ネットワーク型マルチ FPGA システムを対象としたタスク割り当て手法

片野 弘規 † 戸川 望 † 青木 孝 †† 関原 悠介 †† 中西 衛 ††

† 早稲田大学大学院基幹理工学研究科情報理工学専攻
†† NTT 先端集積デバイス研究所

近年，ブロードバンドネットワークサービスや計算流体力学，遺伝子解析等の演算を目的としたシミュレーションにマルチ FPGA システムを利用した研究が進められている．中でもネットワーク状に構成されたマルチ FPGA システムは，複数枚のボードから構成され, 各ボードには複数個の FPGA チップとボード間通信用ルータ FPGA チップがリング状に接続されている．アプリケーションや状況に応じてボードを追加することができ, 柔軟性と拡張性に優れた特徴を持っている．このようなマルチ FPGA システムでは, タスクグラフを構成する各タスクのマルチ FPGA システムへの割り当てが, マルチ FPGA システム上で実行されるアプリケーションの性能とコストを決定づける重要な要素となる．本稿ではネットワーク型マルチ FPGA システムを対象としたタスク割り当て手法を提案する．提案するタスク割り当て手法は，各 FPGA チップが持つ演算リソースの制約を考慮して，タスクグラフ中で通信量が最大となるタスクをクラスタリングする．続いてこれらのクラスタリングされたタスクは各ボードが持つ局所性を考慮して，同一のボードに割り当てていくものである．また，提案したタスク割り当て手法の検証と評価を行った結果を報告する．

A Task Mapping Algorithm for Network-based Multi-FPGA Systems

Hiroki KATANO† Nozomu TOGAWA† Takashi AOKI†† Yusuke SEKIHARA†† Mamoru NAKANISHI††

† Dept. of Computer Science and Engineering, Waseda University
†† NTT Device Technology Laboratories

Recently, multi-FPGA systems have been proposed for broadband network serviecs, computational fluid dynamics (CFD) and gene analysis. Among them network-based multi-FPGA systems are composed of two or more boards, each of which consists of one router FPGA chip and multiple general-purpose FPGA chips. The general-purpose FPGA chips are connected to form a ring and the router FPGA chip performs inter-board communications. How to map a task graph onto such a multi-FPGA system is one of the challenging problems. In this paper, we propose a task mapping algorithm for a network-based multi-FPGA system. In our proposed algorithm, we observe the *locality* of board and FPGA resources. We focus on the communication rate between tasks and assign the ones with many communications between them to the same FPGA chip and to the same board. Experimental results demonstrate the effectiveness of our proposed algorithm.

1 はじめに

回路の再構成が可能な FPGA の性能は，近年の半導体製造技術の発展により向上している．また，近年マルチ FPGA システムを利用した研究が進められており，計算流体力学シミュレーション FLOPS-2D [6], 粒子シミュレーション演算を目的とした PROGRAPE [3], 遺伝子解析等に利用される BEE2 [1], ブロードバンドネットワークサービス向けの CoRD システム [7] 等が挙げられる．中でも CoRD システムは NTT 研究所が提案するマルチ FPGA システムであり, ブロードバンドネットワーク上におけるインターネット TV や IPTV, セキュリティ映像監視システム, デジタルサイネージといった様々な映像配信サービス映像を用いた様々なサービスについて, ハイエンドなネットワークサービスを提供することを目的としたシステムである．CoRD システムはネットワーク状に構成されたマルチ FPGA システムであり, 複数枚のボードから構成され, 各ボードには 5 つの FPGA チップと, 1 つのボード間通信用ルータ FPGA チップがリング状に接続されている．ボード上の各 FPGA チップには複数のタスクが実装される．アプリケーションや状況に応じてボードを追加することができ, 柔軟性と拡張性に優れた特徴を持っている．

CoRD システムのようなネットワーク型マルチ FPGA システム上にタスクを割り当てる際, 以下の 3 点を条件として考慮する必要がある．

(1). 複数枚のボードから構成され, 各ボードには N 個の FPGA チップと, 1 つのボード間通信用ルータ FPGA チップがリング状に接続されたアーキテクチャであること．
(2). 各 FPGA チップはタスクの演算リソースに関する制約を持つこと．

各タスクが実装された FPGA チップによって, マルチ FPGA システムの性能とコストを決定づけるため, マルチ FPGA システムへのタスクの割り当てには，まず条件 (1) と (2) が不可欠である．

タスク割り当てを扱った関連研究として, マルチ FPGA システムと同様にタスクをネットワーク状に接続したネットワークオンチップを対象としたタスク割り当て手法が挙げられる [4], [8], [2]．タスク間のホップ数を最適化し消費電力の最小化を図る Onyx アルゴリズム [4] は, 対象アーキテクチャがメッシュ状であり, また各タスクの演算リソースを考えていないという点で条件 (1) と (2) を満足していない．二項木の概念を用いてタスクグラフ中の各タスクをマージさせタスク割り当てを行い, ネットワークの総通信量, ホップ数, ハードウェアコストの最小化を図る BMAP アルゴリズム [8] も同様に, 対象アーキテクチャがメッシュ状であり, 各タスクの演算リソースを考えていないという点で条件 (1) と (2) を満足していない．タスクグラフ中の各タスクの面積を考慮してタスク割り当てする AUBM アルゴリズム [2] は, 対象アーキテクチャがメッシュ状であるという点で条件 (1) を満たさない．また, タスクの面積を考慮するが, メッシュ型アーキテクチャで構成されたチップ自体の面積の最小化を図るものであるという点で条件 (2) も満たさない．以上のように, 著者らの知る限り条件 (1) と (2) を前提条件としたタスク割り当て手法は知られていない．

また条件 (1) と (2) に加えて，各タスクの適当な割り当て結果を高速に得るため，条件 (3) が不可欠である．

(3). 高速にタスク割り当て箇所を決定すること．

以上の議論のもと，我々はマルチ FPGA システム向けのタスク割り当て手法を提案した [5]．提案手法では，各 FPGA チップが持つ局所性を考慮して, タスクグラフ中で通信量が最大となるタスクを中心として周

囲を探索し，これらのタスクはできるだけ同一あるいは隣接した FPGA チップに割り当てていくことで，低コストなタスク割り当てを実現した．しかしながら，対象アプリケーションの規模が大きいとき，すなわちタスク数が多いとき満足のいく結果を得ることができなかった．

本稿では，マルチ FPGA システム上の各 FPGA チップが持つ演算リソースと各ボードが持つ局所性を考慮したタスク割り当て手法を提案する．ボードの持つ局所性を考慮した手法を加え，従来手法と比較を行う．条件 (1)〜(3) を満足するよう，提案するマルチ FPGA システムを対象としたタスク割り当て手法は，各 FPGA チップが持つ演算リソースの制約を考慮して，タスクグラフ中で通信量が最大となるタスクをクラスタリングする．続いてこれらのクラスタリングされたタスクは各ボードが持つ局所性を考慮して，同一のボードに割り当てていくものである．計算機実験の結果，従来手法と比較してコストを最大 55.1% 削減，SA 法と比較してコストを最大 3.71% 削減し，CPU 時間を平均して約 1/17 に抑えることを確認した．

2 問題の定式化

本稿ではアプリケーションのタスクグラフとして，無向グラフを用いる．重み付きタスクグラフとタスク割り当ての対象としたマルチ FPGA システム，FPGA のリソースを定義する．また，マルチ FPGA システムを対象としたタスク割り当て問題を定義する．

2.1 重み付きタスクグラフ

重み付きタスクグラフ $G(V,E)$ のタスクを $v \in V$ と表す．以降，グラフの節点とタスクとを同一視する．タスク間 u と v の枝を $e=(u,v) \in E$ と表す．枝 (u,v) の重み，すなわち通信量を $c(e)$ または $c(u,v)$ と表す．

2.2 ネットワーク型マルチ FPGA システム

本稿では CoRD システムをもとにしたネットワーク型マルチ FPGA システムにおけるタスク割り当てを考察する．複数枚のボードから構成され，各ボードには N 個の FPGA チップと 1 つのボード間通信用ルータ FPGA チップがリング状に接続されているものを想定する．例として，$N=5$ のマルチ FPGA システムを図 1 に示す．各ボードには 5 個の FPGA チップと，ボート間通信用ルータ 1 つがリング状に実装されている．

いくつかのボードが必要になるかはタスク割り当てによって決まる．k 番目のボードを $B(k)$ とかく．また各ボードは，ボード間通信用ルータに近い方から順に FPGA に番号を付け，$N(k-1)+1$ から $N \times k$ を持つ．

タスクグラフ $G(V,E)$ の各タスク $v \in V$ が割り当てられた FPGA を $f(v)$ とする．マルチ FPGA システムへタスクを割り当てた際の評価を，ホップ数と通信量を用いて以下の式で定義する．

$$Cost = \sum_{(v_i,v_j) \in E} hop(f(v_i), f(v_j)) \times c(v_i, v_j) \quad (1)$$

$hop(f(v_i), f(v_j))$ はタスク v_i が割り当てられた FPGA から，タスク v_j が割り当てられた FPGA へのホップ数である．$c(v_i, v_j)$ は，タスクグラフにおけるタスク v_i, v_j 間の枝の重み，すなわち通信量を表す．したがって，与えられたタスクグラフと対象アーキテクチャについて，コスト関数を最小化するタスク割り当て手法を求めることが重要である．

2.3 FPGA リソース

FPGA が持つリソースとして演算リソースが定義されている [9]．演算リソースにはロジックリソース，メモリリソース，DSP リソースがあり，それぞれ FPGA の種類によって固定値をとるものとする．FPGA へタスクを実装することで演算リソースが消費される．したがって，FPGA における演算リソースの制約は以下のように表すことができる．

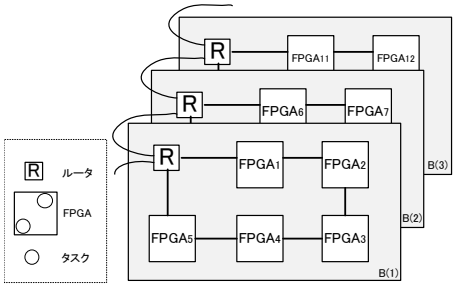


図 1: $N=5$ のマルチ FPGA システム．

$$\sum_{v_i \in V_{mount}} V_{logic}(i) \leq F_{cap_{logic}} \quad (2)$$

$$\sum_{v_i \in V_{mount}} V_{mem}(i) \leq F_{cap_{mem}} \quad (3)$$

$$\sum_{v_i \in V_{mount}} V_{dsp}(i) \leq F_{cap_{dsp}} \quad (4)$$

ここで V_{mount} は，ある FPGA に割り当てられたタスクの集合を表す．$V_{logic}(i)$，$V_{mem}(i)$，$V_{dsp}(i)$ はそれぞれ $v_i \in V$ が消費するロジックリソース，メモリリソース，DSP リソースを表す．$F_{cap_{logic}}$，$F_{cap_{mem}}$，$F_{cap_{dsp}}$ はそれぞれ FPGA のロジックリソース，メモリリソース，DSP リソース容量を表す．よって式 (2)〜(4) は，FPGA に割り当てられた各タスクのロジックリソース，メモリリソース，DSP リソースについて，それぞれの合計が各 FPGA リソースの容量を超えてはならないことを示している．

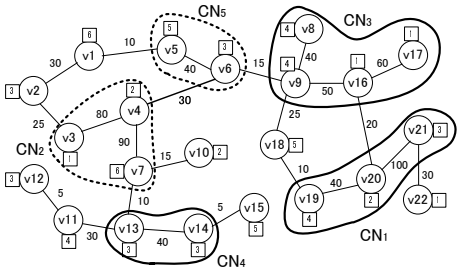
2.4 ネットワーク型マルチ FPGA システムを対象としたタスク割り当て問題

2.1 節，2.2 節，2.3 節により，ネットワーク型マルチ FPGA システムを対象としたタスク割り当て問題を以下のように定義する．

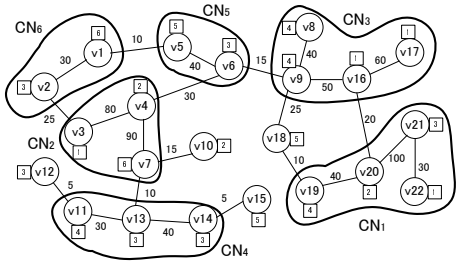
定義 1. 本稿で扱うネットワーク型マルチ *FPGA* システムを対象としたタスク割り当て問題とは，アプリケーションのタスクグラフ $G(V,E)$，ボード上の *FPGA* チップ数が N 個のマルチ *FPGA* システム，各 *FPGA* チップの演算リソース制約が与えられたとき，コスト関数 $Cost$ を最小化するようにタスクを割り当てることである． □

3 ネットワーク型マルチ FPGA システムを対象としたタスク割り当て手法

ネットワーク型マルチ FPGA システムを対象としたタスク割り当ては，通信量が大きいタスク同士をできるだけ同じ FPGA チップに割り当てるか，異なる FPGA チップに割り当てる際もホップ数が小さくなるようにできるだけ同じボードに割り当てる必要がある．したがって局所的なグルーピングを順に拡大するボトムアップな手法を提案する．



(a) タスクのクラスタリング.



(b) Step1 の実行結果.

図 2: Step1.

ネットワーク型マルチ FPGA システムを対象としたタスク割り当て手法として，以下の 5 ステップからなるものを提案する．

(1) タスクのクラスタリング.
(2) クラスタノードのタスクグラフの生成.
(3) ボードのタスクグラフの生成.
(4) 各ボードのクラスタノードのソート.
(5) ボードの順序の決定.

対象タスクグラフの節点を通信量の高いものを考慮しながら順にクラスタリングすることにより，タスクグラフを更新する．

3.1 タスクのクラスタリング (Step1)

Step1 ではタスクのクラスタリングを行う．Step1 のアルゴリズムを以下に示す．

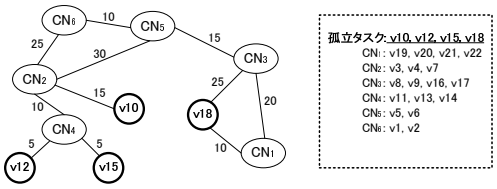
(1.1) タスクグラフ中で最大通信量で接続される 2 つのタスクの探索.
(1.2) 2 つのタスクの演算リソース和と FPGA の演算リソース制約の比較.
(1.3) 全てのタスク対を探索.

(1.1) ではタスクグラフから最大通信量で接続されるタスク $u, v \in V$ を探索する．

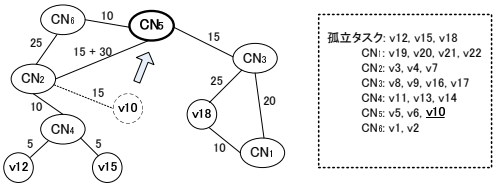
(1.2) では (1.1) で選んだ 2 つのタスク u と v の演算リソース和と，FPGA の演算リソース制約を比較する．通信コストを抑えるため，タスクグラフ中において大きな通信量で接続されるタスク同士は，同じ FPGA へ割り当てる必要がある．したがって，選んだ 2 つのタスク u と v の演算リソース和が FPGA の演算リソース制約内であれば，2 つのタスクをクラスタリングする．ここで，タスクのクラスタリングにより生成されたタスク集合をクラスタノードと呼ぶ．

(1.3) では，通信量が大きいタスク同士を順に (1.1) と (1.2) をタスクグラフ中で隣接するの全てのタスク対が探索されるまで繰り返す．

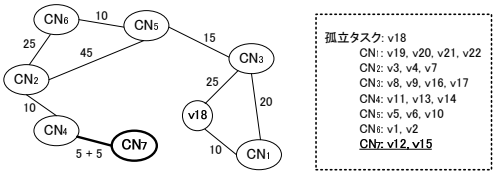
例 1. ここでは演算リソースのうちロジックリソースのみを制約として扱うことにする．与えられたタスクグラフに対して，通信量の高いタスク同士をクラスタリングしていきタスク v_4, v_6 が探索されたときの様子を図 2(a) に示す．図中の四角内の数字は，各タスクが消費するロジックリソースを表す．ロジックリソースの容量 $F_{cap_{logic}} = 10$ とする．タスク v_4, v_6 が探索されたとき，それぞれがクラスタリングされている CN_2 と CN_5 の演算リソース和を計算する．CN_2 の演算リソースは 9, CN_5 の演算リソースは 8 であり，演算リソース和が $F_{cap_{logic}}$ を超えるのでクラスタリングしない．全てのタスク対を探索してクラスタリングを行った結果を図 2(b) に示す．FPGA の演算リソース制約により，タスク対のクラスタリングによって生成されたクラスタノード (図中 CN_1〜CN_6) と，クラスタリングできずに残るタスクが存在する． □



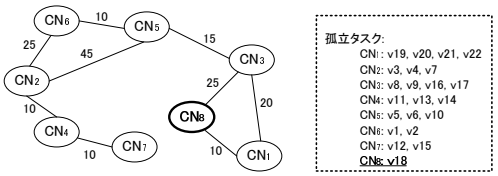
(a) クラスタノードと孤立タスク.



(b) 孤立タスクの移動.



(c) 孤立タスクのマージ.



(d) 孤立タスクをクラスタノードとして生成.

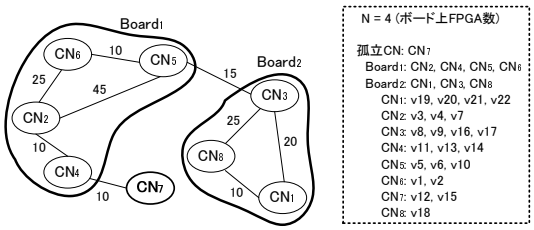
図 3: Step2.

3.2 クラスタノードのタスクグラフの生成 (Step2)

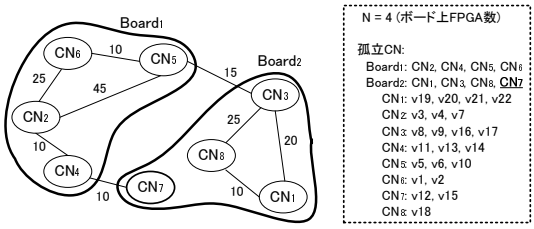
Step2 ではクラスタノードのタスクグラフの生成を行うことにより，各 FPGA チップへ割り当てるタスクを決定する．Step1 により，クラスタリングされずに残ったタスクが存在する．この残ったタスクを孤立タスクと呼ぶ．タスクグラフにおける全ての節点をクラスタノードとなるように，孤立タスクを以下の手順でクラスタノードへ移動，または孤立タスク同士をマージさせクラスタノードを生成する．Step2 のアルゴリズムを以下に示す．

(2.1) 孤立タスクの移動.
(2.2) 孤立タスクのマージ.
(2.3) 残った孤立タスクをクラスタノードとして生成.

(2.1) では孤立タスクをクラスタノードへ移動させる．タスクグラフ中における孤立タスクは，隣接するクラスタノードへ演算リソース制約により割り当てる

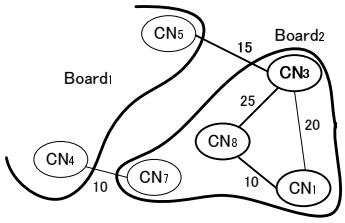


(a) クラスタノードのクラスタリング.

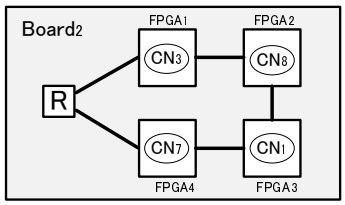


(b) 孤立 CN のクラスタリング.

図 4: Step3.



(a) クラスタノードの直線的ソート.



(b) 配置結果.

図 5: Step4.

ことができなかったものである．したがって，割り当てることのできなかったクラスタノードと高い接続関係にあるクラスタノードへ割り当てる．

タスクグラフ中からクラスタノードと孤立タスクで接続されるもののうち，最大通信量で接続されるクラスタノード CN_i と孤立タスク v を選ぶ．次に CN_i に隣接するクラスタノード集合の中から，CN_i と最大通信量で接続されるクラスタノード CN_j 選び，孤立タスク v を CN_j へ移動させることができるかどうか演算リソース制約により判定する．演算リソース制約を満足しなければ CN_j の次に大きな通信量で接続されるクラスタノードを順に選択していく．これにより，CN_i へクラスタリングすることができなかった孤立タスク v を，CN_i と高い接続関係のある CN_j へ割り当てることができ，少ないホップ数でコストを抑えることができる．

この移動工程を，タスクグラフ中における全ての孤立タスクについて行う．

(2.2) では孤立タスク同士をマージさせることにより，新たなクラスタノードを生成する．(2.1) 実行後，あるクラスタノード CN_i へ孤立タスクが複数接続される時，それら孤立タスク同士の演算リソース和が演算リソース制約を満たせばマージさせ，CN_i と隣接する新たなクラスタノードとして生成させる．マージさせることのできる孤立タスクがなくなったら終了する．

(2.3) では，(2.2) 実行後タスクグラフ中で残っている全ての孤立タスクを，中身がタスク 1 つのクラスタノードとしてそれぞれ生成させる．

例 2. Step1 の実行後のクラスタノードと孤立タスクのタスクグラフを図 3(a) に示す．孤立タスク $v10$ を CN_5 へ移動する様子を図 3(b) に示す．Step1 より，$v10$ は CN_2 へクラスタリングされなかったノードである．したがって CN_2 と高い接続関係にある CN_5 へ移動させる．孤立タスク $v12$ と $v15$ をマージさせ，新たなクラスタノードを CN_7 生成する様子を図 3(c) に示す．孤立タスクの移動とマージの実行後，残った孤立タスク $v10$ 自体を新たなクラスタノード CN_8 として生成する様子を図 3(d) に示す. □

3.3 ボードのタスクグラフの生成 (Step3)

Step3 では，Step2 実行後のタスクグラフ中のクラスタノードをクラスタリングすることにより，各ボードへ割り当てるタスクを決定する．Step3 のアルゴリズムを以下に示す．

(3.1) タスクグラフ中で最大通信量で接続される 2 つのクラスタノードを探索し，クラスタリング.
(3.2) 孤立 CN のクラスタリング.

(3.1) では Step1 と同様に，はじめにタスクグラフから最大通信量で接続されるクラスタノード CN_i, CN_j を探索する．通信コストを抑えるため，タスクグラフ中において大きな通信量で接続されるクラスタノード同士は，同じボードへ割り当てる必要がある．したがって，選んだ 2 つのクラスタノードをクラスタリングする．ただしクラスタリングした際のクラスタノード数が，ボード上に配置することのできる FPGA 数 N を超える場合はクラスタリングしない．

次に，通信量が大きいクラスタノード同士を順に，タスクグラフ中の全てのクラスタノード同士が探索されるまで繰り返す．

(3.1) 実行後，クラスタリングされずに残ったクラスタノードが存在する．この残ったクラスタノードを孤立 CN と呼ぶ．したがって (3.2) では Step2 と同様のアルゴリズムにより，孤立 CN を移動・マージさせ，残った孤立 CN は中身が 1 つのクラスタとして生成する．

例 3. (3.1) の実行結果を図 4(a) に示す．ボード上に配置することのできる FPGA 数 $N = 4$ を制約として，クラスタノードのクラスタリングによって生成された集合 (図中 $Board_1$, $Board_2$) と，クラスタリングできずに残る孤立 CN が存在する．CN_7 を $Board_2$ へ移動する様子を図 4(b) に示す．(3.1) より，CN_7 は $Board_1$ へクラスタリングすることのできなかったクラスタノードである. □

3.4 各ボードのクラスタノードのソート (Step4)

Step4 では各ボードへクラスタリングされたクラスタノード集合のソートを行うことによって，各ボード内における FPGA へ割り当てるノードの配置箇所を決定する．各ボード内では，N 個の FPGA チップとルータがリング状に接続されているため，N 個の FPGA チップが直線状に並んでいる．したがって，Step3 でクラスタリングしたボード内のクラスタノードについて，通信量を考慮しながら各クラスタノードの順序付けをする．$k = 1$ として, Step4 のアルゴリズムを以下に示す.

(4.1) $Board_k$ 内において直線的にソートを行う起点となるクラスタノード CN_{dep} の探索.
(4.2) $Board_k$ 内で接続されているクラスタノードの

配置順序の決定.

(4.3) 孤立 CN の配置.

(4.1) では $Board_k$ 内において直線的にソートを行う起点となるクラスタノード CN_{dep} を探索する．$Board_k$ 内のクラスタノードから，最大通信量で異なるボードと接続されるクラスタノード CN_{dep} を起点とする．異なるボードで接続されるタスク対がある場合，コストはホップ数が増えるため大きくなる．したがって，タスクグラフ中で隣接するが異なるボードにクラスタリングされたクラスタノードは，ボード間通信用ルータに隣接する FPGA へ配置する．

(4.2) では，$Board_k$ 内において (4.1) で探索されたクラスタノード CN_{dep} に隣接するクラスタノードの中で，最大通信量で接続されるクラスタノード CN_i を探索し，ボード上で CN_{dep} が配置された FPGA の隣の FPGA へ割り当てる．次に，$Board_k$ 内において CN_{dep}，CN_i に最大通信量で接続されるクラスタノードを探索し，ボード上で CN_i が配置された FPGA の隣の FPGA へ割り当てる．同様にして，$Board_k$ 内におけるクラスタノード全てを探索し，ボード上における配置箇所を決定する．

(4.3) は $Board_k$ 内において孤立 CN が存在する場合の操作である．タスクグラフ中において，孤立 CN は $Board_k$ 内にクラスタリングされた他のクラスタノード集合と接続されておらず，$Board_k$ と異なる他のボードにクラスタリングされたクラスタノードと接続されている．したがって，孤立 CN と接続されるクラスタノードとのホップ数を少なくしコストを抑えるため，孤立 CN はボード間通信用ルータから (4.1) と反対方向の向きに隣接する FPGA へ配置する．

最後に $k \leftarrow k+1$ として，全てのボード内における FPGA へ割り当てるタスクの配置箇所が決定されるまで (4.1)〜(4.3) を繰り返す.

例 4. (4.1), (4.2) の様子を図 5(a) に示す．$Board_2$ と異なるボードで接続される CN_3 を起点として，通信量の大きい順に接続されるクラスタノード CN_8，CN_1 を探索する．(4.3) の様子と配置結果を図 5(b) に示す．(4.2) で決定したクラスタノード CN_3, CN_8, CN_1 を順にボード間通信用ルータに隣接する FPGA から順に配置する．また，孤立 CN である CN_7 はボード間通信用ルータから反対方向の向きに隣接する FPGA へ配置する． □

3.5 ボードの順序の決定 (Step5)

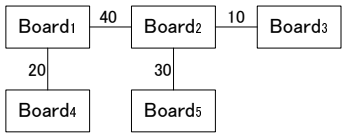
Step5 ではボードの順序の決定を行う．各ボードはリング状に接続される．したがって，各ボード間の通信量を考慮しながらボードの順序付けをする．Step5 のアルゴリズムを以下に示す.

(5.1) タスクグラフ中で最大通信量で接続されるボードを探索し接続.

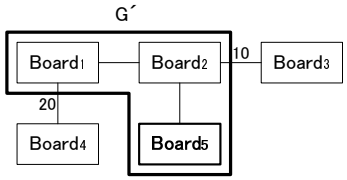
(5.2) 接続されたボードにタスクグラフにおいて最大通信量で接続されるボードを探索.

(5.1) では Step3 で得られたボードのタスクグラフから，最大通信量で接続される $Board_i$ と $Board_j$ を探索し，これら 2 つのボードを接続させる．また，$Board_i$ と $Board_j$ を 2 つの節点とし，この間を接続したグラフ $G' = (V', E')$ を生成する．ここで，$V' = \{Board_i, Board_j\}$, $E' = (Board_i, Board_j)$ である.

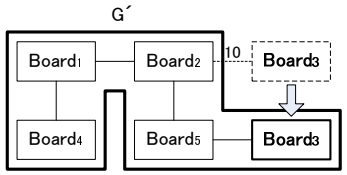
(5.2) では，グラフ G' に含まれるボード集合にタスクグラフから最大通信量で接続されるボード $Board_k$ を探索する．ここでボードを直線状に接続するとは，最大次数 $\Delta(G') \leq 2$ の無閉路グラフを作成することに等しい．グラフ G' へ $Board_k$ を追加して接続させることにより，直線状に接続されたグラフを作成する．このとき以下の場合分けをする.



(a) ボードのタスクグラフ.



(b) 直線状にソート (5.2)(i).



(c) 直線状にソート (5.2)(ii).

図 6: Step5.

(i) タスクグラフ中で $Board_k$ が，グラフ G' 内で端にあるボード，すなわち $deg(Board_l) = 1$ である $Board_l$ に接続されているとき，$Board_k$ はグラフ G' へ $Board_l$ に接続させることで追加する.

(ii) タスクグラフ中で $Board_k$ が，グラフ G' 内で端以外にあるボード，すなわち $deg(Board_l) = 2$ である $Board_l$ に接続されているとき，グラフ G' へ $Board_k$ をグラフ G' 内で端に接続されている $Board_m$ と $Board_n$ のうち，$d(Board_l, Board_m) \leq d(Board_l, Board_n)$ である $Board_m$ に接続させることで追加する.

(i) はグラフ G' 内のボード集合の中で，$Board_k$ とタスクグラフ中において接続されるボード $Board_l$ が，グラフ G' において直線状に接続されたグラフの端に接続されている場合である．$Board_k$ と $Board_l$ は，直線状に並べる際もコストが小さくなるよう G' 内で隣接するように接続させる.

(ii) はグラフ G' 内のボード集合の中で，$Board_k$ とタスクグラフ中において接続されるボードが，グラフ G' において直線状に接続されたグラフの端以外の箇所で接続されている場合である．(i) と同様に，直線状に並べる場合，コストが小さくなるよう G' 内でできるだけ近くに接続させる.

(5.2) をタスクグラフ中の全てのボードが探索されるまで繰り返す.

例 5. 与えられたボードのタスクグラフを図 6(a) に示す．(5.2) の (i) の様子を図 6(b) に示す．はじめに (5.1) によりボードのタスクグラフから最大通信量で接続される $Board_1$ と $Board_2$ をグラフ G' に追加し接続する．次に $Board_1$ と $Board_2$ に隣接し，最大通信量で接続される $Board_5$ を選択する．$Board_5$ はグラフ G' の端に存在する $Board_2$ にタスクグラフにおいて接続されているので，グラフ G' へ $Board_2$ と接続されるように追加する．続いて $Board_4$ も (5.2) の (i) により，グラフ G' へ $Board_1$ と接続されるように追加する．(5.2) の (ii) の様子を図 6(c) に示す．$Board_3$ はグラフ G' の端でないボード $Board_2$ と接続されている．したがって (5.2) の (ii) より，$Board_2 - Board_5$ 間と $Board_2 - Board_4$ 間のうち距離の短い $Board_5$ を選択し，$Board_3$ をグラフ G' へ $Board_5$ と接続されるように追加する． □

表 1: 実験結果.

タスクグラフ	対象アーキテクチャ	アルゴリズム	使用ボード数	コスト	CPU 時間 [sec]
MPEG-4 (タスク数: 12)	$N = 3$	SA 法	2	5772.71 (1.00)	16.89
		[5]	2	5892.07 (1.02)	0.40
		提案手法	2	6061.82 (1.05)	4.63
	$N = 4$	SA 法	1	5254.15 (1.00)	9.59
		[5]	1	5725.5 (1.09)	0.36
		提案手法	1	5368.1 (1.02)	4.89
	$N = 5$	SA 法	1	5242.42 (1.00)	17.59
		[5]	1	5778.91 (1.10)	2.56
		提案手法	1	5353.43 (1.02)	5.53
VOPD (タスク数: 16)	$N = 3$	SA 法	2.1	5991.21 (1.00)	79.86
		[5]	2	6785.5 (1.13)	0.72
		提案手法	2.3	5931.1 (0.99)	7.54
	$N = 4$	SA 法	2	5690.5 (1.00)	51.15
		[5]	2	6314.09 (1.11)	0.39
		提案手法	2	5479.03 (0.96)	6.13
	$N = 5$	SA 法	2	5493.3 (1.00)	48.85
		[5]	2	5865.72 (1.07)	0.52
		提案手法	2	5591.13 (1.02)	6.03
Hog-AdaBoost (タスク数: 69)	$N = 3$	SA 法	9.5	28801.12 (1.00)	819.73
		[5]	8	57365.23 (1.99)	6.87
		提案手法	9.8	30195.31 (1.05)	77.97
	$N = 4$	SA 法	6.3	25013.31 (1.00)	1883.07
		[5]	6	56035.11 (2.24)	7.664
		提案手法	6.8	26955.54 (1.08)	83.855
	$N = 5$	SA 法	5.6	24350.82 (1.00)	1867.93
		[5]	5	59440.22 (2.44)	7.452
		提案手法	6.1	26705.34 (1.10)	77.067

4 計算機実験結果

提案手法を C++ 言語を用いて計算機上に実装した．計算機実験環境は，CPU が Intel Core i7-2640M 2.80GHz，メモリ容量が 16GB である．タスク割り当てを適用する対象アプリケーションとして，MPEG-4, VOPD (Video Object Plane Decoder), Hog-AdaBoost を用いる．それぞれのタスク数は 12, 16, 69 である．FPGA のロジックリソース，メモリリソース，DSP リソースの容量をそれぞれ 10000，4000，200 とした．またタスクグラフにおける各タスクが消費するロジックリソース，メモリリソース，DSP リソースをそれぞれ 2000〜4000，500〜1000，10〜50 の範囲でランダムな数値を与えた．

タスク割り当て手法の比較として，SA (Simulated Annealing) 法と従来手法 [5] を用いた．3 つの対象アプリケーションについて，50 回それぞれの 3 つの手法を適用し，その平均をとった結果を表 1 に示す．

MPEG-4 と VOPD について，提案手法は SA 法と比較してコストを最大 3.71% 削減できることを確認した．また，提案手法は SA 法と比較して CPU 時間を平均約 1/4 に抑えることができた．Hog-AdaBoost について，提案手法は SA 法と比較してコストが平均 7.42% 増加したが，CPU 時間は平均して約 1/17 に抑えることができた．従来手法 [5] と比較するとタスク数が多いときコストに大きな差が確認でき，最大 55.1% のコストの削減がされていることを確認した．これらのことから提案手法は有効な手法であると言える．

5 おわりに

本稿では，ネットワーク状に構成されたマルチ FPGA システムとして，CoRD システムを基にしたリング状のネットワークアーキテクチャへのタスク割り当て手法を提案した．また，実在するアプリケーションを用いて，提案手法を SA 法，従来手法とで比較し評価を行った．実験結果より，提案手法の有用性を確認することができた．

今後の課題としては，FPGA-FPGA 間における IO リソースを制約として加えた場合のタスク割り当て手法の考案が挙げられる．また，マルチ FPGA システムへのアプリケーションの追加時のタスク割り当て手法の考案が挙げられる．

参考文献

[1] C. Chang, J. Wawrzynek, and R.W. Brodersen, “BEE2: A high-end reconfigurable computing system,” in *Proc. IEEE Des. Test Comput.*, vol.22, no.2, pp. 114–125, 2005.

[2] H. C. Chi, J. F. Feng, and Y. C. Hsieh, “Area utilization based mapping for network-on-chip architectures with oversized IP cores, ” in *Proc. IEEE 14th International Conference on High Performance Computing and Communications & IEEE 9th International Conference on Embedded Software and Systems*, pp. 1520–1525, 2012.

[3] 伊吹山秋彦，濱田剛，中里直人，奥山祐市, “PROGRAPE-4 ボードと PGR による 2 次元 Kolmogorov-Smirnov テスト,” 信学技報, RECONF2006-28, 2006.

[4] M. Janidarmian, A. Khademzadeh, and M. Tavanpour, “Onyx: A new heuristic bandwidth-constrained mapping of cores onto tile- based network on chip,” *IEICE Electronics Express*, vol. 6, no. 1, pp. 1–7, 2009.

[5] 片野弘規，李昇周，戸川望，青木孝，関原悠介，中西衛, “局所性を考慮したマルチ FPGA システム向けタスクマッピング手法,” 信学技報, VLD2013-126, 2014.

[6] H. Morishita, K. Inakagata, Y. Osana, and N. Fujita, “Implementation and evaluation of an arithmetic pipeline on FLOPS-2D: multi-FPGA system,” in *Proc. International Workshop on Highly-Efficient Accelerator and Reconfigurable Technologies 2010*, pp. 8–13, 2010.

[7] T. Otsuka, T. Aoki, E. Hosoya, and A. Onozawa, “An image recognition system for multiple video inputs over a multi-FPGA system,” in *Proc. MCSoC 2012*, pp. 1–7, 2012.

[8] W. T. Shen, C. H. Chao, Y. K. Lien, and A. Y. Wu, “A new binomial mapping and optimization algorithm for reduced-complexity mesh-based on-chip network,” in *Proc. International Symposium on Networks-on-Chip*, pp. 317–322, 2007.

[9] 右近祐太, 大塚卓哉, 青木孝, 関原悠介, 宮崎昭彦, “共有型マルチ FPGA システムに向けた動的再配置手法と評価,” 信学技報, VLD2013-100, 2013.