ソフトウェア基礎論　27－1
プログラミング言語　19－1
（1988. 12. 9）

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[1]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[1]、コネクションマシン[2]、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[3]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[3]-[6]。

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[4]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に比べて極めて優れている[2]。

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。

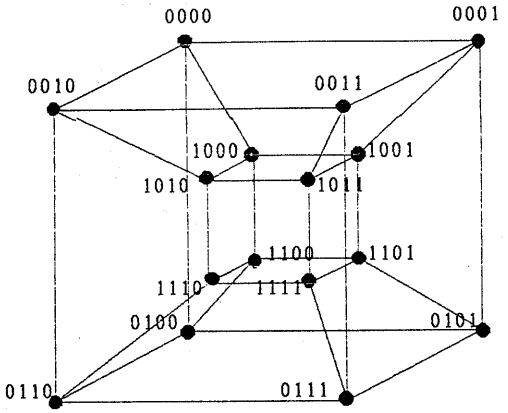

図１　４次元キューブのプロセッサ番号

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

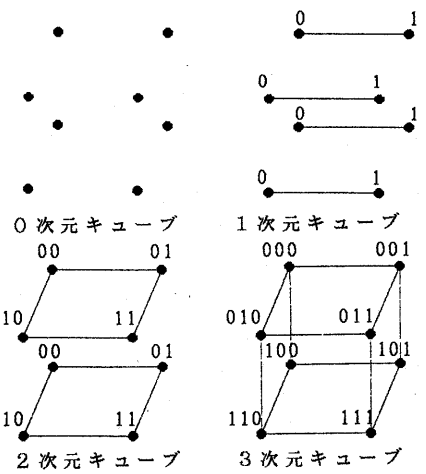

図２　３次元キューブの再帰的構成

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[3]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[3]。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。いっぱ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。仮想のタスクをつけ加えることによっ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[4]を採用する。カーディナリティーとは、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

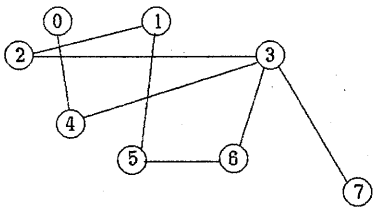

図３　ＴＩＧの例

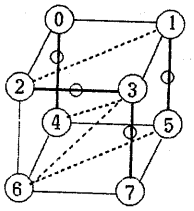

図４　図３のＴＩＧに対するアロケーション１

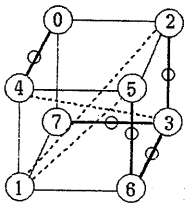

図５　図３のＴＩＧに対するアロケーション２

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^m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,....,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て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クについて、両者でのプロセッサ番号が同じであるアークの数が最大になるように上記の操作を行なう。

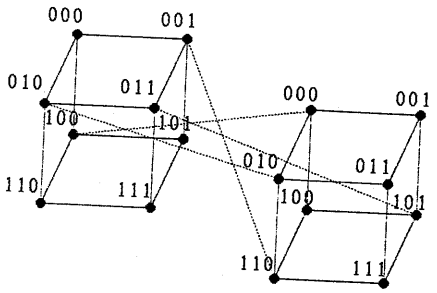

図６　３次元キューブの結合

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本のアークが存在している。その４本のアークがつながっているプロセッサ番号を以下に列挙する

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

この状態で、リンクと一致するアークの数は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クの両端のプロセッサ番号は、

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

となり、リンクと一致するアークの数は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ク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ク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ク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、アーク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 $C_{k,i}(i=0,1,2,..,M)$ 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} を重みとするアークで結合する。ただし、W_{ij} がゼロの時にはアークとはしない。

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。

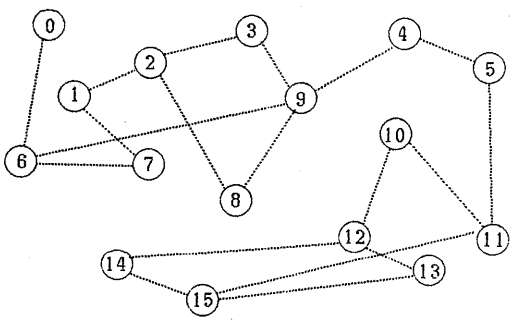

図７　ＴＩＧの例

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,...,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。

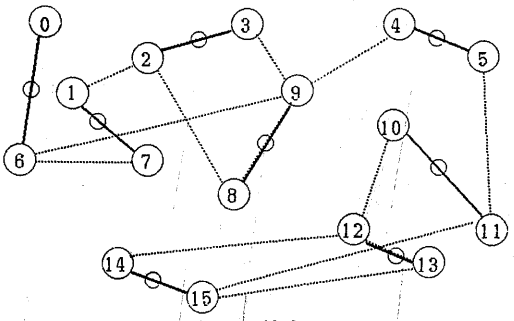

図８　図７のＴＩＧに対する１次元キューブの構成

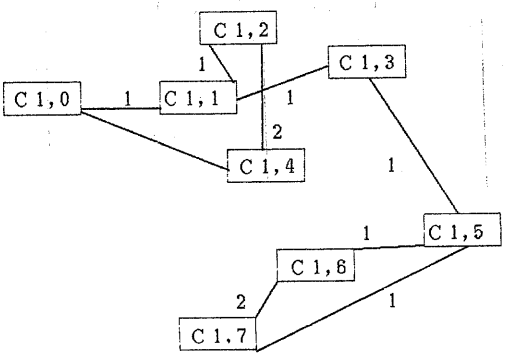

図９　段階１での重み付きグラフ

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。

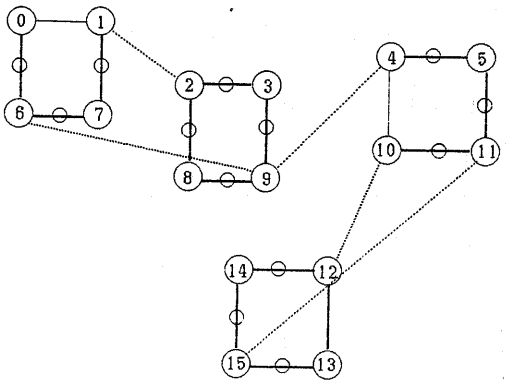

図１０　図７のＴＩＧに対する
２次元キューブの構成

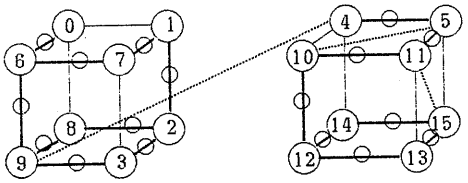

図１１　図１７のＴＩＧに対する
３次元キューブの構成

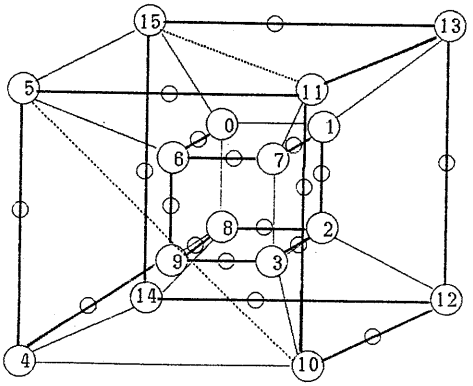

図１２　図７のＴＩＧに対する
４次元キューブの構成

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。

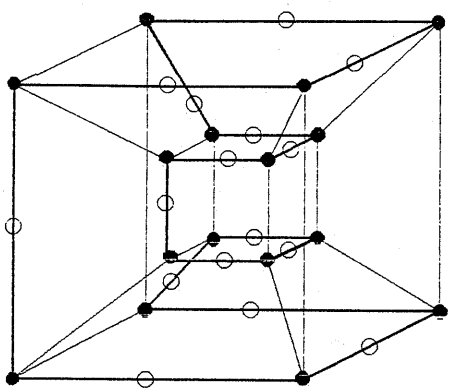

図１３　本手法によるリング状ネットワーク
のアロケーション

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換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。

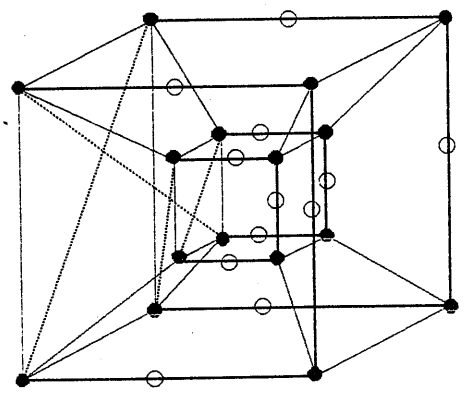

図１４　ノード交換によるアロケーション

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[5]も今後考察していく。

［参考文献］

[1] G.Fox, "Solving Problems on Concurrent Processors,"
Prnetice 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