

# ハイパーキューブ結合の並列計算機 におけるタスクアロケーション手法

## A TASK ALLOCATION METHOD FOR HYPER-CUBE CONNECTED PARALLEL COMPUTER

堀池 聡  
Horiike Satoshi

三菱電機 (株)  
Mitsubishi Electric Corp.

あらまし ハイパーキューブ結合は並列計算機の有力なプロセッサネットワーク方式として注目されている。本稿ではプロセッサがハイパーキューブで結合された並列計算機へのタスクアロケーション手法を提案する。複数のタスクとタスク間の依存関係が、タスクグラフとしてノードとアークで表されているとする。そのグラフのノードをハイパーキューブ計算機の一つのプロセッサに割り当てたときに、できる限り多くのアークがハイパーキューブの通信ラインと一致することを、本手法では目的としている。本手法はハイパーキューブの構造が再帰的に定義できることを活用しており、タスクグラフの中にハイパーキューブに近い構造を見いだす操作を繰り返す。

Abstract A new task allocation method for hypercube connected parallel computers is proposed. Task graphs are expressed by task nodes and communication arcs. Task allocation problem deals with mapping of task nodes onto processors so as to minimize the communication overhead. The objective function is the cardinality that is the number of arcs mapped onto links of parallel computers. The proposed method is recursive one. That is, similar structure to  $n$  dimensional hypercube is found in the task graph using the result for  $n-1$  dimensional hypercube.

### 1. まえがき

近年注目を集めている並列プロセッサのネットワーク構造にハイパーキューブ<sup>[1]</sup>がある。 $n$ 次元のハイパーキューブは $n$ 次元超立方体の頂点にプロセッサ、辺にリンクを配置する構造である。このネットワークは、 $H/W$ コスト、通信距離、対称性、柔軟性、耐故障性等の点で優れており、すでにNCube<sup>[1]</sup>、コネクションマシン<sup>[2]</sup>、iPSC等の並列計算機に採用されている。

ハイパーキューブ構造のような並列計算機上で、並列プログラムを効率よく走らせるためには、従来の逐次処理計算機にはなかった技術が必要になる。例えば、問題のモデル表現の方法、並列性の抽出、タスク間の同期の取り方等がうまく行なわれないと、並列計算機の性能をフルに引き出すことができない。

本稿では、並列計算機特有の問題の1つであるタスクアロケーションを扱う。並列プログラムはいくつかのタスクで構成される。1つのタスクは複数のプロセッサのうちの1つに割り当てられる。通常、あるタスクの実行は他のいく

つかのタスクの実行になんらかの関連があり、その関連はプロセッサ間の通信によって実現される。通信に要する時間を短くするためには、プロセッサ間の距離が短くなければならない。プロセッサ間の距離は各タスクがどこに割り当てられるかによって変わるので、実行性能はタスクの割当てに大きく依存する。

並列プログラムのタスクを並列計算機のプロセッサ上に割当てて、実行時間を最小化する問題はNP完全問題<sup>[3]</sup>である。そこで、良いヒューリスティクスあるいはグラフ理論に基づいた方法が必要となる<sup>[3]-[6]</sup>。

本稿ではタスクグラフで表されたモデルをハイパーキューブで構成された計算機に割当てる手法を提案する。ハイパーキューブの構成が再帰的に定義できることを本手法では活用しており、カーディナリティと呼ばれる評価値をできる限り大きくする。

第2章でハイパーキューブ構造を説明する。第3章でタスクアロケーションの問題を定義する。第4章でハイパーキューブ構造の計算機におけるタスクアロケーション手法を提案する。第5章では例題を用いて本手法を説明し、他の手法<sup>[4]</sup>との簡単な比較を行なう。

## 2. ハイパーキューブ結合

ハイパーキューブネットワークは、リンク本数・直径等の評価値を総合的に判断すると、他のリング上ネットワーク、格子状ネットワークなどに比べて極めて優れている<sup>[2]</sup>。

$n$ 次元ハイパーキューブネットワーク（いかでは $n$ 次元キューブと略称する）では、ノード数が $N = 2^n$ で、 $N$ 個のプロセッサが $n$ 次元超立方体の頂点に配置される。本章ではその接続方法として以下に2通りの説明を与える。

まずプロセッサの番号によって説明する。

プロセッサの数を $N (= 2^n)$ として、各プロセッサに0から $N-1$ までの番号を与える。これらの番号は $n$ ビットの2進数で表現できる。各プロセッサには、 $n$ 個のプロセッサがリンクで接続される。接続するプロセッサはお互いのプロセッサ番号が2進表現で1ビットだけ異なっている。

例えば $n = 6$ のネットワークにおいて番号37(=100101)のプロセッサには、番号36(100100)、39(100111)、33(100001)、45(101101)、53(110101)、5(000101)の6個のプロセッサが接続する。4次元キューブの構成とプロセッサ番号を例として図1に示す。

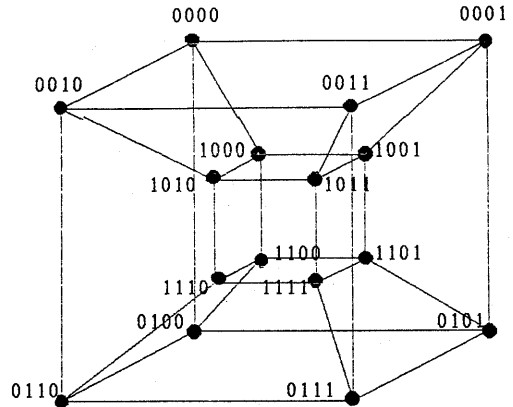


図1 4次元キューブのプロセッサ番号

さらに、ハイパーキューブネットワークは再帰的な定義も可能である。つまり、 $n+1$ 次元キューブの構成は2個の $n$ 次元キューブを用いて次のように定義できる。

$n$ 次元キューブは $N = 2^n$ のプロセッサノードを持つ。 $N$ 個のプロセッサノードには0から $N-1$ までのプロセッサ番号がすでにつけられている。どちらか片方のネットワークに属するプロセッサのプロセッサ番号には $n$ ビット目に

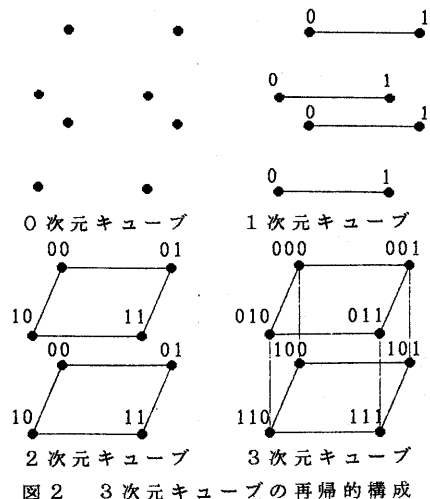


図2 3次元キューブの再帰的構成

1を加えておく。プロセッサ番号の $n-1$ ビット以下が同じである2個のプロセッサは $N$ 組ある。それらを $N$ 本のリンクで結合すれば $n+1$ 次元のハイパーキューブネットワークが構成できる。

0次元から順番に3次元のハイパーキューブが構成される過程を図2に示す。ここで、0次元のハイパーキューブは単独のノードで構成されている。

### 3. タスクアロケーション問題の定義

並列計算機の1つのプロセッサで実行されるプログラムの単位をタスクと呼ぶ。並列計算機上で実行する問題はいくつかのタスクからなる。どのプロセッサにどのタスクを実行させるかを決定する操作をタスクアロケーションと呼ぶ。

通常、あるタスクは実行中に他のタスクとの通信が必要である。タスクをノードとし、通信を行なうノード間をアークで結合すれば、それは1つのグラフを表す。このグラフをタスクグラフと呼ぶ。タスクグラフのノードはタスクノードと呼ぶ。アークはタスクグラフのノードを結ぶアークのこととし、ネットワークにおいてプロセッサを結合するリンクと区別する。

タスクグラフの構造<sup>[3]</sup>は、プログラムの実行に伴い変化する(ダイナミック)場合と、不変である(スタティック)場合の2つに分かれる。ダイナミックな場合はアロケーションに必要となる時間そのものが実行時間に関わってくるので、高速のアロケーションアルゴリズムが必要になる。スタティックな場合はタスクのアロケーションがプログラムのコンパイル時に行えるので、低速であっても効率のよいアルゴリズムが必要になる。

またタスクグラフはTPG(Task Precedence Graph)とTIG(Task Interaction Graph)の2通りがある<sup>[3]</sup>。前者ではタスクのアークが方向性を持った有向グラフで表され、タスクの生成関係を意味している。後者ではアークはプロセス間の通信を表す。いっぽんに通信は双方向に行なわれるので、このグラフのアークは無向である。

本稿では、スタティックなTIGのタスクノードを、ハイパーキューブネットワークで結合

されたプロセッサ上に割り当てる問題を扱う。

TIG中のタスクの数はプロセッサの数に等しいと仮定する。タスクの数が少ないときにはこの仮定は問題がない。仮定のタスクをつけ加えることによつて容易に同数にできるからである。逆の場合、つまりタスクの数の方がプロセッサの数より多い時には、いくつかのタスクをグループ化して1つのタスクに置き換えるという操作が必要であり、マッピング問題における1つの課題である。

また、マッピングの良否を決める指標としては様々な評価関数が考えられるが、本稿ではカーディナリティ<sup>[4]</sup>を採用する。カーディナリティとは、TIGに対してあるマッピングを行なった結果、プロセッサネットワークの実際に存在するリンクと一致するTIGのアークの数で定義される。

図3に示すTIGを例として、3次元キューブに2通りのマッピングを行なった結果を図4と図5に示す。図中で、細い線はアークと一致していないリンクで、太い線はアークとリンク

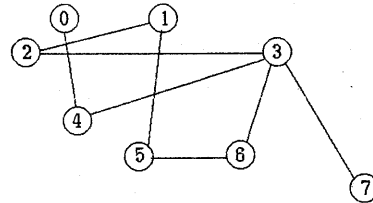


図3 TIGの例

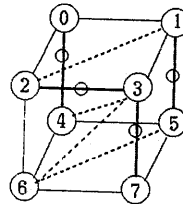


図4 図3のTIGに対するアロケーション1

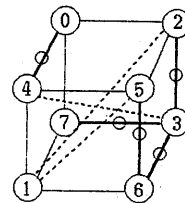


図5 図3のTIGに対するアロケーション2

が一致しており、破線はリンクと一致していないアークを示している。太い線は一致していることを分かりやすくするために丸印を付加している。この表示法は次章以下の図でも同様である。図4のマッピングではカーディナリティが4であり、図5では5となっている。図5の方が良いマッピングであるといえる。

#### 4. アロケーション手法

2章で述べたように、ハイパーキューブを構成するには、2個の $k$ 次元キューブをリンクで結合して、 $(k+1)$ 次元キューブをつくるという操作が繰り返される。提案する手法はこの構成過程を利用している。すなわち、TIGの中に、ハイパーキューブに最も近い構造を0次元から順番に見つけ出していくという操作によってマッピングを行なう。

ノード数が $N=2^n$ とすると、提案するアルゴリズムは、同じ処理を $n$ 回繰り返してタスクノードを $n$ 次元キューブにマッピングする方法である。

まず、TIGにおける $N$ 個のタスクノードを $N$ 個の0次元キューブに分割する。つまり、0次元キューブにはプロセッサが1個含まれているので、そのプロセッサの番号を0とし、タスクノードを1個割り当てる。この状態を初期状態とする。

段階 $k$ で処理を行なう前に、TIGにおける $N$ 個のタスクノードは、 $M=2^k$ 個の $k$ 次元キューブのいずれかに属している。各 $k$ 次元キューブでは、 $K=2^k$ 個のプロセッサにタスクノードが1個ずつ割り当てられている。ここで、 $k=n-m$ としている。そして、段階 $k$ では2個の $k$ 次元キューブをリンクで結合して $k+1$ 次元キューブにマッピングし、TIGのタスクノードを $M/2$ 個の $k+1$ 次元キューブに分割する。

この処理を $n$ 回繰り返すと $N$ 個のタスクは $n$ 次元キューブにマッピングされる。

1つのハイパーキューブに属するタスクノードと他のハイパーキューブに属するタスクノードを1つのペアとして定義する。 $k$ 次元キューブをまとめて、 $k+1$ 次元キューブにマッピングするという操作は、 $k$ 次元キューブ間で $M$ 個

のペアをつくり、ペア間をリンクで結合することである。

ここでカーディナリティを大きくするためには、 $k$ 段階で存在する $M$ 個の $k$ 次元キューブのうちで、どの2個の $k$ 次元キューブを組み合わせ、その2個の $k$ 次元キューブに対してプロセッサをどの様に結合するかが問題になる。以下ではその方法について述べる。

#### 処理1. ハイパーキューブ間の結合法

段階 $k$ で存在する $k$ 次元キューブを $C_{k,i}$  ( $i=1,2,\dots,M$ )とする。まず、ある2個の $k$ 次元キューブ $C_{k,i}$ と $C_{k,j}$ に着目し、もしこの2つの $k$ 次元キューブを組み合わせると $k+1$ 次元キューブにするならばどの様にペアをつくるのが最良であるかを考える。TIGと照らし合わせた結果、この2個の $k$ 次元キューブ間にまたがるアークがいくつか存在していると仮定する。2個のグループを結合するときには、一方の $k$ 次元キューブのタスクノードと他方の $k$ 次元キューブのタスクノードをリンクで結合するが、このとき全体としてはできる限り多くのアークがリンクと一致するような組合せを見いださなければならない。

$C_{k,i}$ と $C_{k,j}$ の2個の $k$ 次元キューブにはそれぞれ番号0から番号 $K-1$ までの $K$ 個のプロセッサがある。 $k$ 次元キューブをまとめるときには同じ番号のプロセッサをリンクで結合する。各プロセッサの番号をそのまま用いてこのように結合したとしても、この2つの $k$ 次元キューブの結合に関してアークとリンクが一致する数が最大になるとは限らない。

そこで、 $C_{k,i}$ か $C_{k,j}$ のどちらかのプロセッサ番号を変更する。プロセッサ番号を変えるときには $k$ 次元キューブのプロセッサの隣接関係は保たれていなければならない。1個の $k$ 次元キューブについて隣接関係を保ちつつプロセッサ番号を変更するには次のビット操作が許される。

ビット操作1.

ビットの並び方を変える。

ビット操作2.

ビットを反転する。つまり、ビット1はビット0にビット0はビット1にする。

2個のk次元キューブにまたがる複数のアーキについて、両者でのプロセッサ番号が同じであるアーキの数が最大になるように上記の操作を行なう。

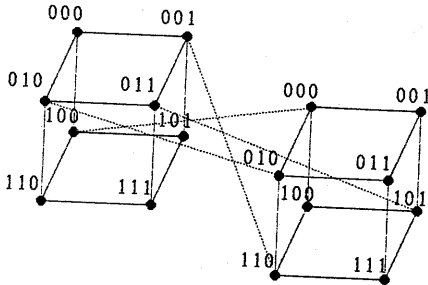


図6 3次元キューブの結合

図6における2個の3次元キューブの結合を例として説明する。各ハイパーキューブには前の段階で決まったプロセッサ番号がつけられている。この時点で2個の3次元キューブの間には4本のアーキが存在している。その4本のアーキがつながっているプロセッサ番号を以下に列挙する

- アーキ1: 番号001と番号110
- アーキ2: 番号011と番号101
- アーキ3: 番号010と番号010
- アーキ4: 番号100と番号000

この状態で、リンクと一致するアーキの数は1本である。ここで、右側の3次元キューブのプロセッサ番号について、0番目のビットと1番目のビットを入れ替え、2番目のビットを反転すると各アーキの両端のプロセッサ番号は、

- アーキ1: 番号001と番号001
- アーキ2: 番号011と番号011
- アーキ3: 番号010と番号101
- アーキ4: 番号100と番号100

となり、リンクと一致するアーキの数は3本で最大になる。

ビット操作1と2により、リンクと一致するアーキの最大数を効率的に求めるアルゴリズムは、検討すべき問題である。ただし、プロセッサの数が1024としてもビットの長さはせい

ぜい10であるので、列挙法によって調べることも可能ではある。

この処理で2個のk次元キューブを組み合わせた時に、リンクと一致させることが可能なアーキの数の最大値をキューブ間のカーディナリティと呼ぶ。

### 処理2. ハイパーキューブの組合せ

次の問題は、ハイパーキューブの組み合わせの方法である。この問題を解決するために重み付きの一般グラフにおける最大マッチングのアルゴリズムを活用する。これは重み付きのグラフのアーキの部分集合のうちで、同じノードを共有しないという条件の下で、アーキの重みの総和を最大にする集合を求めるアルゴリズムである。このアルゴリズムについては文献[7]に詳しい。

段階kの処理を行なう前にできているM個のk次元キューブとTIGから重み付きのグラフをつくる。このグラフではk次元キューブ  $C_{k,i}$  ( $i=0,1,2,\dots,M$ )をノードiとみなす。

$C_{k,i}$ と $C_{k,j}$ に関するグループ間のカーディナリティを $W_{ij}$ とする。ノードiとノードjの間は、 $W_{ij}$ を重みとするアーキで結合する。ただし、 $W_{ij}$ がゼロの時にはアーキとはしない。

この重み付きのグラフに対して、一般グラフの重み付き最大マッチングのアルゴリズムを適用する。得られるマッチングは、段階kにおいてカーディナリティを最も増加させることのできるグループの組合せを示している。

組み合わせるグループが決まれば各タスクノードが属するハイパーキューブの番号とプロセッサ番号を変更する。

### 処理3 番号の変更

$C_{k,0}$ から $C_{k,k-1}$ までのk次元キューブは処理2でどの様に組み合わせるかは決まっている。K+1次元キューブの数は $k/2$ 個であるから、 $C_{k+1,0}$ から $C_{k+1,k/2-1}$ までの番号を新しいハイパーキューブに適当に割り当てる。

プロセッサ番号についてはまず、キューブのカーディナリティが最大になるように一方のグループ番号を変更する。そしていずれかのグループについて、プロセッサ番号の第kビットを

1 にすればよい。

提案するマッピングのアルゴリズムを以下に簡単にまとめる。

ステップ1. (初期設定)

タスクグラフ中のタスクノード  $i$  が割り当てられるハイパーキューブの番号を  $i$ 、プロセッサ番号を 0 とする。また  $k$  も 0 としておく。

ステップ2. (段階  $k$  の処理)

処理1 によって、 $k$  次元キューブの全ての組合せに対してカーディナリティ  $W_{ij}$  を計算する。

処理2 によって  $G_{k,i}$  と  $W_{ij}$  から重み付きグラフをつくる。そのグラフに対して、重み付きグラフの最大マッチングアルゴリズムを適用し、グループの組合せを決める。

処理3 により各タスクノードに割り当てられるハイパーキューブの番号とプロセッサ番号を変更する。

$k = n - 1$  ならば終了する。

$k < n - 1$  ならば  $k = k + 1$  としてステップ2 を繰り返す。

6. 適用例

図7 に示している TIG を用いて本手法を説明する。この TIG のタスクノード数は 16 である。

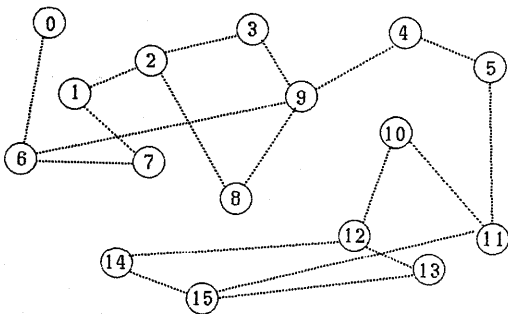


図7 TIG の例

ステップ0 としてタスクノード  $i$  を 0 キューブ  $C_{0,i}$  ( $i=0,1,2,\dots,N-1$ ) とする。

段階0 として次の処理を行なう。

まずキューブ間のカーディナリティを調べる。

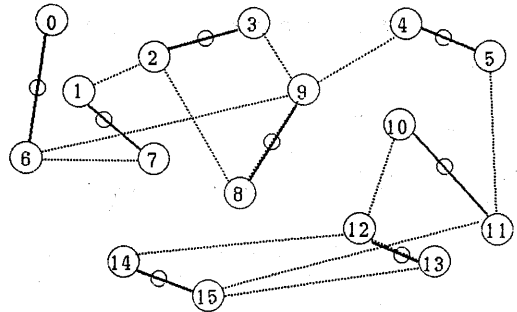


図8 図7のTIGに対する1次元キューブの構成

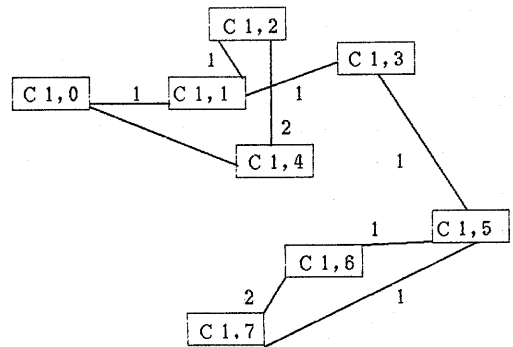


図9 段階1での重み付きグラフ

この段階ではハイパーキューブの番号に対応するタスクノード間でアークが存在するときのみキューブ間のカーディナリティが1であり、アークが存在しないときには0であることは自明である。重み付きグラフのマッチングアルゴリズムを適用した結果を図11に示す。この組合せでできる新しい1次元キューブの構成は以下のようなになる。

- $C_{1,0} = (C_{0,0}, C_{0,6})$
- $C_{1,1} = (C_{0,1}, C_{0,7})$
- $C_{1,2} = (C_{0,2}, C_{0,3})$
- $C_{1,3} = (C_{0,4}, C_{0,5})$
- $C_{1,4} = (C_{0,8}, C_{0,9})$
- $C_{1,5} = (C_{0,10}, C_{0,11})$
- $C_{1,6} = (C_{0,12}, C_{0,13})$
- $C_{1,7} = (C_{0,14}, C_{0,15})$

次に1次元キューブを1つのノードとし、キューブ間のカーディナリティをアークの重みとしたグラフをつくる。この重み付きグラフを図9に示す。再びこのグラフにマッチングアルゴリズムを適用する。その結果図7のTIGは2次元キューブによって図10のように構成される。

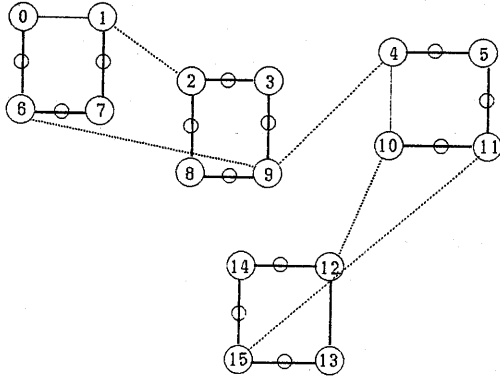


図10 図7のTIGに対する  
2次元キューブの構成

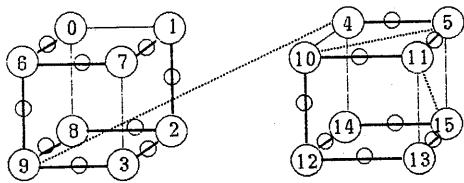


図11 図17のTIGに対する  
3次元キューブの構成

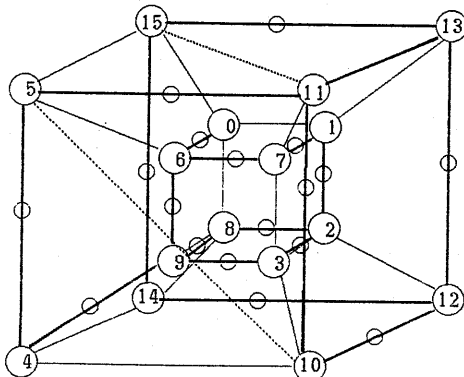


図12 図7のTIGに対する  
4次元キューブの構成

また、新しい2次元キューブは1次元キューブを次のように組み合わせている。

$$G_{2.0} = (G_{1.0}, G_{1.1})$$

$$G_{2.1} = (G_{1.2}, G_{1.4})$$

$$G_{2.2} = (G_{1.3}, G_{1.5})$$

$$G_{2.3} = (G_{1.6}, G_{1.7})$$

以下同様の操作を繰り返すと、図7のTIGは3次元キューブで構成すると図11、4次元キューブで構成すると図12になる。

ハイパーキューブの組合せは  $k=2$  で、

$$G_{3.0} = (G_{2.0}, G_{2.1})$$

$$G_{3.1} = (G_{2.2}, G_{2.3})$$

$k=4$  で、

$$G_{4.0} = (G_{3.0}, G_{3.1})$$

であり、最終的に4次元キューブが構成される。

次に、TIGとしてノード数16のリング状ネットワークを採用する。ハイパーキューブネットワークは内部にリングを含んでいるので、最適のアロケーションを行なうと全てのアークはリンクと一致する。本手法を適用すれば、図15に示すようなアロケーションが行なわれ、最大のカーディナリティ16が得られている。

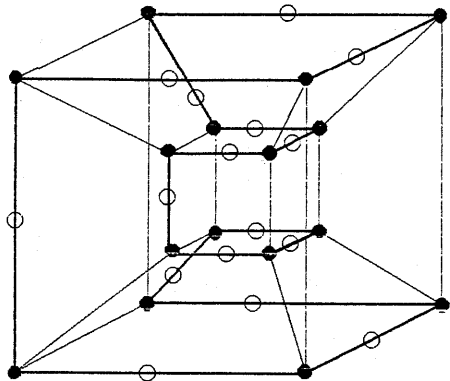


図13 本手法によるリング状ネットワーク  
のアロケーション

本手法との比較のために、次のようなアルゴリズムによるアロケーションを考える。

## ステップ0

初期値として適当なアロケーションを与え、このアロケーションに対するカーディナリティを計算する。

## ステップ1

N個のタスクノードのうちのある2つのノードに対するアロケーションを入れ替えたと仮定した時のカーディナリティの変化を調べる。全てのノードの組に対する変化を調べた後に、最もカーディナリティの値を増加させる2個のタスクノードを実際に入れ替える。

入れ替えの操作が生じた時にはステップ1を繰り返す。

カーディナリティの値を増加させるような組が見つからなかった時には終了する。

この手法は文献[4]で提案する方法を簡略化したものである。[4]では増加させる組が見つからなかった時には、初期状態をランダムに変えて、再び探索することを提案している。

このアルゴリズムを使ってリング状のTIGのアロケーションを行ない、図14の状態になったとする。このアロケーションに対するカーディナリティは12であるが、この状態ではどの2つのタスクノードのアロケーションを入れ替えてもカーディナリティを増加させられない。この時点でのカーディナリティは12であるので本手法の方が適していることがわかる。

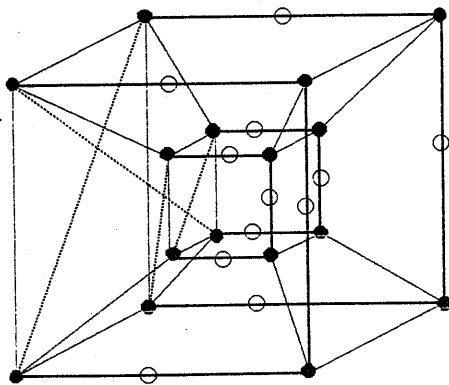


図14 ノード交換によるアロケーション

## 6. あとがき

本稿では、ハイパーキューブの生成過程を利用したタスクアロケーション手法を提案し、例題によって本手法が有効であることを示した。本手法はハイパーキューブの構成における再帰的な定義を活用している。

カーディナリティによる目的関数では、一致しなかったアークの取扱いが考慮されていない。これらのアーク上の通信を実現するパスの長さも組み込んだ目的関数、あるいは最長のパスの長さを最小にする目的関数としたアロケーション手法<sup>[5]</sup>も今後考察していく。

### [参考文献]

- [1] G.Fox, "Solving Problems on Concurrent Processors," Prentice Hall, Englewood Cliffs, NJ, 1988
- [2] W.D.Hillis, "The Connection Machine," MIT Press, Cambridge, Mass., 1985
- [3] P.Sadayappan, F.Ercal, "Nearest-Neighbor Mapping of Finite Element Graphs onto Processor Meshes," IEEE Trans. Comput., vol.C-36, pp.1408-1424, Dec. 1987
- [4] S.H.Bokhari, "Assignment Problems in Parallel and Distributed Computing," Kluwer Academic Publishers,
- [5] S.Y.Lee, J.K.Aggarwal, "A Mapping Strategy for Parallel Processing," IEEE Trans. Comput., vol.C-36, pp.433-442 Apr. 1987.
- [7] P.Y.Richard, Y.S.Lee, and M.Tsuchiya, "A Task Allocation Model for Distributed Computing Systems," IEEE Trans., Comput., Vol.C-31, pp.41-47, 1982.
- [8] E.L.Lawler, "Combinatorial Optimization Networks and Matroids," Holt, Rinehart and Winston, NY, 1976