大規模計算環境のためのランク配置最適化手法 RMATT

今出 広明† 平本 新哉† 三浦 健一† 住元 真司†

数万ノードを超える HPC 向け計算機では，Torus や Mesh などの直接網が多く採用されている．通信先ごとにより経由するルータ数（ホップ数）が異なる直接網においては，トポロジに合わないアプリケーションを実行する場合，ランクを実行させるノード番号の配置(ランク配置)を最適化する必要がある．この計算オーダは $N!$ となることから，大規模なアプリケーションに対してユーザがランク配置最適化を行なうことは簡単ではない．本論文では，ランク配置最適化を自動的に行なうチューニングツール RMATT について述べる．RMATT は対象の通信内容を用いて通信ホップ数とノード間の転送量の全体最適を行なうことに特徴があるが，計算量の削減が大きな課題となる．この問題を解決するために，分割手法とメタヒューリスティクスを用いた手法を並列に処理する方式を開発した．本方式を実装し，ネットワークシミュレータ OpenNSIM 上で 4096 ランクの Allgather アルゴリズムの通信モデルの処理時間を評価したところ，通信処理時間の約 75%の削減を確認した．

RMATT: Rank Map Automatic Tuning Tool for Large-Scale Computer System

HIROAKI IMADE †, SHINYA HIRAMOTO †, KENICHI MIURA † and SHINJI SUMIMOTO †

High performance computing systems with over ten thousands of computing node use direct network such as torus or mesh. On such direct networks, it is important to optimize communication pattern of application which is not fit to direct network topology by changing physical location map of each node. However this optimization is not easy to realize for large scale parallel applications because this computation order is $N!$. This paper proposes a communication tuning tool called RMATT which processes rank location optimization automatically. RMATT does total optimization of a communication location using number of communication hops and amount of communication, however reduction of computation time is issue to solve. To resolve the issue, we have developed a problem dividing method and optimization method using meta heuristic. We have implemented a prototype of RMATT and evaluated using network simulator OpenNSIM. The evaluation result shows that the result of Allgather model benchmark using RMATT reduces about 75% of elapse time of standard Allgather algorithm on 4096 nodes.

1. はじめに

高性能計算に対する要求はとどまることを知らず，Linpack 演算性能の世界ランクである Top500 [1]の 2010 年 11 月のリストでは，トップ7は1ペタフロップスを超える演算能力を持っている．計算能力に対する要求はこれからも続くことが予想され，2018 年にはエクサフロップスクラスの計算機システムが登場するといわれている．

こうした計算性能に対する要求に従い，高性能計算システムを構成する計算ノードの数も増加の一途をたどっており，TUBAME2[2]，T2K[3]，RICC[4]に代表される間接網のシステムに加え，BlueGene/P[5]，Jaguar[6]や京システム[7]に代表される Torus・Mesh といった直接網のシステムも多く利用されるようになった．計算ノードの数が増えるにつれ，間接網のシステムは構築が難しくなってきており，今後，直接網のシステムが増えてくると予想される．

計算ノードが増えるに従い，並列アプリケーション内の通信の最適化も，より難しくなってきている．

間接網を持つシステムは計算ノード間の接続が比較的均等であるため主にそれぞれのノード間の通信に着目して通信の最適化を実施することが可能である．

しかし，直接網のシステムでは，隣接通信主体のアプリケーションを除き，計算ノード間の通信が複数の計算ノードを経由する際に，通信経路が重なり通信時間が増加する場合がある．このため，通信時間の最適化時はそれぞれのノード間の通信に加え，その時々に同時に起こり得るノード間の通信経路の重なりを考慮する必要があり，その最適化は容易ではない．さらには，ノード番号とその直接網の中の物理的な位置は全体のノード数と物理的な形状に応じて変化するため，形状に応じてプログラムを変更するのは現実的ではない．多種多様の通信パターンが交錯する環境下で，アプリケーションプログラムを変更することなくノード

† 富士通株式会社
Fujitsu Limited

間通信の重なりを減らすことが重要である.

本論文では, 計算ノード間の通信の多さに応じて計算ノードの位置(ランク配置)を変更することにより通信時間を最適化するランク配置チューニングツール RMATT (Rank Map Automatic Tuning Tool)について述べる. RMATT はアプリケーションプログラムを変更することなく, 通信ホップ数とノード間の転送量の全体最適化を, アプリケーションの通信内容(通信パターン)を用いて行なうことに特徴がある. しかし, 全体最適化の計算オーダが $N!$となるため計算量の削減が大きな課題となる. この問題を解決するため分割手法とメタヒューリスティクスを用いた手法を並列処理する方式を開発した. 本方式を実装し, ネットワークシミュレータ OpenNSIM [8]上で 4096 ランクの Allgather アルゴリズム [9] の通信処理時間を評価した. 評価の結果, ランク配置最適化により, 通信時間を約 75%削減できることを確認した.

2 章ではランク配置最適化による通信の最適化について述べる. 3 章では RMATT における要件について述べ, 4 章では要件に基づき RMATT の設計を行なう. 5 章では設計に基づいた RMATT の実装について述べる. 6 章では RMATT によるランクマップ最適化の流れについて述べる. 7 章では RMATT の性能を評価し, 8 章では関連研究と比較し, 9 章では結論と今後の予定を述べる.

2. ランク配置最適化による通信の最適化と RMATT

Mesh や Torus などの直接網で MPI アプリケーションを実行する場合, ランクをどのノードで実行するか(ランク配置)により通信処理時間は異なる. これは, ランクの配置によりランク間の通信時のホップ数や輻輳の発生頻度が異なるためである. このため, 直接網においては通信処理内容に応じて各ランクの配置を適切に入れ替えることで, 通信処理時間を最小化することができる. 本論文ではこれを, ランク配置最適化と呼ぶ.

ランク配置を変更するためには, アプリケーションのプログラムを変更するか, ランクの配置を任意に指定できる仕組みを用意する必要がある. Open MPI などの MPI ライブラリでは, ランク配置をファイルにより指定することができる. 本論文では, このランク配置を指定するファイルをランクマップファイルと呼び, ランクマップファイルを用いたランク配置最適化を想定している.

ランクマップファイルを用いたランク配置最適化では, ランク配置の決定と評価を試行錯誤的に行なう必要がある. N ノードの計算機におけるランク配置の総組み合わせ数は $N!$個となることから, 1 万ランクの規模の場合, 総組み合わせ数は $10^{35,659}$ 個のランク配置が考えられる. 開発者の経験や知識により必ずしもすべてのランク配置を評価する必要は無いが, ランク配置の決定と評価をアプリケーション開発者が手作業で行なう場合面倒な作業となる. 今後ペタフロップスクラスの大規模な直接網を持つ計算システムが増加することが予想されることから, 開発者への負担をいかに軽減するのかが重要となると考えられる.

RMATT はこれまで述べたランク配置最適化を自動的に行なうことを目的としたツールである. アプリケーション開発者は夜間や週末, 休日などの余暇時間や他作業を行なう間に余剰CPUを用いてRMATTによるランク配置最適化を行ない, 開発者が通信最適化にかける時間を有効活用できるとともに最適化によるアプリケーション実行時間の短縮により計算資源を有効活用できることを期待している.

3. RMATT の要件と課題へのアプローチ

3.1. 要件

本論文で提案する RMATT の要件を以下に挙げる.

要件 1. 大規模な計算システム, アプリケーションへの対応
ペタフロップスクラスの数千から数万ノードの計算システムと, その上で動作する数千から数万ランクのアプリケーションに対応する必要がある.

要件 2. 多様なネットワーク, 通信パターンへの対応
ペタフロップスクラスの計算システムで今後採用が増えることが予想される, Mesh・Torus といった直接網に対応する必要がある. 直接網においてランク配置最適化が必要な, 多様な通信パターンに対応する必要がある.

3.2. 課題と解決へのアプローチ

本節では各要件における課題とその解決へのアプローチについて述べる.

要件 1 の課題

N 個のランクをネットワーク上に配置する場合, その総組み合わせ数は $N!$個となる. この最適化は非常に大規模な探索空間であり, 全探索により 1 個の最適なランク配置を求めるためには膨大な時間を要する. このため, 探索空間の圧縮と, 最適解探索の高速化が必要となる.

要件 2 の課題

多様な計算システムやアプリケーションに対応するためには, 問題に依存しない最適化手法が必要となる. また, 同じような通信処理時間となるランク配置が多く存在することが予想されることから, 探索空間には無数の局所解が存在すると考えられる. このことから, 局所解回避に強い最適化手法が必要となる.

解決へのアプローチ

これらの問題に対応するために, 以下の点を考慮したランク配置最適化を設計する.

要件1へのアプローチ

・探索空間の圧縮
ランクを複数のグループに分割し, グループ単位でネットワーク上に配置することで総組み合わせ数を減少する. 例として, 1000 ランクのランク配置最適化において 10 ランクずつ 100 グループに分割し, グループ単位で配置する場合, その組み合わせ数は $100\times10!=3.6\times10^{8}$ 個となる. 1000 ランクの総組み合わせ数は 4×10^{2567} 個となるので, 探索空間を大きく圧縮することができる.

・高速化
ランク配置最適化を並列分散処理することで処理時間

を短縮する.

要件 2 へのアプローチ

・最適化手法

問題に依存せず多数の局所解に対応する最適化手法として, 局所解回避に強いメタヒューリスティクスである焼きなまし法(Simulated Annealing: SA)[10]を用いる.

4. RMATT の設計

4.1 設計方針

3.2 章に基づき, 設計の方針を以下に示す.

探索空間の圧縮

ランクを, よく通信を行なうもの同士で複数のグループに分割し, グループごとにネットワーク上に配置することにより総組み合わせ数が減少することを期待する. この分割処理を BISEM(BISEction rank Map)と定義する.

高速化手法

最適化に要する時間を短縮するために, 最適化処理を並列分散化する.

最適化手法

問題に依存せず, 局所解回避に強い最適化手法として焼きなまし法(SA)を用いる. RMATT におけるランク配置最適化を OPTIM(OPTImization rank Map)と定義する.

4.2 全体の処理の流れ

設計方針に基づき, RMATT では, BISEM により探索すべき総組み合わせ数を減らした上で, OPTIM によりランク配置最適化を行なう. RMATT はまず, ネットワークのトポロジやノード数, 通信パターンを元に BISEM を実行する. 次に, BISEM の結果を元に OPTIM のための初期ランク配置を作成する. 最後に BISEM から得られた初期ランク配置を元に, OPTIM を実行しランク配置最適化を行なう. RMATT の処理時間を短縮するために, OPTIM は並列実行される.

4.3 BISEM の設計

BISEM では, ランクをよく通信を行なうもの同士で複数のグループ(ランクグループ)に分割し, グループ単位でネットワーク上に配置することで探索空間の圧縮を行なう. ランクグループの分割では, 以下の点を考慮する.

- ランクグループ内のランクはネットワーク上に固めて配置する
- ランクグループ間で通信がある場合, それらのランクグループはネットワーク上に隣同士に配置する

RMATT ではネットワークを複数の領域に分割し, 各領域間の隣接関係に合わせてランクグループを分割することにより, 上記の点を考慮した分割を行なっている.

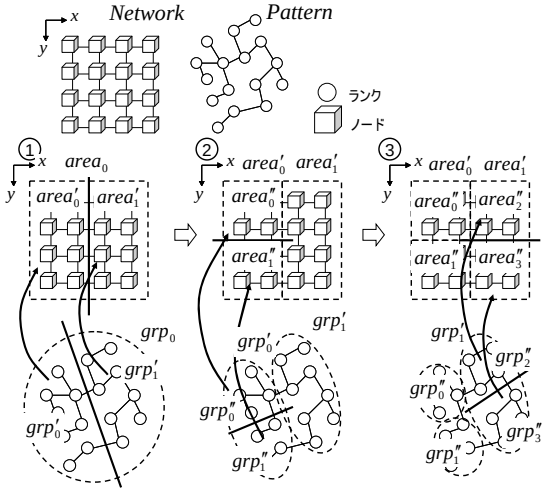

図 1 BISEM による分割処理の流れ

BISEM によるネットワークとランクの分割の流れを図 1 に示す. 図 1 は *Pattern* で示された 16 個のランクを *Network* で示された 4x4 ノードの 2D Mesh に配置するランク配置最適化における BISEM の処理の様子である. *grp* はランクグループを, *area* はネットワーク領域を示す.

ステップ 1

$area_0$(=*Network*)を $area'_0$ と $area'_1$ の 2 つの領域に x 軸上で分割する. 次に grp_0(=*Pattern*)を以下の条件を満たした上で grp'_0 と grp'_1 間の通信量が最も少なくなるように分割する.

条件 1. grp'_0 と grp'_1 のランク数は同じ

条件 2. grp'_0 内のランク間の通信量と grp'_1 のランク間の通信量は同じ

grp'_0 を $area'_0$ に, grp'_1 を $area'_1$ に配置する.

ステップ 2

$area'_0$ について y 軸上で分割し, $area''_0$ と $area''_1$ とする. 次に grp'_0 を条件 1,2 に従い分割し, grp''_0 と grp''_1 とする. grp''_0 を $area''_0$ に, grp''_1 を $area''_1$ に配置する.

ステップ 3

$area'_1$ について y 軸上で分割し, $area''_2$ と $area''_3$ とする. 次に grp'_1 を分割し grp''_2 と grp''_3 とするが, このとき条件 1,2 に加え以下の条件を満たすようにする.

条件 3. (grp''_2 と grp''_0 間の通信量) > (grp''_2 と grp''_1 間の通信量)

条件 4. (grp''_3 と grp''_1 間の通信量) > (grp''_3 と grp''_0 間の通信量)

$area''_2$ と $area''_0$, $area''_3$ と $area''_1$ が互いに隣接しているため, grp''_2 は $area''_2$ に, grp''_3 は $area''_3$ に配置することでネットワーク領域の隣接関係とランクグループ間の通信関係は同様となる.

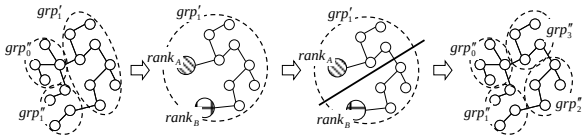

図 2 BISEM におけるランクグループの分割の流れ

ランクグループの分割では，ランクをグラフの頂点とし，ランク間の通信を辺とすることでグラフ分割問題と置き換えることができる．これにより METIS [11]などの一般的なグラフ分割ツールを利用することができる．ただしこれらのグラフ分割ツールは一般に条件 1,2 に対応したグラフ分割を行なうことができるが，条件3,4にも対応したグラフ分割は難しい．そこで図 2 のように，$grp"_0$と$grp"_1$をそれぞれ一つのランク $rank_A$, $rank_B$とし，grp'_1に含めることで，一般的なグラフ分割ツールを用いても，条件 3,4 を満たす分割を行なうことができる．RMATT では，図2においてgrp'_1の分割に用いる $grp"_0$, $grp"_1$を，grp'_1のサブグループと呼ぶ．*grp* のサブグループとは，*grp* が配置されるネットワーク領域に隣接する領域に配置されているランクグループを指す．

なお，グラフ分割ツールにより条件 1,2 を満たすようにランクグループの分割は行なわれるが，ランク数が奇数の場合など，すべてを満たす分割が行なわれるとは限らない．もし条件 1,2 を満たす分割が行なわれなかった場合を考慮し，OPTIM ではランクグループ間をランクが移動できるようにすることで分割結果を調整できるようにしている．

また，図 1 では Mesh を例に挙げたが，Torus の場合も BISEM のアルゴリズムは同様である．ただし，サブグループ生成時にはネットワークの各次元の端にある領域は逆端にある領域とも隣接していることに注意する．

4.4 OPTIM の設計

OPTIM で用いる SA では，問題に応じて新しい解(ランク配置)を生成するためのオペレータと，解を評価するための評価関数について設計する必要がある．

BISEM によりランクグループ単位での配置が決定することから，オペレータがランク配置を変更する範囲は狭くすることができる．RMATT では，ネットワーク上にランダムに一定の範囲を選択し，範囲内にあるランク配置をランダムに入れ替えるオペレータを用いる．スワップする範囲やランクの選択をランダムに行なうことで，ランク配置の変更をバランス良く行なうことができる．スワップする範囲は RMATT ユーザが任意に設定できるが，RMATT では BISEM により分割されたネットワーク領域よりやや広い範囲がスワップ範囲となることを想定している．これは BISEM におけるランクグループ分割時に，条件 1,2 を満たさないランクグループ分割が行なわれる可能性を考慮したためである．スワップ時にランクグループ間をランクが移動できるようにすることで，BISEM による分割結果を修正することができる．

直接網においては，通信処理時間はホップ数以外に輻輳が影響する．しかしながら輻輳の発生には通信のタイミングや外乱も影響するため定式化は難しい．輻輳は，特定のノード間のリンクを流れたバイト数が大きいほど発生しやすく，各リンクを流れるバイト数が一定であれば発生しにくいという特徴を持つ．そこで輻輳の危険性を示す指標として，$\max_{link}$を評価関数に用いる．あるリンクにおいて，リンク内を流れた総バイト数を *link* としたとき，すべてのリンクにおける *link* の最大値を $\max_{link}$とする．

SA において，オペレータによる解の生成と評価関数による評価処理は複数回行なう必要があるが，これらの処理は並列に行なうことができる．そこでこれらの処理を複数のノードに分散処理させる．解の生成と評価処理を分散処理するノードをワーカノードと呼び，ワーカノードの処理結果を受け取り，ランク配置を更新する処理を行なうノードをマスタノードと呼ぶ．マスタノードとワーカノードの処理の流れを以下に示す．

ステップ 1
マスタノードはワーカノードに新しい解(ランク配置)を生成する元となるランク配置を送信する．

ステップ 2
各ワーカノードはオペレータを適用し新しい解を生成し，評価関数を用いて評価する．

ステップ 3
マスタノードは評価結果を受け取る．

5. RMATT の実装

設計に基づき，RMATT プロトタイプの実装を行なった．プログラミングには Java を用い，OPTIM における並列処理はソケット通信を用いた．

5.1 RMATT の構成

RMATT は BISEM コンポーネントと OPTIM コンポーネントからなる．BISEM コンポーネントは，ネットワークのトポロジやノードサイズなどのネットワーク構成と，ランク間の通信内容を記した通信パターンを入力値とし，BISEM を実行する．通信パターンは VampirTrace [12]などのプロファイラが出力した通信ログファイルや，アプリケーションプログラムを解析することで得られる．なお，現在，自動的に通信ログファイルを解析し通信パターンを出力するツールを開発中である．BISEM の結果は OPTIM における SA のための初期ランク配置 map_0として出力される．OPTIM は map_0とネットワーク構成，通信パターンを入力値とし，ランク配置最適化を行なった結果を MPI で利用可能なランクマップファイルとして出力する．

5.2 BISEM の実装

BISEM のプログラムを図 3 に示す．

- $area_0$: ネットワーク領域を示し，ネットワーク領域に属するノードを格納する配列である．$area_0$には全ノードが格納される．
- grp_0: ランクグループを示し，ランクグループに属するランクを格納する配列である．grp_0には全ランクが格納される．
- *Area*, *Area'*: 複数のネットワーク領域を格納する配列である．
- *Grp*, *Grp'*: 複数のランクグループを格納する配列である．
- *MAX_ITRS*: 最大分割数であり，ユーザが指定することができる．*MAX_ITRS* により分割後のネットワーク領域のサイズが決定される．OPTIM におけるスワップ範囲はネットワーク領域の

サイズを元に決定されることから，*MAX_ITRS* の値に応じスワップ範囲は決定される．

- *axis*←decideAxis(*Area*)：ネットワーク領域を分割する軸を決める処理であり，*Area* に属するネットワーク領域において最もサイズが大きい軸が返される．
- *Area'* ←bisectionArea(*Area,axis*)：*axis* 軸でネットワーク領域を 2 分割する処理であり，*Area'* には *Area* に格納された各ネットワーク領域を分割した新しいネットワーク領域が格納される．
- *Grp'* ←bisectionGrp(*Grp,Area'*)：ネットワーク領域の分割結果 *Area'* に基づき *Grp* を分割する処理であり，*Grp'* には *Grp* に格納された各ランクグループを分割した新しいランクグループが格納される．ランクグループの分割には代表的なグラフ分割ツールである METIS [12]を用いる．
- map_0←makeInitMap(*Area,Grp*)はネットワーク領域とランクグループの分割結果から OPTIM に渡す初期ランク配置 map_0 を生成する処理である．各ネットワーク領域において，配置されたランクグループ内のランクを領域内のノードにランダムに配置し map_0 を生成する．RMATT では map_0 をランクの配列として実装し，配列の要素番号が，そのランクが配置されるノード番号としている．

```
area0 ← Network, grp0 ← Pattern;
Area ← {area0}, Grp ← {grp0};

for( itrs=0; itrs<MAX_ITRS; itrs++){
   axis ← decideAxis(Area);
   Area' ← bisectionArea(Area, axis );
   Grp' ← bisectionGrp(Grp, Area');

   Area ← Area', Grp ← Grp';
}

map0 ← makeInitMap(Area, Grp);
```

図 3 BISEM プログラム

5.3 OPTIM の実装

OPTIM における並列実行用のマスタノードのプログラムを図 4 に示す．

```
map ← map0;

while( !isFinal( ) ){
   list ← initOp(map, RANGE, NUM_SWAPS);

   sendToAllWorkers(map, list);
   wait( );
   results ← recvFromAllWorkers( );

   map ← update(map, results);
}
```

図 4 OPTIM におけるマスタノードプログラム

- *map* ，map_0：ランク配置を示し，map_0 は BISEM から受け取った初期ランク配置である．ランク配置はランクの配列として実装している．
- isFinal()：OPTIM が終了条件を満たしているか返す処理である．*map* があらかじめ決めた条件を満たすか，一定回数以上 while 文内の処理を行なうと終了条件を満たす．
- *list*←initOp(*map*, *RANGE*, *NUM_SWAPS*)：ランク配置 *map* からスワップするランクを決定する処理である．*RANGE* はスワップする範囲，*NUM_SWAPS* はスワップするランク数，*list* はスワップするランクを格納する配列である．*RANGE* はユーザにより指定することができるが，BISEM における *MAX_ITRS* の値に基づき適切に指定する必要がある．*MAX_ITRS* の値により決定される BISEM 実行後のネットワーク領域のサイズよりやや広い範囲を *RANGE* として指定する．*NUM_SWAPS* はネットワーク領域の広さや通信パターンを考慮してユーザが指定する．
- sendToAllWorkers(*map*,*list*)：マスタノードから各ワーカノードに *map* と *list* を送信する処理である．
- wait()：ワーカノードの処理が終了するまでマスタノードの処理を待機している．
- *results*←recvFromAllWorkers()：ワーカノードから処理結果を受け取り，*results* にセットする．*results* はワーカノードから受け取った新しいランク配置を格納する配列である．
- *map*←update(*map*, *results*)：*map* と *results* の中から新しいランク配置を決定し，*map* とする．新しいランク配置の決定は SA のアルゴリズムに基づく．

スワップするランクをマスタノードで一意に決定しワーカノードに送信することで，新しいランク配置の生成処理を全ワーカノードで統一することができる．

ワーカノードはマスタノードから受け取った *map* と *list* を元に，新しいランク配置を生成し評価する処理を一定回数繰り返し，マスタノードに送信する．

評価には以下の評価関数を用いる．

$$eval(map) = \sum_{i,j \in \text{全ランク}} hop_{i,j} \times \max_{link}$$

$hop_{i,j}$ はランク i, j 間のホップ数である．この関数により対象のランク配置における通信ホップ数と輻輳を評価する．

6. RMATT を用いた通信処理最適化の流れ

RMATT による通信処理最適化の流れを図 5 に示す．

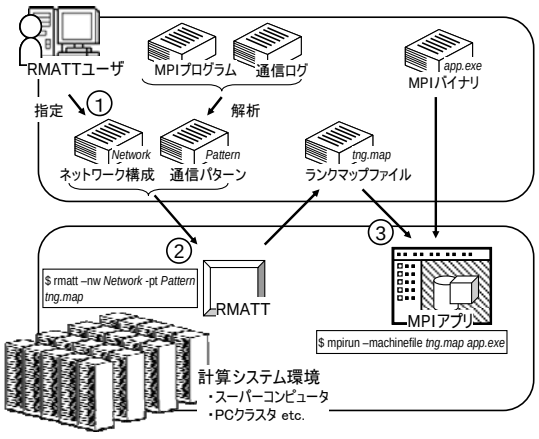

図 5 RMATT による通信処理最適化

1. ネットワーク構成と通信パターンの決定

RMATT ユーザは，MPI アプリケーションの通信パターンと，アプリケーションを実行したい計算システムのネットワーク構成(トポロジとノード数)を決定する．通信パターンはMPIプログラムやプロファイラの解析により得られる．図 5 ではネットワーク構成を *Network*，通信パターンを *Pattern* としている．

2. RMATT の実行
   ユーザはステップ 1 で決定したネットワーク構成と通信パターンを入力値とし，rmatt コマンドを用いて計算機上で RMATT を実行する．必要に応じてワーカノードを指定し RMATT を並列実行することができる．実行結果として RMATT は MPI のランクマップファイル(図 5 では *tng.map*)を出力する．
3. MPI アプリケーションの実行
   RMATT によるランクマップファイルを元に，目的のアプリケーションを実行する．図 5 では RMATT が出力したランクマップファイル *tng.map* を引数に，MPI バイナリ *app.exe* を mpirun コマンドにより実行している．

RMATT の実行により自動的に通信処理の最適化が可能となるため，ユーザの余暇時間や他作業を行なう時間に余剰 CPU 上で RMATT を実行することで，ユーザは時間や計算資源を有効に利用することができるようになる．

7. RMATT の評価

7.1 目的

RMATT が最適化したランク配置の通信処理時間や，最適化に要した時間を測定し，RMATT のランク配置最適化性能を評価する．また，RMATT が最適化したランク配置の通信処理を解析し，通信処理時間が短縮される原因を考察する．さらに，$\max_{link}$ と輻輳の関係と，BISEM による探索空間の圧縮状況を確認する．

7.2 評価プログラムと評価環境

4096 ランクの Allgather bruck アルゴリズム [9]を 16x16x16 ノードの 3D Torus に配置する．図 6 は Allgather bruck アルゴリズムの通信パターンである．

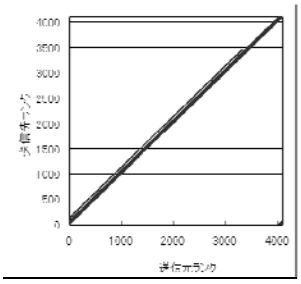

図 6 Allgather bruck 4096 ランクの通信パターン

bruck アルゴリズムは MPI_Alltoall や MPI_Allgather などに一般的に用いられているアルゴリズムである．木構造の通信パターンであるため Fat-Tree などの間接網では有効であるが，Mesh や Torus などの直接網ではランク配置により大きく通信処理時間が異なる．実際の Allgather では通信に $\log_2 4096=12$ ステップを要するが，本問題では模擬的に 1 ステップで実行している．Allgather の通信サイズは InfiniBand のパケットサイズである 2048 バイトとした．bruck アルゴリズムでは n ステップ目におけるランク間の通信サイズは 2048×2^n となる

測定にはネットワークシミュレータ OpenNSIM [8]を用いた．OpenNSIM は Torus や Mesh ネットワークにおいて任意の MPI プログラムを実行した際の通信処理をシミュレートするツールであり，通信処理時間や各ルータのバッファ使用量を計算することができる．OpenNSIM の設定を表 1 に記す．

表 1 OpenNSIM の設定

ネットワーク構成	3D Torus 16x16x16 ノード
ノード間バンド幅	5GB/秒
ハードウェアレイテンシ	120 ナノ秒

RMATT の実行環境として，表 2 の Xeon サーバ 21 台を用い，うちワーカノードに 20 台を用いた．1 コアにワーカノード 1 ノードを割り当て，ワーカノードの総数は 160 ノードとした．

表 2 Xeon サーバ構成

CPU	XeonE5520×2
メモリ	48GB
ネットワーク	GBEther

7.3 結果

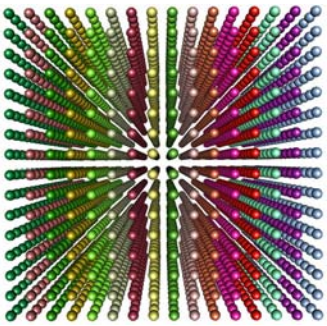

図 7 Allgather のランク配置(x,y,z 順に配置)

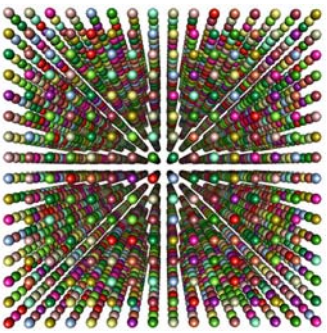

図 8 Allgather のランク配置(RMATT による配置)

ランクの配置状況の評価

ランク配置最適化を行なわず x,y,z 順にランクを配置したネットワークを図7に，RMATT によりランク配置されたネットワークを図 8 に示す．各ランクは最適化しない場合における x,z 面で 1 色ずつ色づけしている．最適化しない場合では規則的に並んでいることが確認できるが，RMATT によるランク配置では人手では非常に困難なランク配置となっていることが確認できた．

通信処理時間の評価

ランク配置最適化をしない場合と RMATT によるランク配置について，OpenNSIM による測定結果を表 3 に示す．

表 3 Allgather の通信処理時間

ランク配置最適化無し(x,y,z 順のランク配置)	22.3 ミリ秒
RMATT によるランク配置	5.5 ミリ秒

表 3 から，RMATT によるランク配置は最適化しない場合よりも通信処理時間が約 75%削減されていることがわかる．

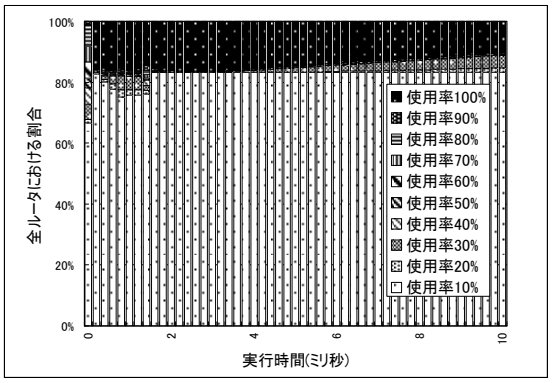

図 9 ランク配置の最適化をしない場合のバッファ使用率

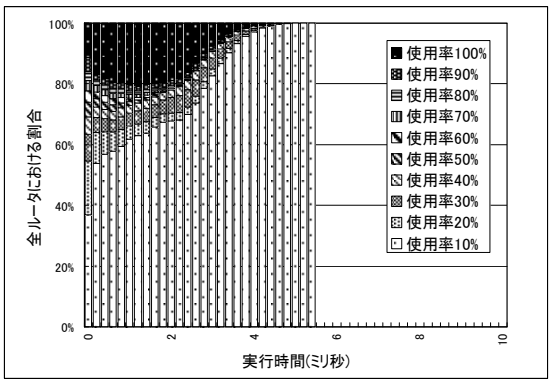

図 10 RMATT によるランク配置のバッファ使用率

輻輳の確認

Allgather の通信処理時の各ルータの輻輳状況を確認するため，処理開始 10 ミリ秒間の各ノードにおけるルータのバッファ使用率を測定した．図 9 は最適化しない場合の，図 10 は RMATT により最適化されたランク配置におけるバッファ使用率である．図 9 ではバッファ使用率が 100%のルータが減少していないことから，長時間輻輳が発生していると考えられる．一方，図 10 では短期間でバッファ使用率が下がっていることから，輻輳した期間も短くなっていると考えられる．

max_{link}と輻輳の関係の確認

また，max_{link} と輻輳との関係を調査するため，各リンクを流れたバイト数を計算し，ヒストグラムにした．

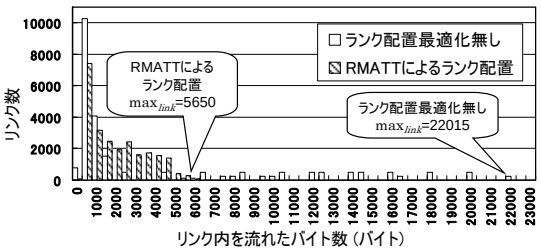

図 11 各リンクを流れたバイト数

図 11 において横軸はリンク内を流れたバイト数を，縦軸はリンク数を示す．各ランク配置における横軸の最大値は評価関数に用いられている max_{link} の値となる．図 11 では,RMATT によるランク配置は最適化無しの場合に比べ max_{link} の値が下がることにより，リンク内を流れたバイト数の分布が小さくまとまっていることが確認できる．極端に流れが大きいボトルネックとなるリンクが無くなることにより，輻輳の発生が起こりにくくなり，その結果通信処理時間が短縮されたと考えられる．

BISEM による探索空間の圧縮の評価

ランク配置最適化の探索空間について，N 個のランクに対し BISEM により m ランクずつ N/m 個のランクグループに分割した際のランク配置の総組み合わせ数は$(N/m)\times m!$となる．本対象問題では BISEM による組み合わせ数は 20,643,840 個となり，BISEM を行なわなかった場合は $3.64\times10^{13,019}$ 個となることから，BISEM により探索空間が大幅に圧縮されたと考えられる．

RMATT 実行時間の評価

RMATT の実行時間には約 16 時間を要した．これは評価値がこれ以上減少しないところ(最適解)まで実行した結果である．一般にメタヒューリスティクスによる最適化では，初期段階で急速に最適化が進み，その後時間を掛けて少しずつ最適解に収束する傾向がある．本問題においても，最適化の進捗は開始 1 時間で約 55%，4 時間後には 80%進んでいた．なお，ある時間における最適化の進捗は以下の式で求めている．

$$(\text{最適化の進捗})=\frac{(\text{ある時間における評価値})-(\text{最適解における評価値})}{(\text{初期配置における評価値})-(\text{最適解における評価値})}$$

本問題では最後まで最適化を行なったため長時間を要したが，ユーザの目的によっては短期間で十分満足するランク配置を得ることもできる．RMATT の実行時間と最適化の進捗の関係については今後も調査を行ないたい．

並列化によるスケーラビリティの評価

160 ノードのワーカノードが 16,000 個のランク配置を評価するまでに要した時間は約 24 秒であった．1 ワーカノード内におけるランク配置 1 個の評価処理時間を測定したところ約 0.18 秒となった．並列処理せず，16,000 個のランク配置を評価する場合 0.18×16,000=2,816 秒かかることになる．このことから 160 ノードのワーカノードを使用する場合と並列化しない場合を比較すると，評価処理時間は約 1/117 に短縮されることが計算により求められた．RMATT プロトタイプは通信にソケット通信を用いていることから，マスタノード・ワーカノード間の通信がボトルネックになっていると考えられる．今後プログラムや通信処理を高速化することで RMATT の処理時間はさらに短縮されると考えられる．

8. 関連研究

大規模環境におけるランク配置最適化の研究は，IBM [13]などで行なわれ BlueGene 上で実行されるアプリケーション [14]などに利用されている．

IBM におけるランク配置最適化 [13]では，大規模問題に対応するために RMATT と同様にランクを複数のグループに分けることで探索空間を圧縮している．しかしネットワークの分割は行なわない．まず分割したグループを特定の条件に基づきランキングする．次にその順位に従い，グループに属するランクをネットワークに配置している．ランクを配置する際には，ランクグループ間の通信関係は考慮せず，ランクグループ内のランクを固めて配置することも考慮していない．

N個のランクに対しmランクずつN/m個のランクグループに分割した際のランク配置の総組み合わせ数は ${}_{N}P_{m}+{}_{(N-m)}P_{m}+\cdots+{}_{m}P_{m}$ となる．4096 ランクの場合，総組み合わせ数は 4.512×10^{30} 個となる．これは分割しない場合より小さいが，RMATT よりも大きい値となる．

また，ランク配置の評価についても RMATT と異なり，ホップ数のみを計算し輻輳については考慮していない．

9. おわりに

本論文では，大規模で多様な計算システムやアプリケーションに対するランク配置最適化ツールとして，分割手法とメタヒューリスティクスを用いた手法を並列に処理するツール RMATT を提案した．また，ランク配置を評価するために，ホップ数と輻輳の危険度を組み合わせた評価関数を RMATT には用いた．本ツールを実装し，OpenNSIM 上で 4096 ランクの Allgather アルゴリズムの通信処理時間を評価したところ約 75%の削減を確認した．また，輻輳が減っていることも確認し，RMATT による通信処理時間の改善の仕組みを解明した．

今後は，RMATT の性能調査をさらに進め，BlueGene などの関連研究との性能比較や，さまざまな実アプリケーションや実機上で評価を行ない,最適化性能の向上を図りたい．とくに，OPTIM について SA よりもさらに最適化性能の高い手法の導入や，さらなる高速化を行なうことを考えている．また，RMATT ユーザが作業時間や計算資源を有効に利用しやすくすることを目的に，チューニングの途中停止や再実行機能，チューニングの進捗状況の表示機能など，ユーザビリティの向上も行なう予定である．将来的にはチューニングツールとして一般ユーザへの提供も考えている．

参 考 文 献

[1] http://www.top500.org/

[2] Satoshi Matsuoka, "The Road to TSUBAME and Beyond, "HIGH PERFORMANCE COMPUTING ON VECTOR SYSTEMS 2007, Part6, pp.265-267, 2008.

[3] Hiroshi Nakashima, "T2K Open Supercomputer: Inter-University and Inter-Disciplinary Collaboration on the New Generation Supercomputer."Intl. Conf. Informatics Education and Research for Knowledge-Circulating Society (ICKS'08), Kyoto, Japan, Jan, 2008.

[4] http://accc.riken.jp/ricc.html

[5] Alam, S., Barrett, R., Bast, M., Fahey, M.R., Kuehn, J., McCurdy, C., Rogers, J., Roth, P., Sankaran, R., Vetter, J.S., Worley, P., Yu, W., "Early evaluation of IBM BlueGene/P, " High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for, pp.1-12, Nov, 2008.

[6] http://www.nccs.gov/

[7] http://www.nsc.riken.jp/index_j.html

[8] https://ngarch.isit.or.jp/taas/opennsim/

[9] Bruck, J., Ching-Tien Ho, Kipnis, S., Upfal, E., Weathersby, D., "Efficient Algorithms for All-to-All Communications in Multi-Port Message-Passing Systems, "Parallel and Distributed Systems, IEEE Transactions on, Vol.8, Issue 11, pp.1143-1156, Nov, 1997.

[10] Scott Kirkpatrick, "Optimization by simulated annealing: Quantitative studies, "Journal of Statistical Physics, Vol.34, Number 5-6, pp.975-986, Mar, 1984.

[11] George Karypis, Vipin Kumar, "A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs, "IAM Journal on Scientific Computing, Vol.20, No.1, pp. 359-392, 1999.

[12] Nagel W.E., Arnold A., Weber M., Hoppe H-C., and Solchenbach, K., "VAMPIR: Visualization and Analysis of MPI Resources, "Supercomputer 63, Vol.12, No.1, pp.69-80, 1996.

[13] Bhanot, G., Gara, A., Heidelberger, P., Lawless, E., Sexton, J. C., Walkup, R., "Optimizing task layout on the Blue Gene/L supercomputer, "IBM Journal of Research and Development, Vol.49 Issue:2.3, pp.489-500, March, 2005.

[14] Ethier, S., Tang, W. M., Walkup, R., Oliker, L., "Large-scale gyrokinetic particle simulation of microturbulence in magnetically confined fusion plasmas, "IBM Journal of Research and Development , Vol.52 Issue 1.2, pp.105-115, Jan, 2008.