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

森江 善之[1,2,a)]　南里 豪志[1,2,b)]

概要：本稿では，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の提案を行った．このタスク配置最適化では既存技術と異なり，通信の実行される時間帯から通信衝突の発生を予測し，通信衝突を削減するタスク配置を探索，出力することができる．この技術を用いることより既存技術に対してさらに通信性能を向上させることが可能となる．6 次元メッシュ/トーラスをネットワークトポロジとする「京」互換機である FX10 に提案手法を適用し，既存技術であるメッセージサイズとホップ数によるタスク配置最適化技術やネットワークの律速点を調べるタスク配置最適化技術との比較を行う性能評価実験を実施した．このとき，メッセージサイズとホップ数のみによるタスク配置最適化技術に対して最大で約 43%, 律速点を調べるタスク配置最適化技術に対して最大で約 79%の性能向上を示し，提案した技術の有効性を示した．また，同期を挿入して同時に転送開始する通信の集合を明確化することで，通信性能が最大で 35%向上することを示した．さらに，提案したタスク配置最適化技術におけるタスク配置求解の実行時間は 96 ノードで 272sec となり，他の技術に比べても実用に足るということがわかった．

キーワード：タスク配置最適化，通信衝突，多次元メッシュ/トーラス

Task Allocation Technique for Avoiding Contentions on Multi-Dimensional Mesh/Torus

Yoshiyuki Morie[1,2,a)]　Takeshi Nanri[1,2,b)]

Abstract: This paper proposed the task allocation technique for avoiding contentions on a multi-dimensional Mesh/Torus. This task allocation technique, unlike the existing technique, can predict contentions from the period of communication time and can search and output the task allocation that reduces contentions. This technique improves the better communication performance than the existing technique. In this paper, the performance evaluation experiment performs comparison with the technique using message size and the number of hops and the technique checking the bottleneck that are the existing technique in FX10 that is a "Kei computer" compatible machine that has a 6-dimensional Mesh/Torus network topology. In the experiment, the maximum performance gain about 43% over the technique using message and the number of hops and about 79% over technique checking the bottleneck. The effectiveness of the proposed technique was shown in this experiment. Moreover, Clarifying the set of the communications that execute to start transmission simultaneously by inserting synchronizations improved the maximum performance gain 35 %. Moreover, the execution time of searching the solution of the task allocation in the proposed technique was 272 sec at 96 nodes, this is a high speed comparatively for practical use even if compared with existing technique.

Keywords: Task allocation technique, Contentions, Multi-dimenstional Mesh/Torus

1. はじめに

近年，先端科学技術計算からの要求で並列計算機への性

[1] 九州大学情報基盤研究開発センター，福岡市東区箱崎 6-10-1
[2] 独立行政法人科学技術振興機構，CREST，東京都千代田区三番町 5
[a)] morie.yoshiyuki.404@m.kyushu-u.ac.jp
[b)] nanri@cc.kyushu-u.ac.jp

能要件が厳しさを増している．これに対応するため，多くの計算センターでは，大規模な分散メモリ型並列計算機の導入が進められている．例えば，「京」は，約８万個の計算ノードが結合されている．このような大規模並列計算機では，ネットワークトポロジを以前のようにクロスバ網とすることはハードウェアコストの問題から困難となっており，ファットツリーやメッシュ/トーラスといったネットワークトポロジが採用されている．「京」でも採用された多次元メッシュ/トーラスはネットワークの直径を削減するなどの特性を持ち次世代のスーパーコンピュータでの利用も期待されている．

多次元メッシュ/トーラスでは，計算ノード間のリンクを共有することでクロスバ網よりもリンク数やスイッチ数などを削減することが可能であり，ハードウェアコストの面で有利である．しかし，リンクを共有していることから，通信衝突を発生させ，通信性能を十分に悪化させる可能性がある．このため，このようなネットワークトポロジに起因する通信衝突による通信時間の増大を緩和することが重要となる．

筆者ら [4][3][5] は，ネットワークトポロジに起因する通信衝突がタスク配置に依存していることに注目してタスク配置最適化による通信性能の向上を図る研究を行っている．この研究では，既存のタスク配置最適化と異なり，ある時間帯で実行される通信が通過するリンクを調べ，各時間帯における通信衝突を検出するというより詳細なタスク配置最適化技術の提案を行っている．

そこで，本稿では計算機がより大規模となってくる中で，その重要性がより増してくると考えられる多次元メッシュ/トーラスに通信衝突を考慮したタスク配置最適化技術を適用し，その通信性能の評価を「京」互換機である FX10 において実施する．このとき，既存技術である TAHB や RMATT との性能比較を行い，多次元メッシュ/トーラスという通信ホップの影響が大きいネットワークトポロジにおいても提案するタスク配置最適化技術が有効であることを示す．

また，提案技術で重要となる通信の実行される時間帯はプログラム実行時間中に変化し得る．このため，予測と異なる通信衝突を発生させる可能性がある．この通信の実行される時間帯をそろえるため，同期を挿入することを検討している．今回は同期の有無による通信時間の変化を性能評価実験において計測し，考察を行った．

本稿の第 2 章では，通信性能の向上のためのタスク配置最適化技術の関連研究の紹介を行う．次に，第 3 章において詳細な通信衝突の予測が重要であることを示す例の紹介を行う．次に，第 4 章では，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の提案を行い，第 5 章では，提案したタスク配置最適化の有効性を示すために実施した性能評価実験について述べる．第 6 章にて，通信衝突を考慮したタスク配置最適化技術に関する考察を行う．最後に第 7 章でまとめと今後の課題について述べる．

2. 関連研究

本章では，通信性能の向上を目的としたタスク配置最適化の関連研究について述べる．

C.D. Sudheer ら [12] や T. Hoefler ら [11] はリングやファットツリーといったネットワークトポロジにおいて，いくつかの通信パターンが実行され，通信衝突が発生する際のタスク配置が通信性能に与える影響について調査を行っている．そこで，彼らはルーティングやネットワークトポロジなどの詳細な情報を考慮することがタスク配置を行うにあたって重要であることを示している．

T. Hatazaki[7] や J. L. Traff[8] は上位リンクが下位リンクに対して相対的に低速になる階層型のネットワークを対象としたタスク配置最適化技術を提案している．頻繁に通信を行うタスク同士が下位の高速なネットワークを使うように配置する．これにより上位の低速なリンクを使うことが減り，通信時間の削減が行われる．また，F. Ercal ら [6] はリンクレイテンシを削減するためのタスク配置最適化技術を提案した．この技術はハイパーキューブを対象としている．このネットワークトポロジでは，直接接続していない計算ノードが存在する．このため，これらの計算ノード間で通信を行う際は，複数の計算ノードを経由する通信を行わなければならない．このことはリンクレイテンシを増加させ，通信時間を増加させる．このリンクレイテンシを削減するため，各通信が多数の計算ノードを経由しないようにタスクを割り付けるタスク配置最適化を行う．このタスク配置最適化により，リンクレイテンシが減少し，通信時間の削減が行われる．しかし，これらの研究では，通信の衝突の影響は考慮に入れられていない．

T. Agarwal ら [9] や G. Bhanot ら [10] は 3D メッシュや 3D トーラスを対象としたタスク配置最適化技術を提案している．通信衝突を削減するため，各通信のメッセージサイズとホップ数の積の総和が小さくなるようにタスクを計算ノードに割り付ける．このようなタスク配置最適化を Task Allocation by using HopByte (TAHB) と呼ぶこととする．TAHB の目的関数は全ての計算ノード間のホップバイトの総和となる．ホップバイトとは計算ノード間の通信量とホップ数の積のことである．このため，この目的関数を最小とするタスク配置では，通信量の多いタスク同士をホップ数が少なくなるような計算ノード同士に割り付けることになる．このタスク配置最適化では，複数のリンクを経由する通信量が少なくなるので，通信衝突が発生する可能性が低くなり通信性能の向上がなされる．また，今出ら [15][16] は，大規模並列計算環境のためのランク配置最適化ツール Rank Map Automatic Tuning Tool (RMATT)

の提案を行っている．RMATT は TAHB と同様の考え方から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パターンを用いて行うものである．対象とするネットワークトポロジは 3D トーラスである．目的関数は，ホップバイトと通信パターンの律速点における通信量を積算した値を用いている．しかし，これらの研究では，通信衝突において重要となる通信の実行される時間帯について考慮がなされていない．

著者ら [4][3] は，同時に同一リンクを同一方向へ通過するメッセージの個数を調べて，タスク配置最適化を行う研究を行っている．このタスク配置最適化では各リンクでの通信衝突の発生を場所だけでなく時刻も含めて予測することで，通信衝突を削減するタスク配置を決定することができる．性能評価実験ではネットワークがツリートポロジ，ファットツリーの並列計算機を用いて通信性能が向上することを示した．しかし，多次元メッシュ/トーラスのようなネットワークの直径が大きいネットワークトポロジにおける評価は行っていない．

3. 詳細に通信衝突を考慮することによる通信性能に対する効果

ここで，ホップ数とメッセージ数の積を調べるだけでは通信性能を向上することができない例を示す．ここでは，簡単のためにネットワークをツリートポロジとする 6 ノードの並列計算機へタスクを割り付ける場合を用いて説明する．次に示すような通信パターンを実行するプログラムがあるとする．まず，はじめのステップでタスク 1 とタスク 2 の間で通信を行い，次のステップでタスク 4 とタスク 5 の間で通信を行う．さらに，次のステップでタスク 1 とタスク 3，タスク 4 とタスク 6 の間で通信を行い，最後のステップでタスク 1 とタスク 6，タスク 2 とタスク 4，タスク 3 とタスク 5 の間で通信を行う．通信は各ステップになってはじめて実行されるものとする．

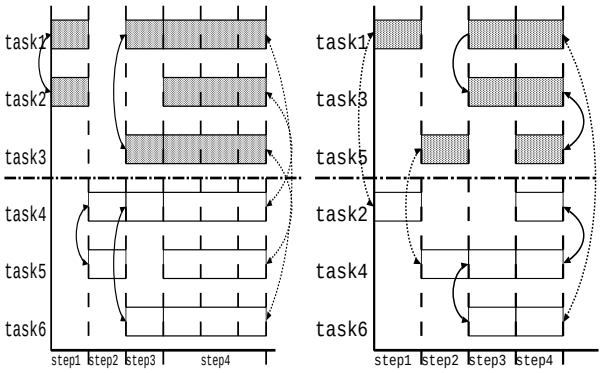

図 1　タスク配置 1 とタスク配置 2 の時の通信の様子．（左）タスク配置 1，（右）タスク配置 2

このような通信パターンを持つプログラムのタスクを対象の計算機に割り付ける．この並列計算機のネットワークは 2 段のツリーとなっており，下位のスイッチ A, B と上位スイッチ C で構成されている．下位のスイッチ A, B にはそれぞれ 3 つの計算ノードを接続されており，スイッチ A，B をスイッチ C を用いて接続している．このような並列計算機のスイッチ A に接続されている計算ノードにタスク 1，2，3，スイッチ B に接続されている計算ノードにタスク 4，5，6 を割り付けるタスク配置をタスク配置 1 とする．このタスク配置の場合，図 3 の左の図で示すように第 4 ステップにおいて 3 つの通信が同時にスイッチ C を通過するため，通信衝突が発生し，実行時間が増大する．

次に，通信の開始する時間帯を考慮に入れ，通信の衝突が起きないようにスイッチ A にタスク 1，3，5，スイッチ B にタスク 2，4，6 を割り付ける．このタスク配置をタスク配置 2 とする．このタスク配置では図 3 の右の図で示すように各ステップでスイッチ C を通過する通信が分散し，通信衝突を発生させない．このため，タスク配置 1 に比べ，タスク配置 2 ではより高速に通信を行うことができる．

タスク配置 1 とタスク配置 2 はどちらもスイッチを通過する通信の数が 3 であるので，メッセージサイズが同じであれば，TAHB の目的関数では同一コストであるとみなされる．このことから通信衝突を回避するタスク配置最適化では，より詳細な情報からタスク配置を行う必要があると考えられる．

4. 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

本章では，前章で述べた問題点を解決する多次元メッシュ/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について述べる．

4.1 通信衝突を考慮したタスク配置最適化技術の概要

通信衝突は，複数の通信が同時に同じリンクを使用することにより発生する．通信に順序関係がある場合などは，各通信により通信開始時刻が異なるため，通信衝突によるペナルティを見積もることはメッセージサイズやホップ数という情報だけでは困難となる．これらを考慮するためには，通信がどのリンクを通過するかと各通信のデータ転送がどの時間帯で行われるかというより詳細な情報を知る必要がある．通信がどのリンクを通るかは通信の送信元と宛先，ネットワークトポロジとルーティングアルゴリズムから得られる．また，各通信のデータ転送がどの時間帯で行われるかは，同時にデータ転送が開始される通信の集合を与えることで得られる．これらの情報を与えて，通信衝突をより正確に調べ，通信時間を予測し，通信時間が削減されるタスク配置の求解を行う．このようなタスク配置最適化技術を Task Allocation with Consideration Concurrent Communication (TAC3) と呼ぶ．

4.2 対象プログラム

TAC3 が適用対象とするプログラムは，繰り返し実行される中核となるコードがあり，そのコード内で通信の組み合わせが不変であるものとする．中核となるコードのことを以下カーネルコードと呼ぶ．このことからプログラム実行前にカーネルコード内の通信のみを最適化の対象とすることができる．

なお，現在のところ提案技術では，カーネルコード内の通信は全て同一メッセージサイズであることを想定している．これは，通信コストの計算が簡単となることと多くの科学技術計算では各計算ノードに等しい負荷を与えることが多く，その間で行う通信のデータ量も等しくなることが多いからである．なお，メッセージサイズが異なる場合への対応については今後の課題である．

4.3 Concurrent Communication Set

提案技術では，同時に発生する通信間での使用リンクの競合を予測する．そこで，同時に転送を開始する通信の集合 Concurrent Communication Set(CCS) を定義する．同時に通信のデータ転送が開始されるための必要条件は次の 2 つとなる．一つは，受信元・宛先がどちらも異なる通信であること，もう一つは，ある通信が完了して初めて実行される通信という順序関係でないことである．CCS はこの 2 つの条件を満たす通信の集合であるとする．

この CCS は，カーネルコードを 1 度実行した通信ログから抽出するものとする．また，CCS は同じプログラムに同じ入力を与えられた場合は同じ結果となるようにする．CCS を生成する方法はいくつか考えられるが，本稿では図 2 に示す手順で CCS を生成する．この CCS の生成アルゴリズムを利用することで，プログラマの負担を減らすことが可能となる．

この CCS をもとに通信衝突を予測してタスク配置を決定しても，実際には，各 CCS の境界をこえて，異なる CCS に所属する通信間による予想外の通信衝突は発生する可能性がある．一方，各 CCS の冒頭で同期をとれば，予想外の衝突が発生しないが，CCS 毎の待ち時間や同期のコストが発生する．これらのトレードオフについては，実験で検証する．

4.4 目的関数

本提案では文献 [1][2][9] と同様に，タスク間通信とネットワークトポロジをそれぞれグラフ $G_t = \{V_t, E_t\}$, $G_n = \{V_n, E_n\}$ で表し，2 つのグラフ間の節点同士の一対一写像 P の最適化としてタスク配置最適化問題を定義する．ただし，$G_t = \{V_t, E_t\}$ は，各通信がどのリンクで要求されるかと各通信のデータ転送がどの時間帯で実行されるかを考慮するため，E_t の定義に変更を加える．グラフは有向グラフとし，辺 $e_{ab} = (v_a, v_b) \in E_t$ は，タスク a からタスク b への独立した個々の通信を表す．また，その通信の ID である cid，通信のデータ量や CCS の ID を重みとして持っているとする．

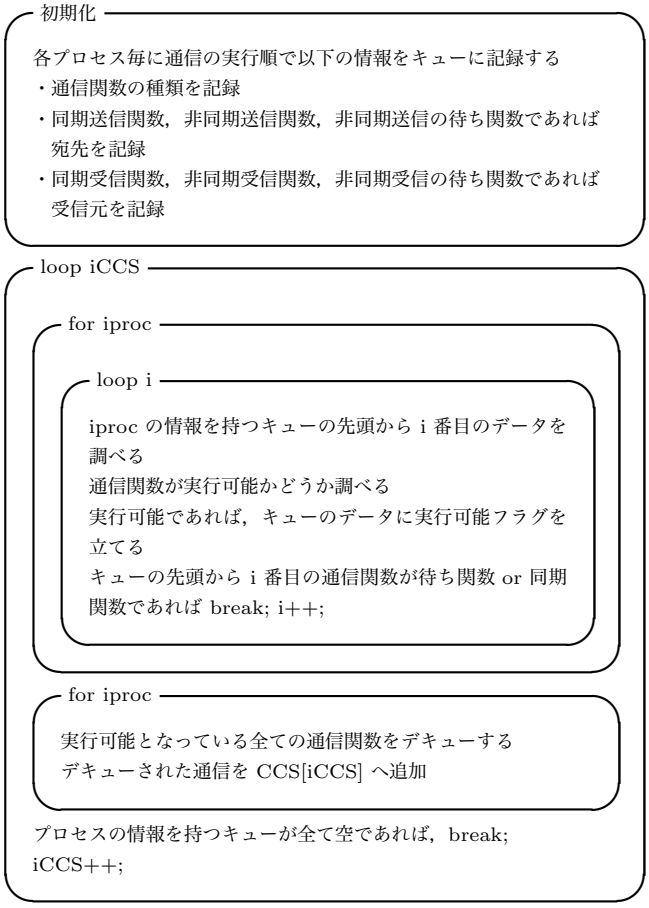

図 2　CCS の生成アルゴリズム

TAC3 の目的関数では，対象並列計算機上で実行される対象プログラムの通信時間の見積もりを行う．まず，各 CCS における各タスクの予想通信時間を求め，その最大値を得る．ここで得た各 CCS 間の予想通信時間の最大値を積算する．これにより，各 CCS における通信衝突による通信時間の増加を加味する．目的関数の計算式は式 (1) となる．

$$f(V_t, V_n, P) = \sum_{t=0}^{N-1} \max_{i=0\cdots n-1} M(t, i, tork(t, i, cid)) \times coll(t, \pi(i), \pi(tork(t, i, cid)))/B \quad (1)$$

t は CCS の ID，N は CCS の総数である．i はタスクの ID，n はタスクの総数である．$\pi(i)$ はタスク i が割り付けられている計算ノードの ID を返す関数である．$tork(t, i, cid)$ は t 番目の CCS におけるタスク i の通信 cid の宛先のタスクの ID を返す関数である．B は各リンクの通信帯域幅である．$M(t, i, j)$ は t 番目の CCS におけるタスク i からタスク j への通信のデータ量を返す関数である．また，$coll(t, p, q)$ は t 番目の CCS における計算ノード p から q 間の経路の各リンクで要求される通信数の最大値を返す関数

である．関数 $coll$ では，以下の手順で値の導出を行う．まず，あらかじめ，第 t 番目の各通信ごとに通過するリンクをルーティングアルゴリズムから調べ，各リンクごとに要求される通信の個数を積算しておく．次に，計算ノード p から q へデータを転送する際に通過するリンクをルーティングアルゴリズムから求め，これらのリンクで要求される通信の個数を比較することでその最大値を得る．この目的関数の値を最小とするタスク配置を求め，適用することで通信性能向上を図る．この目的関数により，初期タスク配置から予測通信時間を求め，タスク配置最適化にかけることのできる時間を決定することも可能である．

本提案では，著者らの既存研究におけるツリー構造を対象としたタスク配置最適化と違い，ルーティングを考慮して目的関数の値を計算している．これは本提案で対象としている多次元メッシュ/トーラスにおいて計算ノード間の経路がマルチパスになっているからである．このため，タスク配置最適化のマッピング対象となる並列計算機のルーティングを比較的詳細に把握する必要がある．例えば，多次元メッシュ/トーラスでは，一般に次元オーダルーティング (DOR) が用いられる．この場合，アルゴリズムだけでなく，各軸をどの順序で選択するかなどの情報が必要となる．これは，同じ DOR でも軸を選択する順序が異なれば，通過するリンクが異なり，各リンクで要求される通信数が対象計算機と異なってしまうからである．また，他にもルーティングが静的である必要がある．本技術では，通信がどのリンクを経由するかを事前に確定する必要があるからである．

4.5 タスク配置の求解アルゴリズム

タスク配置最適化問題は NP 完全であることが知られている [9]．したがって，現実的な時間で最適なタスク配置を求解することは困難であると考えられる．しかし，探索時間があまりに長時間にわたれば，タスク配置最適化による性能向上の利得が失われてしまうので，タスク配置求解にかかる実行時間は比較的小さいものである必要がある．このため，提案したタスク配置最適化の解法として発見的手法であるシミュレーティッドアニーリング [17] を用いる．シミュレーティッドアニーリングは終了条件を変更することによって，比較的短時間で探索を終了させる設定を行うことができるため，本タスク配置最適化の求解アルゴリズムとしてふさわしいと考える．本提案技術では，終了条件として終了温度を設定することとしており，初期温度，温度スケジュール，各温度でのタスク配置評価試行回数からタスク配置評価の試行回数を一意に決める方法を取る．長時間の探索を許さない状況では，タスク配置試行回数を短縮させる設定を行うこととする．

5. 性能評価実験

本稿にて提案した TAC3 の有効性を示すため，TAHB や RMATT を適用した場合との比較を行う性能評価実験を実施する．

5.1 実験概要

FX10 において TAC3 と TAHB や RMATT を適用した CG 法のカーネルコードを抽出したプログラムを実行して，そのプログラムの実行時間を計測し，比較を行う．このとき，TAC3 は CCS の集合とメッセージサイズを情報として与えるタスク配置最適化を行い，タスク配置を出力させる．また，TAHB 及び RMATT では，通信の宛先，送信元，メッセージサイズを情報として与えタスク配置最適化を行い，タスク配置を出力させる．これらのタスク配置の探索を行う際，RMATT 以外の探索パラメタは，初期温度 $1.0 + e1$，終了温度 $1.0 - e8$, 同一温度での繰り返し回数 2500，温度スケジュール係数 0.9 とした．このタスク配置の探索において，全て同じパラメタを用いたのは，同じ試行回数で通信性能に対してどれほど差が生じるかを調査するためである．一方，RMATT では，10 分以上，変化率が 0.1 を越えなければ終了するというデフォルトの終了条件を設定を採用した．このため，探索時間がタスク配置最適化により異なることになる．同一探索時間における各タスク配置最適化の性能評価は今後の課題である．

また，TAC3 では，異なる CCS に所属する通信間で通信衝突が発生しないとして目的関数を評価しているが，実際の実行においては異なる CCS に所属する通信間において通信衝突が発生する場合がある．そこで，TAC3 のタスク配置を適用したプログラムの各 CCS の先頭で同期をする場合 (TAC3 barrier) と同期をしない場合 (TAC3 no-barrier) の 2 通りプログラムを用意し，CCS を明確化することによる影響を調査するため，実行時間の比較を行った．

5.2 実験環境

実験環境として FX10 を用いており，ここで，その仕様を示す．CPU は Fujitsu SPARC64TM IXfx 1.848GHz (16 コア)，メモリは 32GB，計算ノード数は 768 ノード，OS は Linux ベース独自 OS，コンパイラ，MPI ライブラリは Fujitsu Technical Computing Suite v1.0 である．ネットワークは，Tofu interconnect[14] を用いており，このネットワークの各リンクの理論帯域は 5GB/s となる．この Tofu interconnect のネットワークトポロジは XYZABC 軸からなる 6 次元メッシュ/トーラスで各軸のサイズは 4x2x8x2x3x2 である．また，ルーティングアルゴリズムは拡張次元オーダルーティングである．今回の実験では故障計算ノードを用いないようにしたため，次元オーダルー

ティングと同様のルーティングを行う．本実験環境でのルーティングの決定順序は X 軸，Y 軸，Z 軸，A 軸，C 軸，B 軸となっている [19]．

また，本実験では，ノード形状の違いによる性能を比較するため，各ノード数において複数の形状による実験を行った．FX10 のネットワークは図 3 で示すように大きさ 2x3x2 の ABC 軸の 3 次元メッシュ/トーラスを基本単位とし，その基本単位を XYZ 軸の 3 次元のメッシュ/トーラスで結合した構造となる．このため，基本単位である 12 ノードの倍数の計算ノードが割り付けられる．今回使用した九州大学の FX10 では，一般ユーザは最大で 96 個の計算ノードを使用できるため，1x2x4x2x3x2 (形状 1)，2x2x2x2x3x2 (形状 2)，1x1x4x2x3x2(形状 3)，1x2x2x2x3x2 (形状 4) の 4 種類の形状で実験を行った．形状 1，形状 2 は 96 ノード，形状 3, 形状 4 は 48 ノードを用いる実験となる．

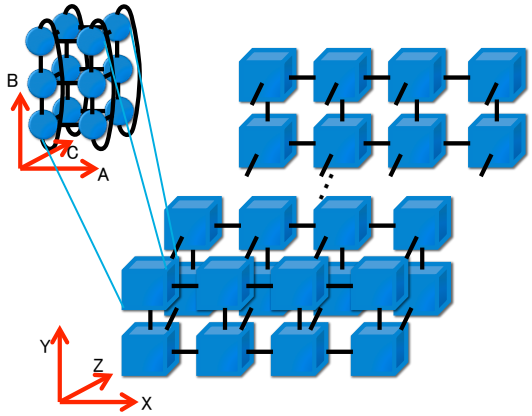

図 3　FX10 のネットワークトポロジ

また，FX10 は，複数の研究者によって共有されているため，計算機資源を有効に使用できるよう空いている計算ノード群に自動的にジョブがスケジュールされる．このとき，論理 3 次元では，形状を指定することは可能であるが，6 次元で形状を指定することはできない．このため，実行時に割り当てられるまで 6 次元の形状がどのようになっているか分からない．そこで，実タスクに仮想タスクを与えることで，各タスクの仕事内容を交換する仮想的なタスクの再配置を行うこととした．

5.3　ベンチマークプログラム

今回は通信性能のみを比べるため，NAS Parallel Benchmarks[18] の CG 法（Conjugate Gradient method）のカーネルコードの通信関数に関連する部分を抽出した．この抽出したカーネルコードを 1000 回ループするプログラムを作成し，本実験におけるベンチマークプログラムとした．このとき，メッセージサイズは自由に設定できるようにしている．本実験では，通信衝突の影響を調査するのが目的であるため，通信衝突の影響が明確となるメッセージサイズとして 1MB を採用した．

5.4　比較タスク配置群

本実験では，デフォルトタスク配置，no-ring，TAHB，RMATT-O2F，RMATT，TAC3 の計 6 つのタスク配置による通信性能比較を行う．

まず，性能の基準とするため，デフォルトタスク配置，no-ring を用意する．デフォルトタスク配置は FX10 で標準で与えられるタスク配置である．ここでは，6 次元メッシュ/トーラスを論理 3 次元トーラスに見せるためのタスク配置が行われている．no-ring は 6 次元の XYZABC 軸の順でタスクを割り付けたタスク配置である．このタスク配置では，論理 3 次元で見たとき，3 次元の各軸のタスクはリング状に並ばない．

次に，TAHB，RMATT-O2F は，目的関数以外は TAC3 と同一の探索アルゴリズム，同一の条件下でタスク配置最適化を実施し，出力されたタスク配置である．このとき，用いた目的関数は，それぞれ以下のようになっている．まず，TAHB の目的関数は式 (2) で与えられる．

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} hop(\pi(i),\pi(j)) \cdot size(i,j) \qquad (2)$$

i, j はタスクの ID，n はタスクの総数である．$\pi(i)$ はタスク i が割り付けられている計算ノードの ID を返す関数である．$size(i,j)$ はタスク i からタスク j への通信量を返す関数である．また，$hop(p,q)$ は計算ノード p から計算ノード q へのホップ数を返す関数である．次に，RMATT-O2F の目的関数は式 (3) で与えられる．

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} (hop(\pi(i),\pi(j)) \cdot size(i,j)) \times max_{link} \qquad (3)$$

i, j はタスクの ID，n はタスクの総数である．$\pi(i)$ はタスク i が割り付けられている計算ノードの ID を返す関数である．$size(i,j)$ はタスク i からタスク j への通信量を返す関数である．また，$hop(p,q)$ は計算ノード p から計算ノード q へのホップ数を返す関数である．$link$ はあるノード間に流れた転送量を示し，max_{link} は全ノード間における link の最大値である．

最後に RMATT は，FX10 にて提供されているランク配置自動最適化ツール RMATT を用いて出力されたタスク配置である．

このとき，TAHB，RMATT-O2F，RMATT，TAC3 は，タスク求解アルゴリズムとしてヒューリスティックであるシミュレーティッドアニーリングを用いているため，それぞれの解法において 10 個ずつのタスク配置を生成し，プログラムを実行することとした．このとき，10 個のタスク配置による平均の実行時間の比較を行った．

5.5 実験結果

まず，各ノード数における CCS 数について述べる．CG 法では，タスクを 2 次元格子状に並べ，行方向で Recursive doubling の通信を行った後，2 次元格子の転置通信を行う．48 ノードでは，32 個のタスクによる通信が行われる．このとき，8x4 の格子にタスクが並べられる．Recursive doubling では，8 個のタスクで 3 ステップで通信を完了するため，3 つの CCS に分けられる．転置通信は，通信が一斉になされるため，CCS は 1 つとなる．このため，全体で，CCS 数は 4 となる．また，96 ノードでは，64 個のタスクによる通信が行われるため，8x8 の格子にタスクが並べられ通信が実行される．このとき，行に割り当てられるタスク数が 48 ノードと同じであるため，CCS 数も同一の 4 となる．

次に，各タスク配置のプログラムの実行時間を表 1 に示す．まず，性能の基本となるデフォルトタスク配置では，実行時間がそれぞれ形状 1 で約 1.7sec，形状 2，3 で約 1.2sec，形状 4 で約 1.4sec となった．また，no-ring では，形状 1 で約 2.4sec，形状 2 で約 1.9sec，形状 3 で約 1.7sec，形状 4 で約 1.5sec となった．次に，TAC3 では，形状 1 以外において実行時間が約 1.0sec となった．この時，TAC3 同期ありは，TAC3 同期なしに比べ，14msec から 17msec 増加している．一方，形状 1 の時，TAC3 同期ありは TAC3 同期なしに対して平均で 81msec 性能が向上している．次に，TAHB，RMATT-O2F では，同じような傾向となり，実行時間は，形状によらず，96 ノードでは約 1.5sec，48 ノードでは約 1.3sec となった．最後に RMATT では，TAHB や RMATT-O2F より 96 ノードで，0.2 から 0.4sec 程度，48 ノードでは，0.1sec 程度，通信性能が悪化した．

また，図 4 にデフォルトタスク配置に対する各タスク配置における性能比を示す．縦軸には性能比，横軸は各タスク配置の項目が上げられている．ここでの性能比は，デフォルトタスク配置の実行時間を各タスク配置における実行時間で割った値を用いており，1.0 を超えるとデフォルトタスク配置より通信性能が改善していることを示している．本実験環境において，TAC3 は通信性能がデフォルトタスク配置に対して，約 48% の性能向上を示した．また，TAC3 は他のすべてのタスク配置より高速となった．特に注目すべきは既存技術である TAHB や RMATT に対して最大でそれぞれ約 43%, 約 79%の性能向上を示したことで既存技術に対して十分に性能向上をさせることができることがわかった．

また，TAC3 において各 CCS の先頭にて同期した場合は形状 1 の場合のみ性能が向上した．形状 1 以外では，TAC3 が全く通信衝突を発生させないタスク配置を生成したため，各 CCS で遅れを生じる通信が存在しなかった．このため，挿入された同期通信が単なるオーバヘッドとして現れ，通信性能が同期通信の分だけ悪化することになったと考えられる．

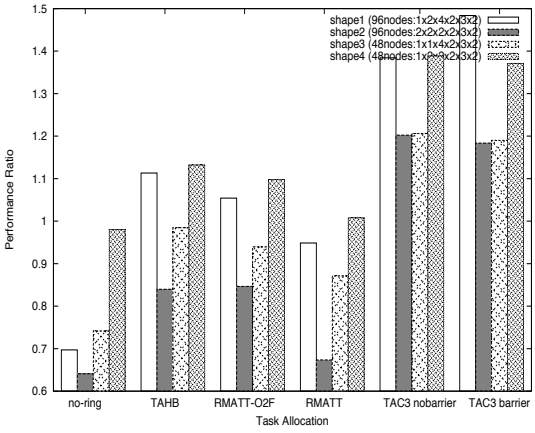

図 4 デフォルトタスク配置に対する各タスク配置の性能比

また，TAHB，RMATT-O2F, RMATT でいずれも no-ring に対しては性能向上したが，デフォルトタスク配置に対してはほとんど同等の性能となった．これはデフォルトタスク配置が 3 次元の各軸において ID が連続するタスクをリング状に並べる配置であることから，通信衝突を発生させにくくなり，通信量とホップ数をもとに探索されたタスク配置と比べて大きく差が生じなかったからだと考えられる．

最後に，RMATT は RMATT-O2F に対して 1 割から 2 割ほど通信性能が悪化した．これは RMATT が，3 次元トーラスを対象としており，6 次元メッシュ/トーラスを考慮できてないからだと考えられる．FX10 では，ネットワークを論理 3 次元トーラスとして表すことは可能であるが，2 ホップ以上の通信が実行される場合は，3 次元トーラスとして動作することを保証していない．一方，RMATT-O2F は目的関数のみ RMATT と同一のものでネットワークトポロジは FX10 の 6 次元メッシュ/トーラスであることを仮定して動作しているため，RMATT ほどの性能低下が起こらなかったと考えられる．

6. 考察

6.1 各 CCS の先頭における同期の有無による通信性能

TAC3 で出力されたタスク配置は形状 1 でもほとんど通信衝突の発生しないタスク配置を生成すること出来ている．これは，CG 法の通信パターンでは，2 の冪乗個のタスクのみが通信を行う．今回の実験では 96 ノードの場合は 64 ノード，48 ノードの場合は 32 ノードしか通信を行わないため，ホップ数を気にしない TAC3 では，通信衝突を全く発生させないタスク配置を探索できたものと思われる．

しかし，これでは，各 CCS の先頭に同期を挿入した効

表 1　各タスク配置における実行時間

	default	no-ring	TAHB	RMATT-O2F	RMATT	TAC3 同期なし	TAC3 同期あり
形状 1(96 ノード)	1.671sec	2.398sec	1.501sec	1.585sec	1.762sec	1.207sec	1.126sec
形状 2(96 ノード)	1.241sec	1.936sec	1.478sec	1.466sec	1.843sec	1.032sec	1.049sec
形状 3(48 ノード)	1.236sec	1.667sec	1.255sec	1.316sec	1.419sec	1.025sec	1.039sec
形状 4(48 ノード)	1.427sec	1.455sec	1.260sec	1.300sec	1.415sec	1.027sec	1.041sec

表 2　TAC3 で出力された各タスク配置において CCS の先頭に同期を挿入した場合と同期を挿入しなかった場合の実行時間

	同期なし (sec)	同期あり (sec)
1	1.236	1.263
2	1.038	1.050
3	1.664	1.264
4	1.452	1.264
5	1.032	1.046
6	1.031	1.049
7	1.033	1.049
8	1.034	1.048
9	1.703	1.262
10	1.032	1.048

果を見ることが出来ない．このため，ここでは，通信衝突を発生させたタスク配置に限定して同期の効果がどれほどあったかを見る．TAC3 において出力された 10 個のタスク配置において各 CCS の先頭に同期を挿入した場合と挿入しなかった場合の実行時間を表 2 に示す．この表において実行時間が約 1.0sec となる場合は，通信衝突のない最適なタスク配置である．つまり，この表から 1, 3, 4, 9 の 4 つのタスク配置において通信衝突が発生したことがわかる．このうち，3, 4, 9 の 3 つのタスク配置では，同期を挿入しなければ通信性能が悪化していることが分かる．特に 9 のタスク配置においては同期を挿入することで約 35%の性能向上を示している．一方，1 のタスク配置は，最終 CCS において通信衝突が発生したため，それより後続の通信がなく，想定していない通信衝突が発生しなかったと考えられる．そのため，1 のタスク配置では，同期のコストが加わる分だけ，通信性能が悪化したと考えられる．ノード数が大規模化する中，全く通信衝突の発生しないタスク配置を常に生成することは困難であることから異なる CCS に所属する通信間で発生する通信衝突を同期により防ぐことは重要であることがわかる．

そこで，同期を挿入しても通信性能が向上する CG 法のメッセージサイズについて調査を行った．表 3 に TAC3 において各 CCS の先頭に同期を挿入した場合と同期を挿入しない場合の各メッセージサイズにおける実行時間と同期なしに対する同期ありの性能比を示す．ここでは，異なる CCS 間に所属する通信間で通信衝突を発生させるタスク配置を 1 つ選んで用いた．この表からメッセージサイズが 256KB までは同期通信を挿入しない方が通信性能が高いことがわかる．メッセージサイズが 512KB を超えると同期通信を挿入する方が通信性能が高くなることが分かる．さらにメッセージサイズが大きくなるごとに性能比が向上している．メッセージサイズが大きくなると各 CCS の先頭に同期を挿入することが重要になると考えられる．

表 3　TAC3 において各 CCS の先頭に同期を挿入した場合と同期を挿入しない場合の各メッセージサイズにおける実行時間

メッセージサイズ (KB)	同期なし (sec)	同期あり (sec)	性能比
32	0.176	0.193	0.910
64	0.202	0.228	0.887
128	0.293	0.307	0.953
256	0.447	0.450	0.994
512	0.826	0.719	1.161
1024	1.540	1.267	1.215
2048	2.971	2.381	1.248

6.2　タスク配置求解の実行時間

今回，タスク配置求解アルゴリズムは RMATT 以外はすべて同じものを用いている．TAC3, RMATT-O2F, TAHB におけるタスク求解にかかる実行時間の差は目的関数評価にかかる時間の違いである．48 ノードの場合は, TAC3 が 128sec，TAHB が 35sec，RMATT-O2F が 122sec であった．96 ノードの場合は，TAC3 が 272sec，TAHB が 72sec，RMATT-O2F が 257sec であった．RMATT はデフォルトでは 10 分間続けて目的関数値の変化率が 0.1 未満となる場合に終了するという条件を持っている．この条件下で RMATT のタスク求解時間は 48 ノードも 96 ノードの場合も 1200sec となっている．今回の実験からノード数が少ない間は TAC3 のような詳細なタスク配置最適化でもタスク求解時間も十分に短いことが分かる．

7.　おわりに

本稿では，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について述べ，その有効性を示すための性能評価実験を行った．「京」互換機である FX10 において TAHB や RMATT に対して最大でそれぞれ約 43%, 約 79% の性能向上を示した．また，同期を用いて CCS を明確化することにより，異なる CCS に所属する通信間における通信衝突が発生しなくなり，通信性能が向上することを示した．さらに，タスク配置求解にかかる時間を求めたが TAHB や RMATT と比べても十分実用に足ることがわかった．これらにより，通信に順序関係がある場合は，通信の時間を考慮した TAC3 のようなタスク配

置最適化が有効であることが分かった．

今後の課題は，以下のものが挙がられる．2 の冪乗の計算ノードしか使用できない中でタスク配置最適化を行うプログラムを生成し，その中でも通信性能が向上することを示す．そのとき，通信衝突の発生が多くなることが考えられ，その中でも CCS を明確化すべく同期を挿入すべきかどうかを調査する．他のベンチマークでの性能評価実験を行う．これは，提案手法の適用範囲を調べるためである．また, 今回手動で行った CCS の自動生成プログラムの開発やシミュレーティッドアニーリングと異なるタスク求解アルゴリズムを試用し，探索時間の比較を行うことなどが課題である．

謝辞　本研究は主に九州大学情報基盤研究開発センターの研究用計算機システムを利用しました．

参考文献

[1] S. H. Bokhari, “On the mapping problem,” IEEE Trans. Computers, 30(3), pp. 207-214, 1981.

[2] S. Y. Lee and J. K. Aggarwal, “A mapping strategy for parallel processing,” IEEE Trans. Computers, 36(4), pp. 433-442, 1987.

[3] Y. Morie, T. Nanri, and M. Kurokawa, “Task allocation method for avoiding contentions by the information of concurrent communication,” in Proceedings of the Tenth IASTED International Conference on Parallel and Distributed Computing and Networks, pp. 62-69, Feb. 2011.

[4] 森江 善之, 末安 直樹, 松本 透, 南里 豪志, 石畑 宏明, 井上 弘士, 村上 和彰, “通信タイミングを考慮した衝突削減のための MPI ランク配置最適化技術,” 情報処理学会論文誌コンピューティングシステム, Vol.48, No.13, pp.192-202, Aug. 2007.

[5] Y. Morie, T. Nanri, “Task Allocation Optimization for Neighboring Communication on Fat-Tree,”in Proceedings of The Fifth International Symposium on Advances of High Performance Computing and Networking (AHPCN-2012), Jun. 2012.

[6] F. Ercal, J. Ramanujan, and P. Sadayappan, “Task allocation onto a hypercube by recursive Mincut Bipartitioning,” Journal of Parallel and Distributed Computing, Vol. 10, N. 1, pp. 35-44, 1990.

[7] T. Hatazaki, “Rank reordering strategy for MPI topology creation function,” PVM/MPI’98, LNCS 1497, pp. 188-195, 1998.

[8] J. L. Traff, “Implementing the MPI process topology mechanism,” Conference on High Performance Networking and Computing, in Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, pp. 1-14, 2002.

[9] T. Agarwal, A.Sharma, and L. V. Kale, “Topology-aware task mapping for reducing communication contention on large parallel machines,” in Proceedings of IEEE International Parallel and Distributed Processing Symposium 2006, pp. 1-10, 2006.

[10] G. Bhanot, A. Gara, P. Heidelberger, E. Lawless, J. Sexton, and Robert Walkup, “Optimizing task layout on the Blue Gene/L supercomputer,” IBM J. Res. & Dev. 49, No. 2/3, pp. 489-500, 2005.

[11] T. Hoefler, T. Schneider, and A. Lumsdaine, “Multistage switches are not crossbars: Effects of static routing in high-performance networks,” in Proceedings of the 2008 IEEE International Conference on Cluster Computing, CLUSTER’08, IEEE Computer Society, Oct. 2008.

[12] C.D. Sudheer, T. Nagaraju, P.K. Baruah, and A. Srinivasan, “Optimizing assignment of threads to SPEs of the cell BE processor,” Tenth IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC), in Proceedings of the 23rd International Parallel and Distributed Processing Symposium, IEEE, 2009.

[13] T. Hoefler and J. L. Traeff, “Sparse collective operations for MPI,” in Proceedings of the 23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS), HIPS Workshop, May. 2009.

[14] Y. Ajima , Y. Takagi , T. Inoue , S. Hiramoto and T. Shimizu, “The tofu interconnect,”in Proceedings of the 19th IEEE Annual Symposium High Performanace Interconnects, pp.87–94, 2011.

[15] 今出 広明，平木 新哉，三浦 健一，住元 真司，“大規模並列計算環境のためのランク配置最適化手法 RMATT,”SACSIS2011, pp.340–347, Mar. 2011.

[16] 今出 広明，平木 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,”HPCS2012, pp.340–347, jan. 2012.

[17] 長尾智治，“最適化アルゴリズム,” 昭晃堂，2000.

[18] “NAS Parallel Benchmarks,” 入手先 ⟨http://www.nas.nasa.gov/Resources/Software/npb.html⟩

[19] Y. Ajima, T. Inoue, S. Hiramoto, T. Shimizu, “Tofu: Interconnect for the K computer,”, FUJITSU Sci. Tech. J., Vol. 48, No. 3, pp. 280—285, July 2012.