ソフトウェア基礎論 29−5
（1989. 5. 18）

通信距離の評価に基づく ハイパーキューブ計算機のタスクアロケーション手法

A TASK ALLOCATION METHOD FOR HYPERCUBE USING COST FUNCTION OF THE TOTAL COMMUNICATION DISTANCE

堀池 聡
Horiike Satoshi

三菱電機（株）
Mitsubishi Electric Corp.

あらまし 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。並列計算機上で実行されるタスクが複数のサブタスクに分割され、各サブタスクをノード、サブタスク間の依存関係がエッジとしてタスクグラフとして表されているとする。そのグラフの各ノードをハイパーキューブ計算機のプロセッサにそれぞれ割り当てたときの各エッジに対応する通信距離の和をコスト関数としている。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、アルゴリズムも再帰的な複数の段階からなる。各段階kではグラフ理論のマッチングアルゴリズムを活用し、k次元のハイパーキューブにマッピングする。

Abstract A new task allocation method for hypercube is proposed. Task graphs are expressed by task nodes and communication edges. Task allocation problem is the mapping of task nodes onto processors so as to minimize the communication overhead. The cost function of the total communication distance is adopted in the proposed method. When the dimension of hypercube is n, proposed method is composed of recursive n stages. At each stage k, matching algorithms is applied to the result of the previous stage k-1, then the task graph is mapped onto k dimensional hypercubes.

1. まえがき

ハイパーキューブ[1]はn次元超立方体の頂点に処理要素、辺にリンクを配置するネットワーク構造である。この構造は、並列計算機に要求される性能、例えばH／Wコスト、通信距離、柔軟性、耐故障性等の点で優れている。すでに研究対象として注目を集めているだけでなく、NCube[1]、コネクションマシン[2][10]、iPSC等の商用の並列計算機にも採用されている。本稿では、このハイパーキューブ結合の並列計算機におけるタスクマッピング（アロケーション）問題を扱う。

並列計算機で実行させるタスクがいくつかのサブタスクから構成されると考える。1つのサブ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、あるサブタスクの実行は他のいくつかのサブ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依存する。結局並列計算機の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]。本稿で提案する手法はハイパーキューブの構成が再帰的に定義できることを利用しており、アルゴリズムそのものも再帰的な複数の段階からなる。そして各段階ではグラフ理論におけるマッチングのアルゴリズムを活用している。

[11]ではカーディナリティをマッピングのコスト関数とした手法を示した。今回の報告ではタスクグラフをマッピングした後のタスク間の通信距離の総和をコスト関数としている。このコスト関数を採用した結果、評価方法が並列計算機の実際の動作に近くなり、マッピングアルゴリズムのステップ数も短くなっている。

第２章でグラフ理論のマッチングアルゴリズムを簡単に紹介する。第３章でハイパーキューブ構造を説明する。第４章でタスクマッピングの問題を定義する。第５章でハイパーキューブ構造の計算機におけるタスクマッピング手法を提案する。第６章では他手法[11]との比較を行なう。

2. グラフ理論における最大マッチング問題

本稿で提案するマッピング手法はグラフ理論におけるマッチングのアルゴリズムを活用している。ノードの集合をＶ、エッジの集合をＥとして、グラフ（Ｖ；Ｅ）のマッチングとは端点を共有しないエッジの集合である。最大マッチング問題はグラフの形状の観点から次の４種類に分類できる。

・２部グラフ

２部グラフとはすべてのエッジの一方の端点がノード集合V^+に属し、他端がV^-に属するようにノード集合Ｖを２つの集合V^+とV^-に分けることのできるグラフである。２部グラフの最大マッチング問題はエッジの本数が最大となるマッチングを見い出す問題である。

・一般グラフ

一般のグラフにおいてエッジの本数を最大にするマッチングの問題である。

・重み付き２部グラフ

２部グラフのエッジに重みが付加したグラフである。重み付き２部グラフの最大マッチングは、エッジの重みの総和が最大となるマッチングを見出す問題である。

・重み付き一般グラフ

一般グラフのエッジに重みが付加しているグラフにおいて、エッジの重みの総和が最大となるマッチングの問題である。

いずれの問題に対しても多項式オーダーで解を求めるアルゴリズムが既に知られている[8]。以上のマッチング問題において全てのノードがいずれかのマッチングのエッジの端点である場合を完全マッチングという。

3. 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、他のリング状ネットワーク、格子状ネットワークなどに比べて極めて優れている[2]。

ｎ次元ハイパーキューブネットワーク（以下ではｎ次元キューブと略称する）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。

このハイパキューブネットワークは以下に述べるように再帰的に構成される。

まず０次元キューブは単一のプロセッサで構成され、そのプロセッサ番号を０とする。この状態を初期状態とする。

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とする。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１を加える。プロセッサ番号のｎ－１ビット以下が同じプロセッサは両方のキューブに必ずある。全部でＮ組あるから、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４次元のハイパーキューブ

が構成される過程を図１に示す。

また、以上のような再帰的な定義に基づいて構成されるハイパーキューブでは、隣接する２つのプロセッサの番号は２進表現で１つのビットだけ値が異なるという性質がある。

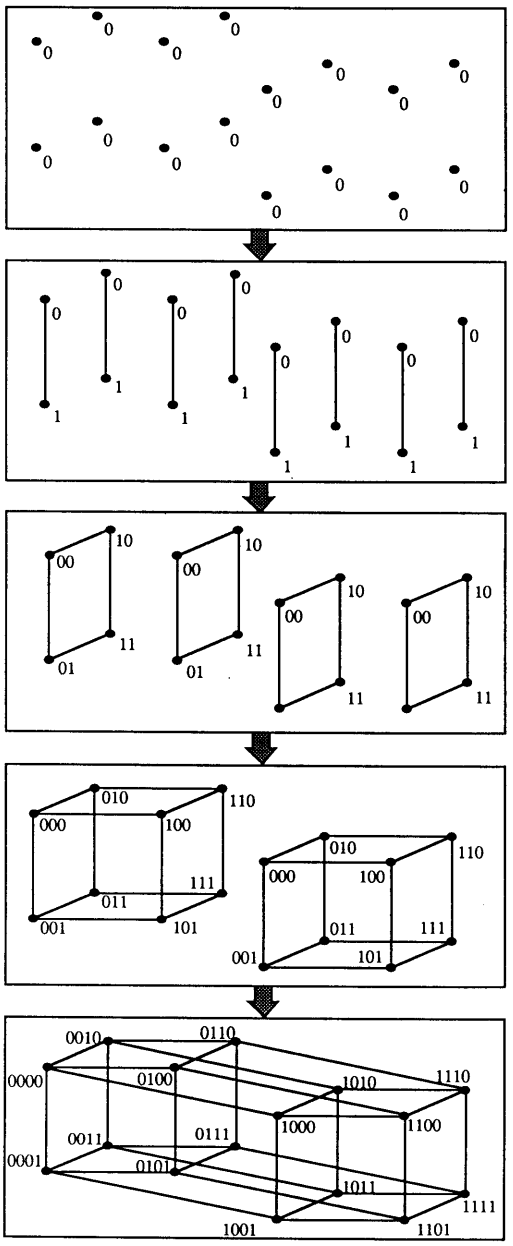

図１　４次元キューブの再帰的構成

４．タスクマッピング問題の定義

４．１　タスクグラフ

並列計算機で実行させるタスクを複数のサブタスクに分割する。サブタスクは１つのプロセッサに実行させる単位である。どのプロセッサにどのサブタスクを実行させるかを決定する問題をタスクマッピング問題と呼ぶ。

通常の場合、あるサブタスクを実行するには他のサブタスクとの通信が必要になる。サブタスクをノードとし、通信を行なうサブタスクに対応する２つのノード間をエッジで結合すれば、グラフが構成できる。このグラフをタスクグラフＴ＝（Ｖ；Ｅ）と呼ぶ。タスクグラフのノードはタスクノードと呼び、エッジは通信エッジと呼ぶ。並列計算機のネットワークもプロセッサをノード、リンクをエッジに対応させることによって１つのグラフに対応させることができる。こちらはそのままプロセッサおよびリンクと呼ぶこととする。

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マッピング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、高速のマッピング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マッピング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]。前者ではタスクの通信エッジ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。後者では通信エッジはプロセス間の通信を表す。一般に通信は双方向に行なわれるので、このグラフの通信エッジは無向である。

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。タスクの数がプロセッサの数より少ないときは仮想のタスクをつけ加えれば両者を同数にできる。しかし逆の場合、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る。マッピング問題における1つの課題であり、今後検討していく。

4．2　コスト関数

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。

例えば[4],[11]ではカーディナリティーをコスト関数としている。カーディナリティは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の通信エッジの数で定義される。

リンク結合の計算機において、リンクに一致していない通信エッジの通信はいくつかのプロセッサノードを中継して行なわれる。ここで通信を実現するために経由するリンクの最小数を距離と呼ぶ。

一般にリンク結合の並列計算機ではこの距離が長いほど通信が遅くなる。ところがカーディナリティの評価ではリンクと一致しない通信エッジはすべて同等に扱うことになる。そこで本稿ではあるマッピングに対してＴＩＧの各通信エッジの通信を実現するときの距離の総和をコスト関数とする。つまりマッピングｆのコスト関数Ｃｆを次式で定義する。

$$Cf = \sum_{(v_i, v_j) \in E} dist(f(v_i), f(v_j)) \qquad (1)$$

ここで関数$H_{ij}(a,b)$を値ａの第ｉビットと値ｂの第ｊビットを比較し、値が等しいときに0、等しくないときに1を与える関数と定義する。ハイパーキューブ構造では隣接するプロセッサ番号は2進表現において1ビットだけ異なる。よって2つのプロセッサP_sとP_rの距離D_{sr}は2進表現において異なるビットの数に等しくｎ次元キューブでは

$$D_{sr} = \sum_{i=0}^{n-1} H_{ii}(P_s, P_r) \qquad (2)$$

と表現できる。

総距離Ｃｆは最小化する問題であるが、説明の便宜上ゲインの考え方を導入し、最大化する問題にする。

通信エッジ$Eij=(v_i, v_j)$のゲインEij_rは、ネットワークの最大距離（ハイパーキューブでは次元ｎ）とその通信エッジがマッピングされたときの距離の差

$$Eij_r = n - dist(f(v_i), f(v_j)) \qquad (3)$$

で定義する。ＴＩＧに対するゲインは通信エッジのゲインの総和

$$G_r = \sum_{(v_i, v_j) \in E} \{n - dist(f(v_i), f(v_j))\} \qquad (4)$$

で定義する。以下ではG_rを単にゲインと呼びE_rは通信エッジのゲインと呼ぶ

例として図2に示すＴＩＧのマッピングのゲインについて説明する。図2のＴＩＧはノード数4であり、2次元のキューブに対してマッピングする。図3と図4に2通りのマッピング結果を示す。点線は図2における通信エッジを示し、各通信エッジに付した値はエッジのゲインである。図3のマッピングではゲインは3（=1+1+1+0）であり、図4では2（=1+1+0+0）であるから図3の方がよいマッピングと評価される。

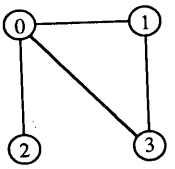

図2　ノード数4のＴＩＧ

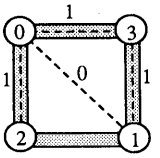

図3　図2のＴＩＧに対するマッピング1

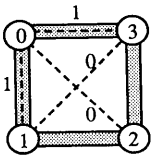

図4　図2のＴＩＧに対するマッピング2

5. マッピング手法

3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おり、マッピングアルゴリズム自体が再帰的なアルゴリズムである。

N個のノードは、0次元から始まって順番に1次元上のキューブに再帰的にマッピングされ、n回でn次元キューブにマッピングされる。

初期状態として、TIGの各タスクノードi (i=0,1,..,N-1) が0次元キューブ $C_{0,i}$ (i=0,1,2,..,N-1)に属し、それぞれプロセッサ $P_{0,i,0}$ にマッピングされているとする。

各段階k (k=1,2,..,n-1) でTIGの各タスクノードはk次元キューブ $C_{k,i}$ (i=0,1,..,M-1)のいずれかに属し、その中のプロセッサ $P_{k,j,i}$ (i=0,1,..,K-1) にマッピングされる。ここで

$$n = m + k \tag{5}$$

$$K = 2^{k} \tag{6}$$

$$M = 2^{m} \tag{7}$$

としている。

この状態を元にして、段階k+1ではk次元キューブを2個づつ組み合わせることによって各タスクノードをk+1次元キューブ $C_{k+1,i}$ (i=0,1,..,M/2-1) のプロセッサ $P_{k+1,i,j}$ (j=0,1,..,2K-1)にマッピングする。

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。

段階kでTIGのある通信エッジについて、その両端のノードが同じキューブ $C_{k,i}$ の中にマッピングされている時その通信エッジはキューブ $C_{k,i}$ に属するという。k−1段階ではどのキューブにも属さず、k段階の処理によってあるk次元キューブに属するようになった通信エッジを $E_{k,l}$ (l=0,1,..,L_k-1) とする。本手法ではk次元キューブへのマッピングの状態を保存してk+1次元キューブへマッピングする。よって、いったんキューブに属したアークは、その段階から最後の段階までキューブに属する。通信エッジ $E_{k,l}$ の両端のプロセッサを P_{ls}、P_{le}とする。同様に、各段階で P_{ls}、P_{le}の距離は最後まで変わらないので通信エッジ $E_{k,l}$ のゲインは段階kで確定しており、$G_{k,l}$ で表わすと

$$G_{k,l} = n - dist(P_{ls}, P_{le}) \tag{8}$$

となる。$G_{k,l}$ を用いて段階kでのゲイン G_k を次式で定義する。

$$G_k = \sum_{k=0}^{k} \sum_{l=0}^{L_k-1} G_{k,l} \tag{9}$$

これは段階kの時点でキューブに属する通信エッジに対してだけゲインを計算している。最終的なゲインGは次式で表わすことができる。

$$G = \sum_{k=0}^{n-1} \sum_{l=0}^{L_k-1} G_{k,l} \tag{10}$$

ところで式(10)は

$$G_K = G_{k-1} + \sum_{l=0}^{L_k-1} G_{k,l} \tag{11}$$

と表わすことができる。

本稿で提案する手法は、次元を1つ上げるときに式(11)の第2項をできる限り大きくするようなキューブの組合せを見つけることによって、最終的なゲイン(10)を大きくする。以下ではその方法について詳述する。

手続1. ハイパーキューブ間の結合法

段階k+1の処理を行なう前に存在するk次元キューブを $C_{k,i}$ (i=0,1,..,M-1)とする。まず、任意の2個のk次元キューブ $C_{k,i}$ と $C_{k,j}$ に着目し、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時に新しくキューブに属する通信エッジのゲインの和を最大にする方法について述べる。

TIGのキューブに属さない通信エッジの内で、一端のノードがあるキューブ $C_{k,i}$ のプロセッサにマッピングされ、他端のノードが別のキューブ $C_{k,j}$ のプロセッサにマッピングされているS本の通信エッジを E_s (s=0,1,..,S-1)とする。各通信エッジ E_s について $C_{k,i}$ 側のプロセッサ番号を $P_{s,i}$ で表わし、$C_{k,j}$ 側の番号を $P_{s,j}$ で表わす。$C_{k,i}$ と $C_{k,j}$ の2個の

k次元キューブにはそれぞれ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。段階k＋1の処理以前に与えられらた各プロセッサの番号をそのまま用いて結合したとしても、この2つのk次元キューブを結合して新たにキューブに属したS本の通信エッジのゲインの総和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。ただし、k次元キューブに属する通信エッジに対するゲインの値は変えてはならない。この制約を満足するには、プロセッサ間の相対的な位置関係（隣接関係）を保たなければならない。以下の2つのビット操作はプロセッサの相対的な位置関係を保証する。

ビット操作1.
　ビットの位置の並び替え
ビット操作2.
　ある位置のビットを反転

上記の2操作を用いて新たに加わる通信エッジのゲインの総和を最大にする方法について述べる。

まず4章で定義したビット比較の関数Hを用いて次の関数T(q,r)を定義する。

$$T(q,r)=\sum_{l=0}^{L-1} H_{qr}(P_{li},P_{lj}) \qquad (12)$$

この関数はキューブiのプロセッサ番号の第qビットを第rビットに移動したときに、S本の通信エッジの第rビットの内で一致するものの数を与える。そして、もしこの値がS/2より小さいならば、その位置のビットを反転すればさらに一致度が大きくなる。そこでT(q,r)を次式で定義する。

$$U(q,r)=\max\{T(q,r),S-T(q,r)\} \qquad (13)$$

ここで次のような2部グラフ（V^+,V^-; E）をつくる。V^+ のノードq（q=0,1,.,k-1）はプロセッサ番号を変更する前のビット位置を表わし、V^- のノードr（r=0,1,.,k-1）はプロセッサ番号の変更後のビット位置を示す。V^+ の個々のノードとV^- の個々のノードの間をK^2 本の重み付きエッジで結合する。ここでノードqとノードrを結ぶエッジの重みはU(q,r)である。

このグラフに最大マッチングのアルゴリズムを適用する。得られる結果は、キューブ$C_{k,i}$のプロセッサ番号について、ビット位置の変更方法を示す。また、最大マッチングで採用されたエッジの値が式（13）のマックス計算で後者の値が選ばれた場合には新しいビット位置に移すときに反転しなければならない。

以上の手続きによって2つのキューブを組み合わせたときに新たにキューブに属する通信エッジのゲインを最大にできる。

図5における2個の3次元キューブiとj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本の通信エッジが存在している。その4本の通信エッジがつながっているプロセッサ番号を表1に列挙する。

U(p,q)の値を計算すると

$$\begin{array}{lll} U(0,0)=3, & U(0,1)=2, & U(0,2)=4, \\ U(1,0)=3, & U(1,1)=2, & U(1,2)=2, \\ U(2,0)=2, & U(2,1)=\underline{3}, & U(2,2)=\underline{3} \end{array} \qquad (14)$$

となる。上式で値に下線を引いているものはビットを反転してえられる値だからである。この値を重みとして2部グラフをつくると図6となる。このグラフに2部グラフのマッチングアルゴリズムを適用すると、

ノード0（V^+）とノード2（V^-）
ノード1（V^+）とノード0（V^-）
ノード2（V^+）とノード1（V^-）

という組合せが得られる。つまり、キューブi側のプロセッサ番号は、0位のビットを2ビット目に移し、1位のビットを0ビット目に移し、2位のビットを反転して1ビット目に移して新しい番号とする。新しいプロセッサ番号を用いたキューブの図6と、通信エッジの2個のプロセッサの表を図7に示す。この結果、3次元キューブiとjを組み合わせて4次元キューブを構成したとき、新しくキューブに属する4本の通信エッジのゲインの総和は10となる。

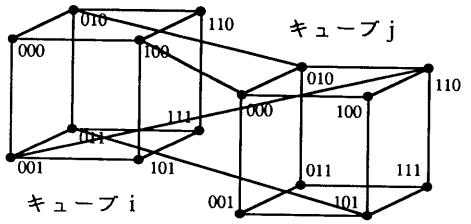

図５　３次元キューブの結合

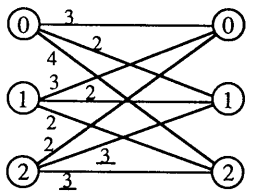

図６　ビットの変換方法を求める２部グラフ

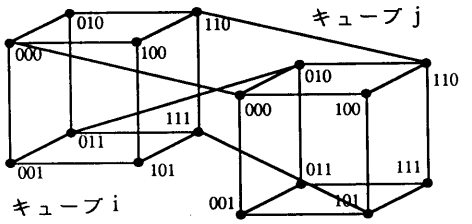

図７　番号変更後の２個のキューブ

表１　図５における通信エッジ
（斜線部は一致しているビット）

通信エッジ	プロセッサ（ｉ側）	プロセッサ（ｊ側）
0	0 0 1	1 1 0
1	0 1 1	1 0 1
2	0 1 0	0 1 0
3	1 0 0	0 0 0

表２　変換後のプロセッサ番号
（斜線部は一致しているビット）

通信エッジ	プロセッサ（ｉ側）	プロセッサ（ｊ側）
0	1 1 0	1 1 0
1	1 1 1	1 0 1
2	0 1 1	0 1 0
3	0 0 0	0 0 0

手続き２．ハイパーキューブの組合せ

段階ｋでＮ個のタスクノードはＭ個のｋ次元キューブにマッピングされている。このＭ個のキューブのすべての２個の組合せに対し手続き１の計算をする。この組合せは$M(M-1)/2$組ある。この中から全体としてゲインの増分G_kが最大となるように$M/2$組の組合せを見つけなければならない。ただしこの組合せでは同じキューブを共有してはならない。

この目的のためにまず次のようなノード数Ｍ個の重み付きグラフをつくる。

このグラフではｋ次元キューブ$C_{k,i}(i=0,1,2,..,M-1)$をノードｉとみなす。手続き１で計算されたキューブ$C_{k,i}$とキューブ$C_{k,j}$を組み合わせたときに増えるゲインの値をW_{ij}とする。ノードｉとノードｊの間は、W_{ij}を重みとするエッジで結合する。ただし、W_{ij}がゼロの時には計算の負担を軽減するためにエッジの結合は省略する。

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においてゲイン値の増分を最大とするキューブ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。

手続き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で組合せ方が決定した。

プロセッサ番号についてはまず、手続き１の結果にしたがって各組合わせについて片方のｋ次元キューブのプロセッサ番号を変更する。そしてそのキューブのプロセッサ番号の第ｋビットを１にすればよい。

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。

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）

ＴＩＧ中のタスクノードｉ（ｉ=0,1,...,N-1）は0次元キューブ$C_{0,i}$のプロセッサ$P_{0,i,0}$にマッピングされているとする。

ｋ＝１とする。

ステップ２．（段階ｋの処理）

［手続き１］

ｋ次元キューブの全ての組合せに対してゲインの増分値W_{ij}（i=0,1,..,M-1, j=i+1,i+2,..,M-1）を計算する。

［手続き２］

$G_{k,i}$（i=0,1,..,M-1）をノードとし、W_{ij}をエッジの重みとする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、キューブの組合せを決める。

［手続き３］

処理１と処理２によって得られた結果にしたがい、各タスクノードをキューブ$C_{k+1,i}$（i=0,1,2,..,M/2-1）のプロセッサ$P_{k+1,i,j}$（j=0,1,..,2K-1）にマッピングする。

ステップ３．（終了判定）

ｋ＝ｎならば終了する。

ｋ＜ｎならばｋ＝ｋ＋１としてステップ２を繰り返す。

図８のＴＩＧに対して本手法を適用し、処理の概要を図示する。このＴＩＧのノード数は16で４次元キューブにマッピングされる。

図９は上から順番に各段階での処理結果を表わす。円の中の番号はタスクノードの番号を示す。太い線はキューブのリンク、破線はキューブに属している通信エッジ、実線はキューブにまだ属していない通信エッジを表わす。

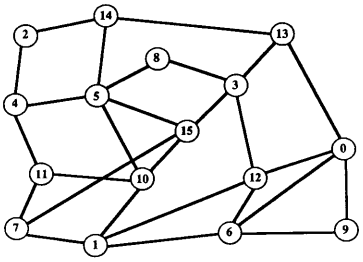

図８　ＴＩＧの例

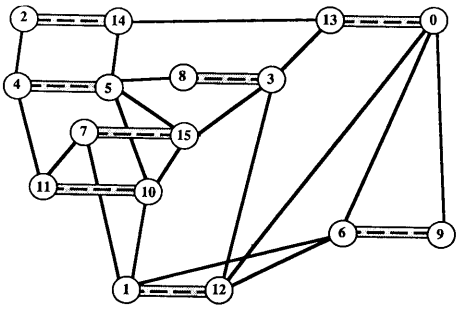

図９(a)　ｋ＝１

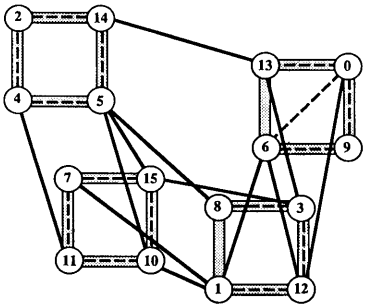

図９(b)　ｋ＝２

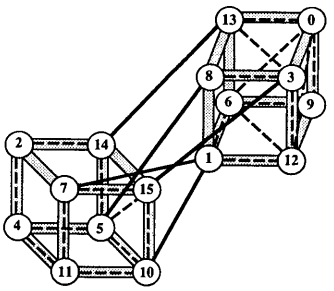

図９(c)　ｋ＝３

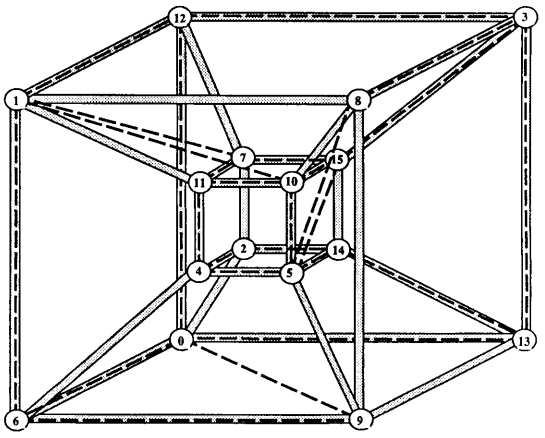

図９(d)　ｋ＝４

図９　各次元でのマッピング結果

6．比較

提案手法の性能を文献［11］の手法と比較する。この手法は本手法と同様に、任意の構造のＴＩＧをハイパーキューブ構造の計算機にマッピングするもので、通信エッジの通信距離の総和をコスト関数としている。

タスクノードをＮ個とするとこのアルゴリズムでは、１つのタスクノードを１つのプロセッサにマッピングするというステップをＮ回繰り返す。各ステップではゲイン値の増分が最も大きくなるタスクノードとプロセッサの組合せを以下の候補の中から選ぶ。

［タスクノードの候補］
マッピングするプロセッサが決まっていないすべてのタスクノード

［プロセッサの候補］
タスクノードがマッピングされていないすべてのプロセッサ

6．1　マッピング結果の比較

比較はノード数を32として6種類の構造のグラフを対象とした。それらは、ハイパーキューブ、バタフライ、メッシュ、２進木、リング、及び文献[4]のＦＥＭ(Finite Element Method)のグラフである。これらのうちでハイパーキューブ以外のグラフを図10に示す。

比較の計算は次のように行なった。まずＴＩＧの構造だけを与え、タスクノードの番号は乱数によって設定する。そして、本手法と提案手法を適用する。表３にはこの繰り返しを1000回行い、ゲインの値が大きかった、つまり総距離数が短かった回数をそれぞれ示している。

ハイパーキューブ以外のＴＩＧではすべて本手法がよい値を示しており、本手法が一般的によいマッピングを与える。

ハイパーキューブ構造のＴＩＧでは、従来手法によれば最適マッピングが得られるため、表では本手法が０となっている。逆にリング構造のＴＩＧでは本手法が最適マッピングを与えるので従来手法が０である。

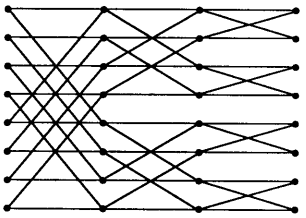

図10(a)　バタフライ

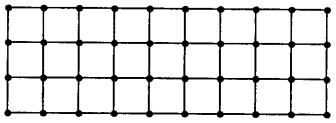

図10(b)　メッシュ

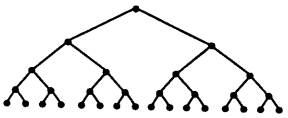

図10(c)　２進木

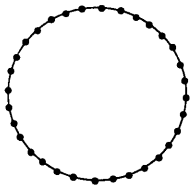

図10(d)　リング

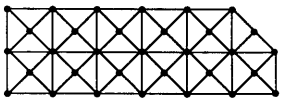

図10(e)　ＦＥＭ

図10　比較の対象としたグラフ

表３　マッピング性能の比較

ＴＩＧの構造	提案手法	従来手法
ハイパーキューブ	0	7590
バタフライ	9563	285
メッシュ	5514	4004
２進木	4973	2714
リング	8058	0
ＦＥＭ	7450	1910

6．2　実行時間の比較

両手法とも多項式オーダーの処理時間のアルゴリズムである。実際の計算時間を比較するためにn次元キューブ構造のＴＩＧを対象とし、ワークステーションＳＵＮ上で両手法を実行した。ただし従来手法については、実行効率をよくするために、タスクノードとプロセッサを以下の候補から選ぶようにアルゴリズムを変更した。

［タスクノードの候補］
マッピングするプロセッサがまだ決まっていないタスクノードのうちで、すでにマッピングされているタスクノードのいずれかと通信エッジで結ばれているもの

［プロセッサの候補］
タスクノードがまだマッピングされていないプロセッサのうちで、すでにタスクノードにマッピングされているプロセッサとリンクで結ばれているもの

表4に6次元から10次元までのキューブに対するマッピングの処理時間を示す。明かに本手法が処理に必要な計算時間が短いことを示す。

表4　ハイパーキューブ構造のＴＩＧに対する処理時間の比較
（実行計算機：ＳＵＮ4）

次元	提案手法	従来手法
7	2秒	13秒
8	10秒	110秒
9	48秒	963秒
10	226秒	8246秒

7．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マッピング手法を提案した。いくつかのグラフについて従来手法と比較し本手法が優れ、計算時間も短いことを示した。

タスクグラフを作成するときのタスクの分割手法、あるいはプロセッサが少ないときのタスクのグループ化が今後の課題である。

［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

[9] Francine Berman,"Experience with an Automatic Solution to the Mapping Problem",

[11]W.K.Chen, E.F.Gehirnger, "A Graph-Oriented Mapping Strategy for a Hypercube",Proceedings of the third Conference on Hypercube Concurrent Computers and Applications,pp200-209, (1988).

[10]堀池、Walter Bender、”コネクションマシンによる英文文章間の相関値計算”、信学技法、CPSY88-5,pp.31-38(1988).

[11]堀池、”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”、信学技法、COMP88-64,pp.1-8(1988).