10000 タスクに対応するタスク配置手法 SFMOPT の提案と評価

浅野 悠[†] 吉瀬 謙二[†]

近年，メッシュ/トーラス接続網を採用する数万ノードを超える大規模な計算機が登場している．メッシュ/トーラス接続網では，計算タスクをどのノードに配置するかによって並列アプリケーションの実行時間が変化する．ここで，タスクを配置するノードを決定する問題をタスク配置問題と呼ぶ．この問題では，タスク配置の組み合わせ数が多く，プログラマが手動で最適な配置を決定することが難しい．以前我々はこの問題を現実的な時間で解くことのできるタスク配置手法 MOPT(Merge OPTimization) を提案した．本稿では，重複した試行の排除や並列化といった手法を用いて，MOPT の最適化時間を短縮した FMOPT(Fast MOPT) を提案する．またサブグループ化の導入により，FMOPT を更に高速化した SFMOPT(Subgrouping FMOPT) を提案する．SFMOPT を用いることで，今まで 2 時間かかっていた 8192 ノードの最適化問題を 31 秒で解くことが可能となる．

The Proposal and Evaluation of SFMOPT, a Task Mapping Method for 10000 Tasks

Haruka Asano[†] and Kenji Kise[†]

In recent years, computing systems which adopt mesh/torus network have appeared. Their communication time differs in where the positions of calculation tasks are located in such a direct network. Especially the positions of communicated tasks are important. The problem to determine the positions of all tasks is called a task mapping problem. The task mapping problem is complex because there are a lot of combinations. Therefore, it is difficult for the programmers to seek the best task mapping. We proposed a fast task mapping method MOPT (Merge OPTimization) before. In this paper, we propose FMOPT (Fast MOPT) which shortened the optimization time of MOPT using duplicate exclusions and parallelization. We also propose SFMOPT (Subgrouping FMOPT) which further accelerated FMOPT by introducing subgrouping. By using SFMOPT, we can optimize 8192 nodes in 31 seconds which had taken 2 hours before.

1. はじめに

近年のスーパーコンピュータや大規模計算機では，多数のコンピュータノード (ノード) を接続するシステムが一般的である．また，今後の技術として多数のプロセッサを接続するメニーコアシステムも提案されている．

多数のノードから成るマシンを構成する接続網として，**間接網**と**直接網**が存在する．間接網に比べ，個々のノード同士をリンクで直接接続する直接網は構成の柔軟性とスケーラビリティに優れている．このため多数のノードから成るシステムでは直接網が採用される．また近年の大規模計算機では，直接網としてメッシュ/トーラスネットワークを採用することが多い．メッシュ/トーラスネットワークを用いるシステムの例として，京コンピュータ[1)], Jaguar[2)] などが挙げられる．

メッシュ/トーラスネットワークには，ノード間の関係が均一ではないという問題がある．例えば，データを送信するノードと受信するノードの位置によって通信のホップ数が異なる．また，ネットワークのスループットも他のノードの通信状況に影響を受ける．このため，計算タスク (タスク) をどのノードに割り当てるかによってアプリケーションプログラムの実行時間が変化する．ここで，タスクを処理するノードを決める問題を**タスク配置問題**と呼ぶ．

タスク配置問題では，例えば，1 次元の ID を持つ MPI(Message Passing Interface) で記述された並列プログラムの場合，その ID を 2 次元または 3 次元のメッシュ/トーラスネットワークのどのノードに割り当てるかを決定する．MPI で記述されたプログラムは多数存在し，タスク配置によって性能が変化する．このため，タスク配置問題を解く高速で高性能な手法が必要とされる．

† 東京工業大学 大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology

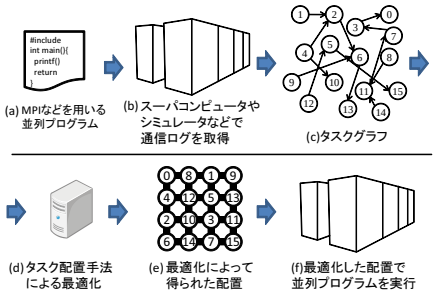

図 1 タスク配置手法の適用プロセス

タスク配置問題では，その配置の組み合わせ数が多いため最適解を求めることが難しい．N 個の計算タスクを N 台のノードに割り当てる場合の組み合わせ数は $N!$ であり，N が大きい場合に，この全探索は現実的ではない．そこで我々は，1 万ノードを超えるシステムが対象であっても現実的な時間でタスク配置問題の解を得るタスク配置手法 MOPT(Merge Optimization) を提案した[3]．

MOPT を使うことによって，8192 ノードの最適化問題を 2 時間，16384 ノードの最適化問題を 5 時間で解くことが可能である．このままでも一日数回程度のタスク配置であれば実用可能であり，スーパーコンピュータにおける科学技術計算等では運用できる．しかし近年マルチコア化の普及により多くのコアを使用するプログラミングの敷居は下がってきており，今後もメニーコア化の流れにより，数千・数万タスクのプログラムを使用する頻度は高まると思われる．そうした状況でタスク配置を行うにあたり，一日に数回程度のタスク配置しかできないアルゴリズムでは使い勝手が悪く，より高速なタスク配置手法が求められる．本稿ではこの MOPT を高速化した FMOPT，及び FMOPT をベースとした更に高速な SFMOPT を提案し，その効果を評価する．

本稿の構成を以下に示す．2 章で MOPT の概要を述べる．3 章では MOPT を高速化する手法を提案する．4 章では提案した高速化の効果を評価する．5 章では関連研究を述べる．6 章でまとめ及び今後の課題を述べる．

2. MOPT の概要

2.1 タスク配置手法の適用プロセス

本稿では，メッシュ/トーラス接続網の高性能計算機を対象として，タスク配置問題の解を求めるタスク配置手法を扱う．ここで，**タスク**とはノードに割り当てる並列プログラムの処理である．1 つのノードには 1 つのタスクが割り当てられ，その割り当ては実行を通じて変わらないものとする．

図 1 に，タスク配置手法の適用プロセスを示す．(d) のタスク配置問題を解くために要する時間を**最適化時間**と呼ぶことにする．また，最適化後の実行である (f) において，実行時間のうち並列アプリケーションの通信に要する時間を**通信時間**と呼ぶことにする．タスク配置問題は，この通信時間が短くなるようなタスク配置を目的とする．

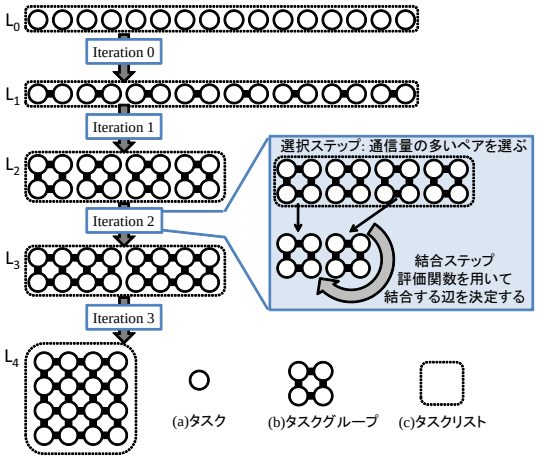

図 2 MOPT による最適化の流れ

2.2 MOPT の概要

MOPT は少ない最適化時間でタスク配置問題を解くアルゴリズムである．16 個のタスクを 4×4 のメッシュ接続されたノードに割り当てる例 (**図 2**) を用いて，MOPT による最適化の流れを説明する．また，いくつかの用語を定義する．

図 2 に示すように，MOPT は図 2(a) の丸で示すタスクを繰り返し結合することによって最終的な配置を求める．それぞれの繰り返しにおける一連の処理を**イテレーション**と定義する．

図 2(b) に示す結合されたタスクを**タスクグループ**と定義する．ただし，イテレーション 0 の入力となる単一のタスクもタスクグループである．図 2(c) の点線で示す，各イテレーションで結合されたタスクグループの集合を**タスクリスト**と定義する．

各イテレーションでは，入力となるタスクリストに含まれるタスクグループを 2 つずつペアにして結合させ，その結果をタスクリストとして出力する．図 2 のように 16 個のタスクを結合する場合，最終的な配置を求めるまで 4 回のイテレーションが必要となる．

各イテレーションの処理は，図 2 の色つき枠に示す通り**選択ステップ**と**結合ステップ**に分けられる．

選択ステップでは，タスクリスト中から結合させるタスクグループのペアを決定する．具体的な手順としては，まず入力タスクリストに含まれる全ての 2 タスクグループ間について通信量を計算する．そして，最も通信量の多いタスクグループのペアを結合対象として選択する．選ばれた 2 つのタスクグループをタスクリストから排除し，再びタスクリスト中で通信量が最大となるタスクグループのペアを選択する．これを入

力タスクリストが空になるまで行う．

選択ステップで用いる，2 つのタスクグループ間の通信量を式 (1) に定義する．G_1,G_2 はタスクグループ，s,t はタスク，$Traffic(s,t)$ はタスク s とタスク t の通信量を示す．

$$Comm(G_1, G_2) = \sum_{s \in G_1} \sum_{t \in G_2} Traffic(s,t) \quad (1)$$

結合ステップでは，選択ステップで選んだ各タスクグループのペアを，それぞれどのように回転・反転させて結合させるか決定する．ここでは各タスクグループのペアに対し，評価関数による評価値が最小となる回転・反転の組み合わせによる結合を採用する．こうして結合されたタスクグループを出力タスクリストに加え，次のイテレーションに進む．

結合ステップの評価関数には `Eval_Cost` と `Eval_Link` の 2 つを用いている．`Eval_Cost` は，結合候補として一時的に結合したタスクグループに含まれる各タスク間における通信バイト数とホップ数の積 (通信コスト) の総和を評価値とする．`Eval_Link` は，一時的な結合をしたタスクグループで XYZ 次元順ルーティングに基づく通信が行われたとき，最も多くのデータが流れるリンクにおける流れたバイト数を評価値とする．評価関数に `Eval_Cost` を用いた MOPT を **MOPT mincost**，評価関数に `Eval_Link` を用いた MOPT を **MOPT minlink** と呼ぶ．

結合ステップにおいては先述の通り 2 つのタスクグループそれぞれの反転・回転パターンを評価するが，この反転・回転パターンを**配置パターン**，配置パターンの組み合わせを**結合パターン**と呼ぶことにする．また，タスクグループを結合する方向軸を**結合軸**と呼ぶ．結合軸はイテレーション毎に固定であり，2 次元ならば X 軸と Y 軸を交互に繰り返し，3 次元ならば X,Y,Z 軸をこの順で繰り返す．

3. MOPT 高速化手法の提案

3.1 高速化の方針

MOPT を高速化するため，まず我々は MOPT における実行時間の内訳を調べた．MOPT による最適化時間のうち，多くを占めるのは結合ステップにおける評価関数であった．例えば 8192 ノードの MOPT minlink の場合，最適化時間全体のうち結合ステップの実行時間が 99.8%，結合ステップに含まれる評価関数の実行時間が 98.5%を占める．今回は数十倍から数百倍の高速化を目指し，結合ステップの高速化を行う方針とした．

3.2 FMOPT の提案

まず，我々は MOPT に対し結合パターン数の削減及び並列化を適用した **FMOPT**(Fast MOPT) を提案する．FMOPT は従来の MOPT と出力結果が全く同じかつ MOPT より高速化であるため，FMOPT は MOPT に取って代わることが可能である．

3.2.1 結合パターン数の削減

我々は結合ステップの等価性に着目し，結合ステップで評価を行う結合パターン数を削減した．以下に結合ステップにおける結合パターン生成の実装について述べる．MOPT において，各配置パターンは**置換**及び**反転**という 2 つのパラメータで示される．置換は元々の X,Y 軸 (3 次元ならば X,Y,Z 軸) の入れ替えを差し，反転は X 軸，Y 軸 (3 次元ならば XY 平面，YZ 平面，ZX 平面) それぞれに鏡を置いたかのような反転をすることを示す．便宜上，YZ 平面に対する反転を X 反転，ZX 平面に対する反転を Y 反転，XY 平面に対する反転を Z 反転と呼ぶ．また，結合後タスクグループのサイズを一定とするため，長さが違う辺の置換は行わない．2 次元で正方形を結合させる場合，置換が 2 通り，反転が 4 通りで計 8 通りの配置パターンが存在する．また 3 次元で立方体を結合させる場合，置換が 6 通り，反転が 8 通りで計 48 通りの配置パターンが存在する．

MOPT では，これら 8 又は 48 通りのうち結合可能な全ての結合パターンを評価していた．しかし，異なる配置パターンの組み合わせであっても結合したタスクグループは同じ評価値となることがある．このようなタスクグループの評価は重複して行うべきでないが，従来の MOPT ではその点が考慮されておらず，全ての配置パターンの組み合わせを評価していた．今回我々はどのようにすればこのような重複を排除できるか考察した．

まず，MOPT mincost の場合について考える．評価関数 `Eval_Cost` は全ノード間のホップ数が同じであれば同じ評価値となる．すなわち，ある結合パターンで生成されたタスクグループに置換や反転をかけても評価値は変わらない．

以下では，3 次元 MOPT における立方体の結合における評価値を議論し，評価値が同じになる場合を明らかにする．**図 3** は，3 辺 ABC を含む立方体のタスクグループと 3 辺 abc を含む立方体のタスクグループを，辺 A と辺 a が一直線に並ぶよう結合軸を X として結合させる場合を示す．このとき結合パターン (1) に対し `Eval_Cost` による評価値が等しく，かつ異なる配置パターンの組み合わせによる結合パターンである (2) から (8) が存在する．(1) を結合軸方向に 90° ずつ回転させたものが (2) から (4) であり，(1) から (4) に Z 反転をかけたものがそれぞれ (5) から (8) となっている．この場合，(1) の結合パターンを評価すれば (2) から (8) を評価する必要はない．

MOPT minlink の場合は MOPT mincost と評価値の同じ結合パターンの種類が異なる．MOPT minlink では評価関数に XYZ 次元順ルーティングを使用しているが，軸の置換を行うと XYZ 次元順ルーティング

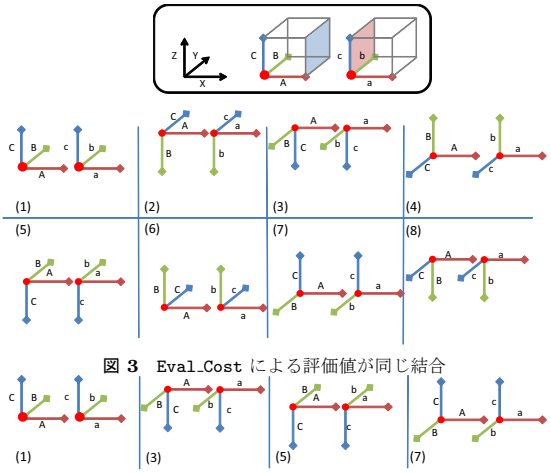

図 3 Eval_Cost による評価値が同じ結合

図 4 Eval_Link による評価値が同じ結合

の結果は変化するため，Eval_Cost による評価値も変化してしまう．しかし，ある結合パターンに対し反転をかけたものに対しては MOPT mincost と同様評価値は等しい．**図 4** は，図 3 のうち (1) と Eval_Link による評価値が同じ結合を抜粋したものである．

提案手法では，これら評価値が等しい結合パターンを排除する．単純なルールで削減を行えるよう，図 3 に示す (1) から (8) までの結合パターンの特徴を整理する．まず，(1) と (6) の左のタスクグループを比較すると，これらは結合軸以外を入れ替えた配置パターンであることが分かる．また，この (1)(6) の Z 反転が (5)(2)，Y 反転が (7)(4)，Y 反転・Z 反転の両方をかけたものが (3)(8) である．一般的には，ある片方の配置パターン A に対し，特定の回転・反転をかけた A' に対するパターンは，全て A に対する結合パターンのいずれかと評価値が同じとなる．そのため，こうした A' については評価を行わない．この A' に該当するのは

(1) A の結合軸を除く 2 軸を入れ替えた配置 2 通り (MOPT minlink の場合)
(2) A の結合軸以外の反転 4 通り

であり，これらの評価を省くことで評価すべき結合パターンは MOPT mincost の場合 1/8，MOPT minlink の場合 1/4 になる．

直方体の場合，辺の長さが変わってしまうような置換は不可能であるため評価値が同じとなる結合パターン数は異なる場合がある．3 次元 MOPT において，2 つのタスクグループ (G_1,G_2 とする) に対して X 結合・Y 結合・Z 結合の順を繰り返し結合を行うと考えると，従来手法において評価する結合パターン数は**表 1**，そのうち提案手法において評価する結合パターン数と従来手法に対する削減率は MOPT mincost の場合は**表 2**，MOPT minlink の場合は**表 3** に示す通りとなる．提案手法の適用により，MOPT mincost は 6 から 7 倍程度，MOPT minlink は 4 倍の高速化が見込める．

表 1 従来手法の結合パターン数

結合	G_1		G_2		計
	置換	反転	置換	反転	
X	6	8	6	8	2304
Y	2	8	2	8	256
Z	2	8	2	8	256

表 2 提案手法の結合パターン数 (FMOPT mincost)

結合	G_1		G_2		計	削減率
	置換	反転	置換	反転		
X	3	2	6	8	288	1/8
Y	2	2	2	8	64	1/4
Z	1	2	2	8	32	1/8

表 3 提案手法の結合パターン数 (FMOPT minlink)

結合	G_1		G_2		計	削減率
	置換	反転	置換	反転		
X	6	2	6	8	576	1/4
Y	2	2	2	8	64	1/4
Z	2	2	2	8	64	1/4

2 次元版 MOPT の場合も，削減する結合パターンの法則性は基本的に変わらない．ただし Z 軸がないため，Z 反転を含む結合パターンは排除する．

3.2.2 結合ステップの並列化

結合ステップにおける評価関数は同じイテレーションの結合パターン間で独立であり，互いの結果が互いに影響しない．よってこの評価は並列計算できる．

FMOPT のスレッドはイテレーション毎に同期を取る．各イテレーションでは，タスクリストに含まれる全タスクグループの全結合パターンを並列に評価する．

3.3 SFMOPT の提案

FMOPT は MOPT に比べ常に一定の高速化が見込まれるが，これはノード数が増加しても同様である．すなわち，タスク数の増加に伴う計算時間の増加率は改善されていない．FMOPT では，タスク数が増加すると結合ステップにおけるタスクグループのサイズ増大によって評価にかかる時間が大きく増加する．我々は，サブグループ化によって評価関数の時間を大きく削減する **SFMOPT**(Subgrouping MOPT) を提案する．この SFMOPT は FMOPT の近似解法である．

SFMOPT では，イテレーション回数がある程度大きくなってきたら，タスクグループに含まれるの複数個のタスクを**サブグループ**にまとめる．以降この操作を**サブグループ化**と呼ぶ．2 次元 SFMOPT において，4×4 のタスクを含むタスクグループで 2×2 のタスクをサブグループ化する様子を**図 5** に示す．図 5 において，2×2 のサブグループを囲んだ枠それぞれがサブグループとなる．

評価関数による評価を行う際は，このサブグループを一つのタスクとみなす．サブグループ間の通信量は，式 (1) で定義したタスクグループ間の通信量と同様に

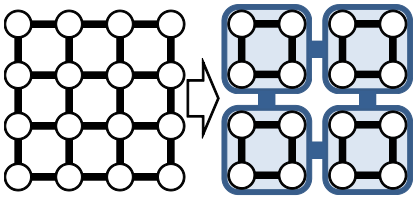

図 5 2 × 2 のタスクをサブグループ化する例

定義する．図 5 のタスクグループの評価を例にすると，サブグループ化を行わない場合，このタスクグループの評価を行うためには 4 × 4 のタスク間通信量を考慮して計算を行う必要がある．一方 2 × 2 のタスクをサブグループ化した場合，評価関数は 2 × 2 のサブグループ間通信量のみを考慮すればよい．すなわち，タスクグループを評価する時間は 2 × 2 のタスクが含まれるタスクグループを評価する時間とほぼ等しくなる．今回用いる評価関数の時間計算量はタスク数に依存しているため，タスク数が減少することで評価関数の高速化が見込める．

現在の SFMOPT においてサブグループの形状は 2 次元の場合は正方形，3 次元の場合は立方体に限定している．これはサブグループ間通信量を求めるオーバーヘッドを考慮している．サブグループの形状を正方形又は立方体にした場合，全ての結合パターンのサブグループ構成が変わらない．そしてこのサブグループは以前のイテレーションにおけるタスクグループであり，それらの通信量は選択ステップで計算した結果を再利用可能である．よってサブグループ間通信量を求めるオーバーヘッドはほぼなくなる．一方正方形又は立方体の形状ではない場合，置換の組み合わせによって結合パターンのサブグループ構成が変化する．それらのサブグループは以前のイテレーションのタスクグループではない場合も存在し，このとき通信量の再計算によるオーバーヘッドが発生する．

サブグループ化を行うと，タスクグループの評価値の大小関係はサブグループ化しない場合と異なってくる．評価関数による評価の精度が落ちるため，SFMOPT は FMOPT より評価値の悪いタスク配置を出力してしまう可能性が高いと考えられる．しかし通信時間の良し悪しは必ずしも評価値の大小と同じではない．むしろ SFMOPT は FMOPT に比べ大局的な評価をしているとも言えるため，SFMOPT の出力したタスク配置の方が FMOPT のそれより通信性能が良いというケースも考えられる．

SFMOPT では，サブグループのサイズとサブグループ化を開始するイテレーションを任意に設定できる．また現段階では実装していないが，パラメータを複数設定することも可能である．例えば，6 イテレーションからタスクを 2 × 2 × 2 のサブグループにまとめ，9 イテレーション目からタスクを 4 × 4 × 4 のサブグループにまとめるといった事が可能である．このように任意のイテレーションでタスクを任意のサイズのサブグループにまとめることで，タスクグループ内のタスク数を柔軟に調節することが可能となる．これにより，タスク数の増加による評価関数の時間増加をコントロールすることが可能である．

以下ではサブグループ化による高速化の上限について述べる．D 次元 SFMOPT(D は定数) にて，全体のタスク数を N とする．全イテレーションにおける全タスクグループについて，サブグループのサイズを最大限にする場合最も高速である．このときタスクグループに含まれるサブグループ数は 2^D 個以下なので，結合パターン毎の評価関数は $O(1)$ である．結合パターン数も定数であるため，各タスクグループペアの結合パターン決定は $O(1)$ となる．タスクグループのペア数は $O(N)$ であるため，最大限の高速化をした場合に結合ステップは $O(N)$ となる．

4. 評　　価

4.1 FMOPT の評価

従来の MOPT と比較しどの程度高速化したか調べるためにノード数を変化させながら時間計測を行い，高速化率を算出した．この計測は Intel Core i7 870 を搭載する計算機上で，Linux の `gettimeofday()` 関数を使用することにより行った．並列化の実装には OpenMP を用いた．入力にはダミーのデータを用いたが，MOPT の最適化時間は入力データに依存しないため，シミュレーション等から得られるトレースを用いた場合と比較して最適化時間に変化はない．

オリジナルの MOPT と比較し，結合パターン数の削減による高速化を施した MOPT(並列化なし FMOPT) の高速化率を**図 6** に示す．図 6 より，並列化なし FMOPT の MOPT に対する高速化率は表 2 及び表 3 における削減率とほぼ等しく，FMOPT min-link は 4 倍，FMOPT mincost は 5 倍から 8 倍の間である．また FMOPT mincost においては，最終イテレーションにて立方体同士の結合を行う 128，512，8192 ノードにおいて高速化率が大きくなっている．これは表 2 に示す通り，立方体同士の結合は削減率が大きく，更に最終イテレーションであり占める時間の割合が大きいためである．

並列化なし MOPT と比較し，並列化した FMOPT の高速化率を**図 7** に示す．図 7 より，4 並列でおおよそ 2，3 倍の高速化を達成できたことが分かる．mincost の方が結合ステップの所要時間が短く，最適化時間全体のうち結合ステップの占める割合が少なくなるため高速化率は小さい．

次に，並列化をした FMOPT の実行時間とノード数の関係を**図 8** に示す．図 8 より，最適化時間は 1024，8192 ノードで大きく増加していることが分かる．512 ノードから 1024 ノード，4096 ノードから 8192 ノー

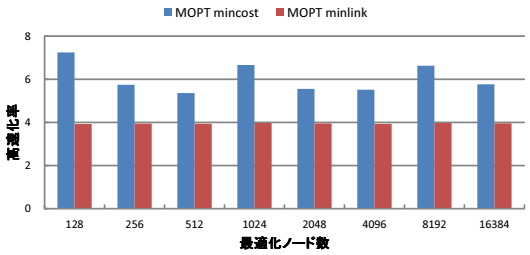

図 6 結合パターン数の削減による高速化率

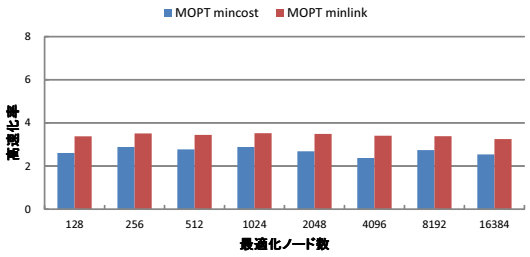

図 7 並列化による高速化率

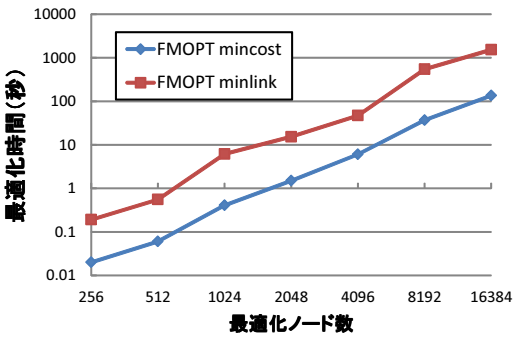

図 8 FMOPT による最適化時間

ドは立方体タスクグループの結合を行うイテレーションが追加される．前述の通り立方体タスクグループの結合はパターン数が多いため，最適化時間も大きく増加する．最も 10000 タスクに近い 8192 ノードの `Eval_Link` を用いる最適化時間を比較すると，MOPT は 7348 秒，4 並列 FMOPT は 547 秒となった．

4.2 SFMOPT の評価

SFMOPT について，2 つの評価を行った．1 つは FMOPT と同じ様に最適化時間の変化である．もう 1 つは，出力として得られたタスク配置を適用した際の評価値及び通信時間である．いずれの評価についても，ターゲットの計算機は 3 次元メッシュ/トーラスとした．

この節において比較する手法を以下のように定義する．

XYZ
: 最適化を行わず，XYZ 次元順にタスクを並べる手法

None
: 結合ステップにて各結合パターンの評価を行わず，常に一定の結合パターンを採用する手法

FMOPT
: FMOPT を用いる手法

SFMOPT
: SFMOPT を用い，イテレーション N から 2×2×2 のタスクをサブグループ化する提案方法

FMOPT-half
: 上記のイテレーション N 直前まで FMOPT の結合ステップを行い，以降は None と同じ結合ステップを行う手法

サブグループ化を適用するイテレーションは評価毎に定める．

FMOPT-half を比較対象とした理由は 2 つ存在する．1 つは，FMOPT-half はイテレーション N 以降の計算を完全に省くため，イテレーション N 以降の計算を削減する手法の中で最も高速だからである．最適化時間において，SFMOPT が FMOPT-half より良くなることはない．2 つ目は，SFMOPT におけるサブグループの評価，すなわちイテレーション N 以降の計算の必要性を議論するためである．SFMOPT が良いタスク配置を求めた場合，選択ステップ及び FMOPT と同じイテレーション N までの評価が重要で，それ以降のサブグループ化による評価は重要ではなかったという可能性が存在する．FMOPT-half と比較することで，サブグループ化したタスクグループの評価がどの程度タスク配置の性能に影響するかが分かる．

4.2.1 最適化時間の評価

評価環境や入力データは FMOPT の評価と同じものを使った．サブグループ化の開始はイテレーション 9 とした．

図 9 は，評価関数に `Eval_Link` を用いた場合，FMOPT, SFMOPT, FMOPT-half の最適化時間を比較したものである．まず，FMOPT と SFMOPT を比較する．図 9 から，サブグループ化を使うことでノード数増加による計算時間の増加が大幅に抑えられていることが分かる．これは，サブグループ化により本来多くの時間を占めるイテレーション 9(512 ノードを含むタスクグループの結合) にかかる時間が大幅に減少し，圧縮をかける前のイテレーション 6 が支配的になるためである．このイテレーション 6 までに処理するタスクリストの大きさはノード数に比例するため，最適化時間も 1 次関数に近い形になったと考えられる．最も 10000 タスクに近い 8192 ノードの最適化時間を比較すると，FMOPT は 547 秒，SFMOPT は 31 秒となった．

次に，SFMOPT と FMOPT-half を比較する．FMOPT-half は結合ステップを途中から全く行って

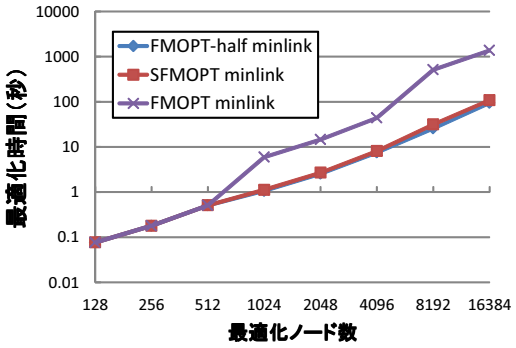

図 9　サブグループ化の有無による最適化時間の比較 (minlink)

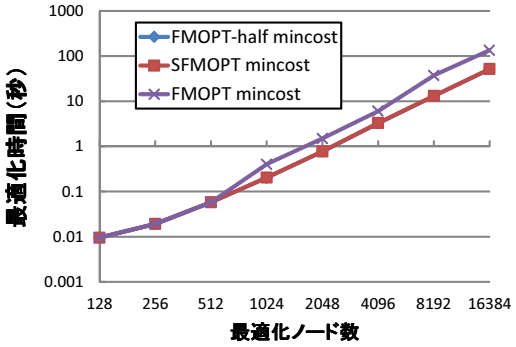

図 10　サブグループ化の有無による最適化時間の比較 (mincost)

いないにも関わらず，SFMOPT と最適化時間に大きな差はないことが分かる．最も差のある 8192 ノードにおいても，SFMOPT は FMOPT-half の 1.1 倍程度の最適化時間しかかからなかった．FMOPT-half と SFMOPT は実行時間を多く占めるイテレーションが同じであるため，このような結果になったと考えられる．

図 10 は，評価関数に `Eval_Cost` を用いた場合，FMOPT, SFMOPT, FMOPT-half の最適化時間を比較したものである．`Eval_Link` の場合と同様，SFMOPT は FMOPT と比較して計算時間の増加が抑えられており，FMOPT-half と計算時間に大差はないことが分かる．しかし，`Eval_Cost` は `Eval_Link` より計算時間が短い，すなわち最適化時間のうち結合ステップの占める割合が `Eval_Link` を使う場合より短いため，削減の効果は `Eval_Link` を使う場合に比べ低い．

4.2.2　評価値と通信時間の評価

それぞれのタスク配置手法で得られる配置を評価するため，NAS Parallel Benchmark のうち CG，BT，MG，SP の通信トレースを用いる．ノード数は 512，サブグループ化はイテレーション 6 以降に適用した．通信時間は OpenNSIM[4] を利用して計測した．

512 ノードの最適化は僅か 1 秒以内で終わるため，最適化時間の面でサブグループ化が必要な場面は少ない

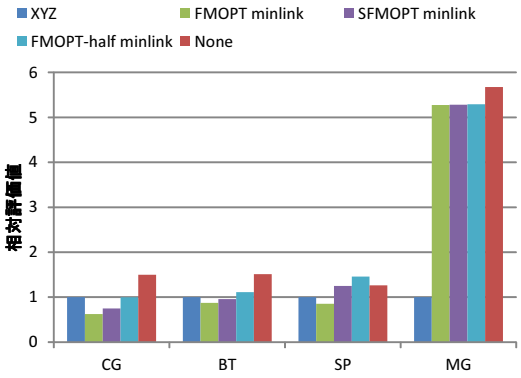

図 11　Eval_Link による評価値の変化

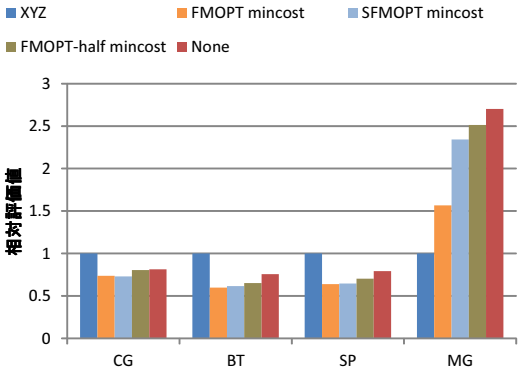

図 12　Eval_Cost による評価値の変化

と思われる．しかし今回，サブグループ化を行う対象として想定した 10000 ノード程度についてのトレースデータ及び評価環境が用意できなかったため，このような評価とした．

まず，それぞれの手法で最終的に得られたタスク配置の評価値について述べる．結合ステップの評価関数に `Eval_Link` を用いた場合について，その値を比較したグラフを**図 11** に示す．また評価関数 `Eval_Cost` を用いた場合について，その値を比較したグラフを**図 12** に示す．両グラフにおいて，縦軸は XYZ 次元順に配置した場合の評価値を 1 としたときの相対的な値である．2 章で述べた通り，いずれの評価関数においても評価値は低い方がよい性能となる．

まず，**図 13** に示す `Eval_Link` を使用する手法について述べる．MG を除くベンチマークでは，SFMOPT は FMOPT より性能は低下するものの，XYZ 次元順と比較すれば SP ベンチマークを除いて評価値は良い．FMOPT と SFMOPT で評価値に差があるのは，評価関数の性質によるものである．`Eval_Link` は通信量最大のリンク 1 本を評価するが，サブグループ間の通信量が多くとも，その中で通信量の多い単独のリンクがあるかはサブグループ化の際束ねたリンク間での

偏りに依存してしまう．このためサブグループ化により評価関数の大小関係が変化しやすい．また MG ベンチマークでは FMOPT・SFMOPT 共に上手く働いておらず，かえって性能低下を引き起こしている．これは MG ベンチマークが 3 次元グリッドを処理するアプリケーションであり，XYZ 次元順がほぼ最適解であるためと思われる．このように最初から XYZ 次元順に対し最適化されているプログラムに対しては，MOPT を使う意味はあまりないといえる．

次に，**図 14** に示す `Eval_Cost` を使用する手法について述べる．MG を除くベンチマークでは，`Eval_Link` の場合と異なり FMOPT と SFMOPT でほとんど評価値に違いがない．`Eval_Cost` ではネットワーク全体を評価し前述のような偏りは発生しないため，サブグループ化をしても評価関数の大小関係はそれほど変化しなかったと思われる．MG ベンチマークでは `Eval_Link` と同様，最適化によって性能低下を引き起こしてしまう結果となった．

次に，タスク配置を適用した際の通信時間について述べる．今回我々は，XYZ によって配置を行った場合の通信時間を 1 としたときの通信時間の相対的な高速化率を評価した．評価関数に `Eval_Link` を用いるものの評価を図 13 に示す．また，評価関数に `Eval_Cost` を用いるものの評価を図 14 に示す．個別ベンチマークと全ベンチマークの平均，及び MOPT を適用するに相応しくない MG を除いたベンチマークの平均を示す．

まず，図 13 に示す `Eval_Link` を使用する手法について述べる．一部のベンチマークで例外はあるが，MG を除く平均で見れば FMOPT，SFMOPT，FMOPT-half，None，XYZ の順で高速化率が高い．SFMOPT は FMOPT-half より高い高速化率であることからは，単に選択ステップや前半のイテレーションが重要だったというわけではないことが確認できる．また SFMOPT は FMOPT よりも少し性能は劣るが，その差は 5%未満であり，最適化時間も考慮すると SFMOPT は十分に効果的であるといえる．

次に，図 14 に示す `Eval_Cost` を使用する手法について述べる．SFMOPT が FMOPT-half，None，XYZ より高速化率が高いのは `Eval_Link` の時と同じであった．しかし，CG 及び SP においては SFMOPT が FMOPT よりも良くなっていた．この現象の原因は解析中であるが，プログラムの性質として CG 及び SP では `Eval_Cost` よりも `Eval_Link` の方が重要という可能性がある．該当するベンチマークにおいて，`Eval_Link` による評価値は FMOPT より SFMOPT の方が低かった．

5. 関 連 研 究

高性能計算機のためのタスク配置アルゴリズムとして，RMATT[5] が提案されている．RMATT はタスクを複数のグループに分割し，更に初期タスク配置を求める BISEM ステップと，Simulated annealing を使用し更なる最適化を行う OPTIM ステップから成る．前者の BISEM は SFMOPT に近い手法といえる．BISEM と SFMOPT を比較すると，BISEM は全体のタスクを分割してグループの作成と配置を行うのに対し，SFMOPT は 1 つ 1 つのタスクを結合させてタスクグループを拡大していく．最適化にかかる時間は BISEM より SFMOPT の方が短い．RMATT はヒューリスティック手法の一つである Simulated annealing を用いるため，時間をかけるほど差異的なタスク配置を求めることができる．ノード数が少なく十分な最適化時間がある場合，ヒューリスティックな手法がより良い配置を求める可能性がある．また，RMATT で用いられている評価関数は「ホップ数と最大リンク通信量の積の総和」であり，我々が用いている評価関数と異なっている．

RMATT では，その高速化手法が提案されている[6]．この手法は，Simulated annealing において変更されなかったランクについての計算の省略，通信しないノード間についてのテーブル作成の省略，という 2 つの高速化から成る．特に前者は評価関数の実行時間を削減するものであり，SFMOPT におけるサブグループ化の

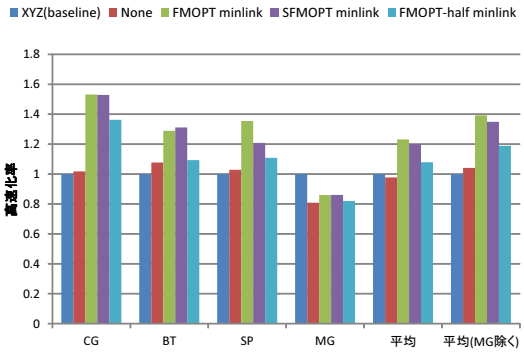

図 13 各配置による通信時間の高速化率 (minlink)

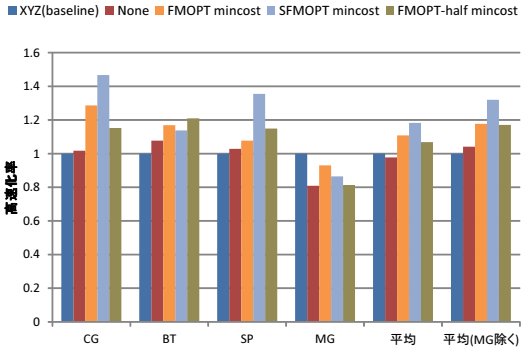

図 14 各配置による通信時間の高速化率 (mincost)

狙いと共通する．この高速化を施した RMATT においても，最適化にかかる時間は BISEM より SFMOPT の方が短い．

3 次元トーラスを採用した BlueGene/L 上のタスク配置アルゴリズムが文献 7) で提案されている．この手法では，タスクを優先順位の付いたグループに分割し，その順番にターゲットとなるトーラス上へ配置する．SFMOPT と異なり分割時点ではグループ内タスクの位置関係は決定されておらず，グループ内タスクが必ずしも固まった場所に配置されるとは限らない．配置の際にはホップ数のみを評価する点も，通信量を考慮する SFMOPT と異なる．

同じく高性能計算機のためのアルゴリズムとして，TAC3[8] が提案されている．RMATT と同様，Simulated annealing によりタスク配置を求める．TAC3 は同時転送を開始するタスクの集合を入力とし，通信の衝突が最少になるようなタスク配置をヒューリスティックに求めていく．通信の発生した時間を考慮するという点は，通信量だけを入力としている SFMOPT とは大きく異なる．

6. おわりに

数万ノードを超えるメッシュ・トーラス接続網を採用した計算機向けのタスク配置手法 MOPT は，その出力結果を変化させずに，結合パターン数を削減することで 4 倍，さらに 4 ノードを使用して並列化することで 3 倍程度の高速化が可能であること示した．また，平均 5%の通信時間増加と引き換えに評価関数の処理時間増加を抑える更なる高速化が可能になることを示した．

今後 SFMOPT を更に高速化する場合は，選択ステップの計算時間が無視できないと思われる．従来の MOPT において選択ステップの実行時間は全体の 1%未満とごく短時間なものであったが，今回結合ステップの高速化を行ったことで実行時間中の選択ステップの割合は増加している．16384 ノードの最適化だと選択ステップは SFMOPT minlink で 32%，SFMOPT mincost で 67%を占める．今後の課題として，計算量の議論とともに選択ステップについて考慮する必要があると考えられる．

高速化以外の課題としては，MOPT の高精度化が挙げられる．FMOPT は少ないノード数において十分に高速とも考えられる．例えば 512 ノードの SFMOPT minlink による最適化は 0.4 秒で終了する．このようなノード数ではサブグループ化による高速化のコンセプトとは逆に，多少時間を犠牲にしてよりよい結果を出したいというケースが存在すると考えられる．

また近年のスーパーコンピュータにおける接続網は，京の 6 次元，BluGene/Q の 5 次元など 3 次元を超えることも多い．現状 SFMOPT は 2 次元及び 3 次元のタスク配置しかできず，これを多次元に拡張できるようにすることも課題である．

謝　辞

MOPT を考案し，そのプログラムを提供して下さった佐野伸太郎さんに感謝いたします．

今回の投稿に対し，丁寧な修正案を頂いた査読者の方々に感謝いたします．

本研究の一部は，科学技術振興機構・戦略的創造研究推進事業 (CREST) の「ディペンダブルネットワークオンチッププラットフォームの構築」の支援による．

参考文献

1) Yokokawa, M., Shoji, F., Uno, A., Kurokawa, M. and Watanabe, T.: The K computer: Japanese next-generation supercomputer development project, *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 371–372 (2011).
2) Bland, A. S., Kendall, R. A., Kothe, D. B., Rogers, J. H. and Shipman, G. M.: Jaguar: The world's most powerful computer, *Cray User Group* (2009).
3) 佐野伸太郎, 五十嵐俊哉, 吉瀬謙二: メッシュ/トーラス接続型スーパコンピュータに適した高性能タスク配置手法, 電子通信情報学会論文誌 D, Vol. J96-D, No. 2, pp. 269–279 (2013).
4) 柴村英智, 薄田竜太郎, 平尾智也, 吉田真, 神戸隆行, 三輪英樹, 三吉郁夫, 井上弘士, 村上和彰: クラウド環境による OpenNSIM インターコネクトシミュレーションサービス, 情報処理学会研究報告 2010-ARC-192(15), pp. 1–9 (2010).
5) 今出広明, 平本新哉, 三浦健一, 住元真司: 大規模計算環境のためのランク配置最適化手法 RMATT, 先進的計算基盤システムシンポジウム SACSIS2011 論文集, pp. 340–347 (2011).
6) 今出広明, 平本新哉, 三浦健一, 住元真司, 黒川原佳, 横川三津夫, 渡邊貞: 大規模計算向け通信時間最適化ツール RMATT における実行時間の高速化, 2012 年ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, pp. 93–100 (2012).
7) Bhanot, G., Gara, A., Heidelberger, P., Lawless, E., Sexton, J. and Walkup, R.: Optimizing task layout on the Blue Gene/L supercomputer, *IBM Journal of Research and Development*, Vol. 49, No. 2.3, pp. 489–500 (2005).
8) 森江善之, 南里豪志: 多次元メッシュ/トーラスにおける通信衝突を考慮したタスク配置最適化技術, 2013 年ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, pp. 95–103 (2013).