみ議論を行う.

3. 環境依存パラメータを考慮した通信コストのモデリング

3.1 MPI 実装やシステム特性を考慮したモデル改良

本研究では, 配信アルゴリズム毎の特徴を数式化した通信コストを計算し, 最適なアルゴリズムの推定, 切り替えを検討している. 表 1 で記されているアルゴリズム毎の通信コストのメッセージサイズ n に注目するとそれぞれの式は n の一次方程式となる. これにより, メッセージサイズに依存する項としない項が存在し, メッセージサイズの大小によりアルゴリズムの良し悪しが変化するため, メッセージサイズによる最適であるアルゴリズムは変動する (例：図 8). また, 異るシステム間では, 通信速度に関するパラメータ M と通信遅延に関するパラメータ S の比率が異るため, 最適なアルゴリズムが切り替わるタイミングも異る. このような通信コストに関する理論上のオーダの議論や配信アルゴリズムの特徴の把握程度は可能であった. しかしながら, 様々な実システムに実装し実用的な最適化を行うという観点では, hybrid d におけるパラメータ d の決定を含む複数の配信アルゴリズムからのアルゴリズム選択が必要であり, 各アルゴリズムの実装上の制約や実機の特徴を考慮したさらに詳細な通信コストのモデル式構築がもとめられる.

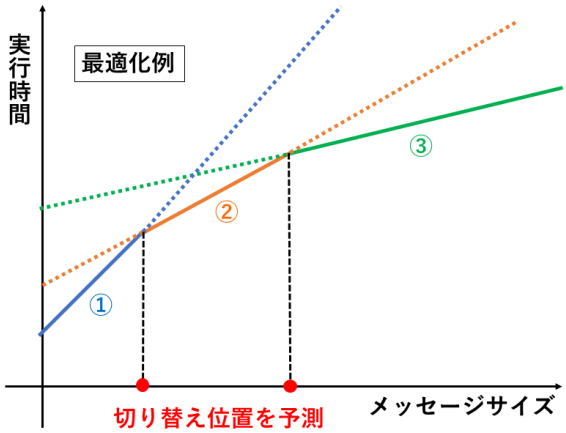

図 8 配信アルゴリズムの最適化例

まず最初に, 動的マルチキャストの実装においては, グループ内ノードへのデータ配信の前段階としてマルチキャストグループ構築のための通信が必要である. この際の通信データ量はノード数程度のビットマップデータであるため, 通信コストとしては 1 回分のノード間遅延の増加としてモデル式に導入した.

次に, MPI 環境でアルゴリズムを実装する際の制約等について考える. まず, ring では collect 部の全ての stage で, 全てのノードにおいて同時シフト型の通信 (node i から node (i+1)%p への配信が全ての i で同時実行) が行われる. 理論的なモデル式ではこれを 1 ステップと考えるが, MPI での実装や NIC での競合を考慮すると 2 ステップと考慮する必要がある (例えば、偶数番ノード群 E と奇数番

ノード群 O に分割し、最初のステップでは E が送信 O が受信等). 一方, hybrid d では collect 部で 2 台のノードペア間で同一サイズのデータ交換を行うが, 理論上のモデル式では双方向通信が完全に成功し NIC の帯域が理論上 2 倍になることを想定している. しかしながら, 実機においては必ずしも 2 倍にはならない. そこで, 通信速度に関するパラメータ M に関して, 相互通信する上で 1byte のデータ送信に要する時間単方向通信を行うときは M_1(相互通信する上で 1byte のデータ送信に要する時間 [sec]), 双方向通信を行うときは M_2(相互通信する上で 1byte ずつのデータ送受信 (合計, 2byte) に要する時間 [sec]) の 2 つのパラメータを導入する.

なお, 後述するが M_1 と M_2 の比もメッセージサイズに依存して変化する. さらに, メッセージサイズに依存しないノード間遅延 S(0[byte] メッセージ通信時の実行時間としてパラメータ化) についても, 同様に単方向で通信を行うときは S_1 を双方向で通信を行うときは S_2 としそれぞれの値を用いることとした. これらを考慮して改良した通信コストのモデル式を表 2 に示す.

表 2 MPI 実装やシステム特性を考慮した通信コスト

ring	$\frac{3(p-1)}{p} nM_1 + S_1\{2\log_2 p + 2(p-1)\}$
hybrid d	$\{(\frac{d-1}{d} + \frac{1}{d}\log_2 \frac{p}{d})M_1 + \frac{d-1}{d}M_2\}n$ $+2S_1 \log_2 p + S_2 \log_2 d$

3.2 通信速度のメッセージサイズ依存性を考慮した改良

ring と hybrid d(d>2) は, stage 毎に通信するメッセージサイズが変化するが (2.2 節参照), これまでのモデルでは通信速度がメッセージサイズによって大きく変化する実システムの性質を組み込んでいない. そこで, 本節では通信速度のメッセージサイズ依存性を考慮したモデル式の改良を行う. この際, 本来ならば各ステージで通信する全てのメッセージサイズについて考慮すべきであるが, コストモデルにおいてメッセージサイズに依存する項では, 大きなメッセージを通信するステージの影響が支配的と考え, サイズ n のメッセージを配信する際には, サイズ n, n/2 および n/4 に対して, M, M', および M" の 3 段階までのパラメータ (M_1, M_2 についても同様) を導入し n/4 以下のサイズのメッセージには M" を適用することでモデル式の簡略化を図った (表 3).

4. 評価実験

4.1 実験環境の概要

4.1.1 Cray XC40 の基本性能の調査

本研究では北陸先端科学技術大学院大学の並列計算機 Cray XC40[6] を用いて評価実験を行った (表 4). なお, 本研究は全ての実験を一般ユーザの権限でおこなっており, 利用可能なノード数の最大値は 256, 物理ノードへの明示的なランク割当は行っていない. また, MPI ベースのノード間通信性能に注目するため 1 ノードには 1 ランクのみを割当て実験を行った.

まず最初に, 実システムでの実行環境に依存するパラメータ M(相互通信する上で 1byte のデータ送信に要する時間 [sec]) と S(ノード間の遅延 [sec]) に関して計測を行った. Cray XC40 の 2 ノードにそれぞれ 1 プロセスずつ設定し, ping-pong プログラムの実行結果から M と S を導出した. 単方向通信時の性能としては, MPI_Send および MPI_Recv の単方向通信関数を用いて 2 ノード間で往復通信を行いその 1/2 の値として M_1 ならびに S_1 を導出した. 一方、双方向通信時の性能計測では, 2 ノード間でバリア同期を取った直後に MPI_Sendrecv 関数を用いてメッセージの送受信が行い, MPI_Sendrecv 関数の実行時間から M_2 ならびに S_2 を導出した. なお, 理論上は $M_2 = M_1$ となる. 送受信を行うデータは double 型の配列とし, 0[Byte] から 1[GByte] まで、2 の冪乗となるサイズで計測を行った. 各実行毎に 500 回の計測を行い, 上位下位それぞれ 10%を除いたトリム平均を行った実行時間からパラメータ M, S を導出した.

メッセージサイズ 0[byte] の通信にかかる時間を, MPI の runtime オーバーヘッドを含むノード間遅延時間とみなすと, S_1 は 1.5×10^{-6}[sec], S_2 は 2.0×10^{-6}[sec] となる. また, 本研究で用いるパラメータ M は, 単位が [s/byte] であるため, その逆数に比例する値として単方向通信時ならびに双方通信時の通信速度をグラフ化したものを図 9 に示す. 1KB から 1MB の領域で通信速度が大きく変化していることが確認できる. また, 双方向通信時の性能が単方向通信時の 2 倍には至っていないことと, その比率がメッセージサイズによって若干変動していることも確認できた.

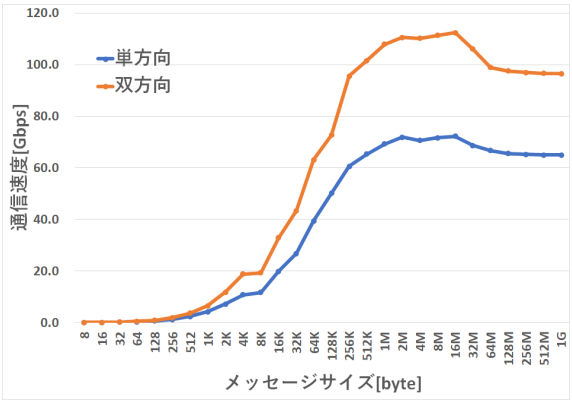

図 9 メッセージサイズ毎の通信速度

4.2 配信アルゴリズムの評価

実装したアルゴリズムのうち, ring, hybrid 1, hybrid 4, hybrid 8, hybrid p(集団通信に参加するノード数) の 5 つについて動的マルチキャストに要する時間を計測した. Cray XC40 の 64 ノードを用いて評価実験を行った. 起動時のノード数は 4, 8, 16, 32, 64 ノードを用意し, 各ノードに 1

表 3　メッセージサイズに対する通信速度の変化を考慮した通信コスト

ring	$\frac{1}{2}nM_1' + \frac{5p-6}{2p}nM_1'' + S_1\{2\log_2 p + 2(p-1)\}$
hybrid 1	$\{(\frac{d-1}{d} + \frac{1}{d}\log_2\frac{p}{d})M_1 + \frac{d-1}{d}M_2\}n + 2S_1\log_2 p + S_2\log_2 d$
hybrid 2	$\{(\frac{d-1}{d} + \frac{1}{d}\log_2\frac{p}{d})M_1' + \frac{d-1}{d}M_2'\}n + 2S_1\log_2 p + S_2\log_2 d$
hybrid d (d>2)	$\{\frac{1}{2}M_1' + \frac{1}{2}M_2' + (\frac{d-2}{2d} + \frac{1}{d}\log_2\frac{p}{d})M_1'' + \frac{d-2}{2d}M_2''\}n + 2S_1\log_2 p + S_2\log_2 d$

表 4　実験環境

計算ノード	548node/19728CPU コア
CPU	Intel Xeon E5-2695v4
	2.1GHz 18Core*2 ソケット
メモリ	128GB
総理論演算性能	662.8TFLOPS
ネットワーク	CRAY Dragonfly Topology[7]
コンパイラ	Intel(R) C++
MPI	cray-mpich

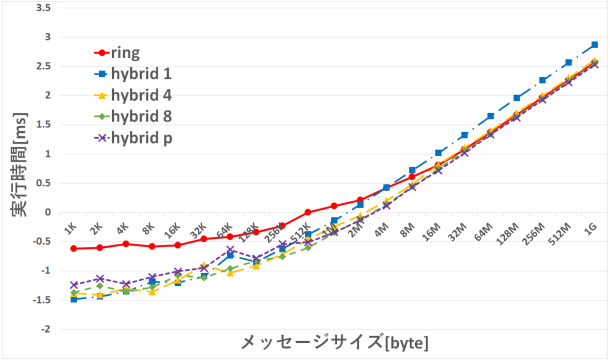

図 10　動的マルチキャスト時間（配信先ノード数 64）

ランクを割り当てワーカーノードを起動し, 起動したノードすべてにメッセージの送信を行う. マルチキャスト対象となるノード数が最大となる条件での実験に相当する. それぞれのノード数に対して 1[Kbyte]〜1[Gbyte] まで 2 倍間隔でメッセージサイズを変化させて計測を行った.

計測方法としては, 始めにメッセージを所有するノードがあるノードに送信を行う前に計測をスタートし, 全ノードがすべてのメッセージを受け取り終わるまでの時間を計測する. 各実行毎に 500 回の計測を行い, 上位下位それぞれ 10%を除いたトリム平均を行った値を実行時間とした. 図 10 は, ノード数 p=64 の場合のメッセージサイズに対するマルチキャスト時間の計測結果 (両対数グラフ) である.

メッセージサイズとともに最適な配信アルゴリズムが変化することが確認できる. 具体的には, 今回の実験環境では, メッセージサイズの増加とともに最適配信アルゴリズムが hybrid 1 から hybrid 4, hybrid 8, hybrid p と変化する傾向を確認した. また, Cray XC40 の通信速度が高速なため, ring が最速アルゴリズムとなることはなかった. 変化のタイミングは異るが, 他のノード数に対する実験でも同様の傾向が確認できている. また, 実装したアルゴリズムに先行研究 [2] のアルゴリズムも包含されるため, 当然ながらベストケースの性能も改善できた. なお, 最速である配信アルゴリズムが切り替わる位置 (メッセージサイズ) の近傍では, 最速アルゴリズムの局所的な逆転現象が見られるが、それぞれの実行時間の差は僅かな差であり, 計測のゆらぎの影響も加わってこのような逆転が発生したものと考える.

4.3　通信コスト予測モデル式による最適配信アルゴリズムの自動推定の可能性検討

4.3.1　通信コストを用いた予測モデル式の計算方法

実行環境下での最適な配信アルゴリズムを確認するには, 実際に配信アルゴリズムを実装したプログラムを実行し検証しなければ分からない. そこで, 通信コストをもとに実行環境下での最適なアルゴリズムを予測し, 実行時のアルゴリズム最適化の検討を行った.

本実験では 3 つのモデル式を用いて評価を行う (表 5). 予測モデル式 1 と 2 は双方向通信による速度向上を考慮したモデル (表 2 に対応) であるが, モデル式 1 は機器のカタログ性能 (ハードウェアレベルの最速値 [8]) を用いたものであるのに対し, モデル式 2 では 4.1.1 節で求めた通信パラメータ M,S の実測値 (ソフトウェアのラインタイムオーバヘッドも含む) を利用している. 最大メッセージサイズに対応する通信パラメータの実測値は利用するが, アルゴリズムの途中でのメッセージサイズ変化による通信パラメータ変化は考慮しないモデルである. 予測モデル 3 は 3.2 節で提案したモデル式 (表 3) に対応したモデルである.

表 5　予測モデル式の計算方法

	通信コスト	パラメータ M, S の値
予測モデル式 1	表 2	文献 [8] の値を使用
予測モデル式 2	表 2	計測した値を使用
予測モデル式 3	表 3	計測した値を使用

4.3.2　実験結果と考察

実験の方法については 4.2 節と同様の方法で行い, 今回の実験では集団通信に参加するノード数 p の値として 128 と 256 を追加し, hybrid d の d の値を $2^m (0 \leqq m \leqq 8)$ で変化させて, それぞれのメッセージサイズ n と配信ノード数 p に対して最速となる d の値を調査した. 2020 年 3 月上旬と 5 月上旬に測定した結果を表 6 と表 7 に示す. いずれも n×p が大きくなると d も大きくなる傾向は共通であるが, 最適な d が切り替わるタイミングは若干異っている. 前述のとおり, 計測ゆらぎの影響とともに, 今回は物理ノー

ドとの対応関係が異っている影響も含まれていると考える.

これらの実測値に対して, 表 8, 9, 10 は予測モデル式 1, 2, 3 にパラメータを代入して計算した数値である. 今回の 2 回分の計測に対して, モデル式 1 を適用したときに対してモデル式 3 を適用したときの改善率をそれぞれ表 12, 13 に示す. 実行時間の予測値としては, いずれも計測値の傾向に近付いていており, モデル式 1 を適用したときよりもモデル式 3 を適用した方が速くなっていることが多いと表 12, 13 を見ることで確認できる. 表 12 と表 13 を比べたときに表 12 での, 8Kbyte 近傍での著しい性能の低下がみられる. この現象は, Cray XC40[6] の利用者が 3 月上旬では 5 月上旬に比べてると多数であったため, 通信経路での輻輳等による通信時間の増加し, さらに計測のゆらぎなどが大きく影響してしまったのではないかと考える. 表 13 では, この現象の影響があまりみられず, 多数の箇所で改善されていることが確認できため, モデル式 3 はモデル式 1 に比べると有効であるといえる.

表 8 と 9 では, メッセージサイズが小さい領域で切替タイミングが実測結果に近付く傾向が見られた. これは, モデル式 1 において想定している通信速度と実際の通信速度がこの領域では大幅に異るためであると考える. 次に, 表 9 と 10 を見比べると, メッセージサイズが 8KB から 512KB の範囲で d の切替タイミングが変化し実測値に近付くが, この領域は実システムにおいて通信速度がメッセージサイズの増加とともにほぼ線形に変化する領域に対応しており, 通信速度の変化を考慮した効果が確認できる.

予測モデル式 3 までである程度実機パラメータを反映した予測制度の向上が確認できた. ここまでのモデルは実機で計測した通信パラメータを機械的に適用した予測モデルであったが, 一つの試行として通信パラメータに人為的な補正 (具体的には, 予測モデル式 3 に代入したパラメータ M を 1.3 倍, S を 2 倍に補正) を行ったモデル式 4 を当て嵌めたところ (表 11) 更に実測値に近付くことが確認できた. この補正は, モデル式 3 に導入できていない通信経路での輻輳等による通信時間の増加, ノード間の経路長の違いによる遅延時間のバラツキ, 双方向通信の開始時間のズレなどの動的要因に対応するものと考えることも可能である.

5. おわりに

複数のアルゴリズムを包括する hybrid d を用いた動的マルチキャストを実装し, 動的マルチキャストにおける hybrid d の有用性が確認できた.

hybrid d の切り替えのタイミングについては, ping-pong プログラムを用いて通信速度に関する実機パラメータを測定し, 予測モデル式 3 に適用することで, 最適な d をほぼ推定可能であることがわかった. 動的要因を考慮した更なるモデル式の改良も可能であるが, 実際の実行時間に与える影響を考慮すると今回の実験環境では予測モデル式 3 程度で十分であると考える. なお, 今回の実験は, 通信速度が速く S の影響が表れにくい Cray XC40[6] での実験のため, 他の実験環境での予測モデル式の妥当性を今後検証する必要がある.

謝辞 本研究は, 北陸先端科学技術大学院大学情報社会基盤センターのシステムを利用して実施しました. 感謝いたします. 本研究の一部は、JSPS 科研費 (20K11744) の助成を受けて実施した.

参考文献

[1] MPICH ー High-Performance Portable MPI(https://www.mpich.org/)
[2] 長嶺祐輔: ” 投機的手法を用いたインタラクティブシミュレーションシステムの多重投機拡張と動的マルチキャストの応用 ”, 福井大学大学院工学研究科修士課程情報・メディア工学専攻修士論文.2017
[3] 中垣達哉:”ネットワークの物理トポロジーを考慮したマルチキャスト通信の高速化”, 福井大学工学部情報・メディア工学科情報・メディア工学専攻卒業論文.2017
[4] Mike Barnett, Satya Gupta, David G. Payne, LanceShuler, Robert van de Geijn, Jerrel Watts: ” Buildinga High-Performance Collective Communication Li-brary ”, Proceedings of the 1994 ACM/IEEE conference on Supercomputing Pages 107-116
[5] 松本幸, 安達知也, 田中稔, 住元信司, 曽我武史, 南里豪志, 宇野篤也, 黒川原佳, 庄司文由, 横川三津夫: ” MPI Allreduce の「京」での実装と評価 ”, 情報処理学会論文誌 コンピューティングシステム Vol.5 No.5 152162 (Oct. 2012)
[6] JAIST 情報社会基盤センター. 入手先 ⟨http://isc-w3.jaist.ac.jp/iscenter/⟩ (参照 2016-02-04).
[7] John Kim, William J.Dally, Steve Scott, Dennis Abts: Technology-Driven, Highly-Scalable Dragonfly Topology, ieee computer society, DOI 10.1109/ISCA.2008.19.
[8] Bob Alverson: ” Cray XC Series Network ”, http://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf

表 6 計測結果 (3 月上旬)

	4	8	16	32	64	128	256
1K	1	1	1	1	1	1	1
2K	1	1	1	1	1	2	2
4K	1	1	2	2	1	2	2
8K	2	2	2	2	2	2	4
16K	1	1	4	4	2	2	8
32K	1	1	1	2	2	2	2
64K	2	1	2	2	2	2	2
128K	2	2	2	2	2	2	32
256K	2	2	2	8	8	8	32
512K	2	4	8	16	16	16	16
1M	2	4	8	16	32	16	8
2M	2	4	16	8	32	32	8
4M	4	4	16	16	32	32	8
8M	4	4	8	16	32	32	16
16M	4	4	16	16	32	16	32
32M	4	4	16	32	64	32	128
64M	4	8	16	16	32	64	128
128M	4	8	16	16	16	64	128
256M	4	8	16	16	32	64	128
512M	4	8	16	16	64	64	128
1G	4	8	16	16	16	64	128

表 7 計測結果 (5 月上旬)

	4	8	16	32	64	128	256
1K	1	1	2	1	1	1	1
2K	1	1	1	1	1	1	1
4K	1	1	2	2	4	4	2
8K	2	2	2	2	4	2	4
16K	1	1	4	8	4	4	8
32K	1	1	1	2	8	8	8
64K	1	1	2	4	4	4	16
128K	1	2	2	4	4	8	64
256K	1	4	4	4	8	8	64
512K	2	4	4	8	16	16	16
1M	2	4	8	8	16	16	64
2M	2	4	8	8	16	32	64
4M	2	4	8	8	16	64	64
8M	4	4	8	16	16	64	32
16M	4	4	8	8	32	32	64
32M	4	4	8	16	64	64	32
64M	2	4	8	16	64	128	128
128M	2	4	8	16	64	64	64
256M	2	4	8	16	64	128	64
512M	2	4	8	16	64	128	64
1G	2	4	8	16	64	128	128

表 8 予測モデル式 1 による切り替え

	4	8	16	32	64	128	256
1K	1	1	1	1	1	1	1
2K	1	1	1	1	1	1	1
4K	1	1	1	1	1	2	2
8K	1	1	2	2	2	2	4
16K	1	2	2	4	4	4	4
32K	2	2	4	4	8	8	8
64K	2	4	4	8	8	16	16
128K	2	4	8	8	16	16	32
256K	2	4	8	16	16	32	32
512K	2	4	8	16	32	32	64
1M	2	4	8	16	32	64	64
2M	2	4	8	16	32	64	128
4M	2	4	8	16	32	64	128
8M	2	4	8	16	32	64	128
16M	2	4	8	16	32	64	128
32M	2	4	8	16	32	64	128
64M	2	4	8	16	32	64	128
128M	2	4	8	16	32	64	128
256M	2	4	8	16	32	64	128
512M	2	4	8	16	32	64	128
1G	2	4	8	16	32	64	128

表 9 予測モデル式 2 による切り替え

	4	8	16	32	64	128	256
1K	1	1	1	1	1	1	1
2K	1	1	1	1	1	1	2
4K	1	1	2	2	2	2	4
8K	1	2	2	4	4	4	8
16K	1	2	2	4	4	8	8
32K	2	2	4	4	8	8	8
64K	2	4	4	8	8	8	16
128K	2	4	4	8	8	16	16
256K	2	4	8	8	16	16	32
512K	2	4	8	16	32	32	32
1M	2	4	8	16	32	32	64
2M	2	4	8	16	32	64	64
4M	2	4	8	16	32	64	128
8M	2	4	8	16	32	64	128
16M	2	4	8	16	32	64	128
32M	2	4	8	16	32	64	128
64M	2	4	8	16	32	64	128
128M	2	4	8	16	32	64	128
256M	2	4	8	16	32	64	128
512M	2	4	8	16	32	64	128
1G	2	4	8	16	32	64	128

表 10 予測モデル式 3 による切り替え

	4	8	16	32	64	128	256
1K	1	1	1	1	1	1	1
2K	1	1	1	1	1	1	1
4K	1	1	2	2	2	2	2
8K	2	2	2	4	4	4	4
16K	1	1	4	4	8	8	8
32K	1	2	2	2	8	16	16
64K	1	1	4	8	16	16	16
128K	1	2	2	8	16	16	32
256K	1	2	8	8	16	32	32
512K	2	2	8	16	16	32	32
1M	2	4	8	16	32	32	64
2M	2	4	8	16	32	64	64
4M	2	4	8	16	32	64	128
8M	2	4	8	16	32	64	128
16M	2	4	8	16	32	64	128
32M	2	4	8	16	32	64	128
64M	2	4	8	16	32	64	128
128M	2	4	8	16	32	64	128
256M	2	4	8	16	32	64	128
512M	2	4	8	16	32	64	128
1G	2	4	8	16	32	64	128

表 11 予測モデル式 4 による切り替え

	4	8	16	32	64	128	256
1K	1	1	1	1	1	1	1
2K	1	1	1	1	1	1	1
4K	1	1	1	1	2	2	2
8K	2	2	2	2	2	4	4
16K	1	1	1	4	4	4	8
32K	1	2	2	2	2	8	16
64K	1	1	4	8	16	16	16
128K	1	2	2	8	16	16	16
256K	1	2	8	8	16	16	32
512K	2	2	8	8	16	32	32
1M	2	4	8	16	16	32	64
2M	2	4	8	16	32	32	64
4M	2	4	8	16	32	64	64
8M	2	4	8	16	32	64	128
16M	2	4	8	16	32	64	128
32M	2	4	8	16	32	64	128
64M	2	4	8	16	32	64	128
128M	2	4	8	16	32	64	128
256M	2	4	8	16	32	64	128
512M	2	4	8	16	32	64	128
1G	2	4	8	16	32	64	128

表 12 モデル 1 に対してモデル 3 を適用したときの改善率 (3 月上旬)

	4	8	16	32	64	128	256
1K	0	0	0	0	0	0	0
2K	0	0	0	0	0	0	0
4K	0	0	4.0	0.4	-2.6	0	0
8K	10.5	18.2	0	-3.0	-41.5	-30.7	0
16K	0	5.8	6.8	0	3.6	9.6	25.9
32K	10.3	0	11.8	34.9	0	-17.6	24.1
64K	7.6	19.0	0	0	5.02	0	0
128K	-5.7	14.3	14.5	0	0	0	0
256K	-0.6	2.9	0	6.5	0	0	0
512K	0	-3.5	0	0	5.0	0	2.2
1M	0	0	0	0	0	2.7	0
2M	0	0	0	0	0	0	-0.9
4M	0	0	0	0	0	0	0
8M	0	0	0	0	0	0	0
16M	0	0	0	0	0	0	0
32M	0	0	0	0	0	0	0
64M	0	0	0	0	0	0	0
128M	0	0	0	0	0	0	0
256M	0	0	0	0	0	0	0
512M	0	0	0	0	0	0	0
1G	0	0	0	0	0	0	0

表 13 モデル 1 に対してモデル 3 を適用したときの改善率 (5 月上旬)

	4	8	16	32	64	128	256
1K	0	0	0	0	0	0	0
2K	0	0	0	0	0	0	0
4K	0	0	0.3	9.5	12.2	0	0
8K	11.6	12.0	0	-1.1	-10.2	0	0
16K	0	14.4	7.9	0	-10.9	-1.0	1.0
32K	13.0	0	7.4	4.7	0	-3.3	-8.6
64K	11.3	7.0	0	0	2.5	0	0
128K	1.2	2.6	4.1	0	0	0	0
256K	-2.7	-1.8	0	1.9	0	0	0
512K	0	-4.7	0	0	1.8	0	-2.2
1M	0	0	0	0	0	8.5	0
2M	0	0	0	0	0	0	4.6
4M	0	0	0	0	0	0	0
8M	0	0	0	0	0	0	0
16M	0	0	0	0	0	0	0
32M	0	0	0	0	0	0	0
64M	0	0	0	0	0	0	0
128M	0	0	0	0	0	0	0
256M	0	0	0	0	0	0	0
512M	0	0	0	0	0	0	0
1G	0	0	0	0	0	0	0