修士学位論文

並列ネットワークにおける負荷分散方式に関する研究

東北大学大学院工学研究科
電気・通信工学専攻

遠藤 貴之

目 次

図目次

表目次

第1章

はじめに

1.1 研究の背景

世界初のコンピュータと言われる ENIAC (Electronic Numerical Integer And Calculator) が 1946 年に作られてから約 50 年が経つが, その間に計算機は目覚しいほどの進歩を遂げた. 集積化回路技術の発達により, 嘗ては部屋一杯の規模を要した計算機と同等の回路を, 今では一つの VLSI チップに収める事が可能である. また, 処理速度の面でも飛躍的な向上を見せ, 現在の計算機は昔とは比べものにならない程大規模なデータを高速に処理出来る. しかし, 画像処理, 信号処理と言った大規模なデータの処理や, 流体力学や気象予測に見られるような複雑な科学計算など, 使用者の用途に応じて, 計算機の処理速度はさらなる向上を求められている.

計算機の高速化の方法としては, クロックの高速化,GaAs などの材料を使った新たな素子の使用, 並列処理プロセッサなどの, 新たなアーキテクチャの使用などが挙げられるが, 一つの計算機の速度や性能を向上させるという方法は, 近い将来に物理的な限界を迎える事が予測されるため, これからの高速化技術には, 並列処理アーキテクチャによる高速化が重要であると考えられる. 並列処理とは, 処理すべき仕事を複数の独立した仕事に分割し, 複数の計算機で同時に処理することで, 全体の処理速度を向上させる方法である. 並列処理プロセッサでは, 比較的単純な計算を行うプロセッサを二分木や網の目状といった通信性の高い構造に多数配置して, 各計算機に仕事を分配して並列処理を行う.

並列処理システムで効率良く処理を実行するには, 負荷分散が非常に重要になってくる.

負荷分散とは, 並列処理システム上に割り当てられた仕事の不均衡によって生じるシステム全体のパフォーマンスの低下を, 過負荷な計算機の負荷を別の計算機に移送することで防ごうとするものである. 負荷分散を適切に行うことで, 並列処理システム全体の利用率が向上し, 処理速度の向上が期待出来る.

並列処理プロセッサにおいては, ワークステーションによって構成される分散システムなどと異なり, システムが同一のプロセッサで構成されているといった特徴から, 負荷が各プロセッサ間で完全に分散される事が望ましい. また, 通信性が高いという特徴もあり, それらを生かしたアルゴリズムは有効性が高いと思われる.

負荷分散の方式には、**静的負荷分散**と**動的負荷分散**の二つの方式がある。静的負荷分散は, あらかじめ負荷の割り当てを決めておくもので, 負荷が正確に見積れる場合には適しているが, システム稼働時に負荷の変動がある場合には負荷の不均衡が避けられない.

一方, 動的負荷分散は仕事の処理結果や処理状況に応じて負荷の分配を決定する方式であり, 負荷の割り当てを 1 つのプロセッサが集中的に行う方式と各プロセッサが分散的に行う方式がある. 前者は制御や管理が簡単であるという利点があるが, 耐故障性や通信量という観点から見ると問題がある. 後者は並列処理システムの特徴を生かすもので, 多くの研究が発表されている.

動的負荷分散の方式については様々な手法が提案されているが, その中の一つに**反復改良 (iterative)** と呼ばれるものがある [3]. 反復改良とは, 一つのアルゴリズムを繰り返し実行する事で, 不均衡な負荷をしだいに均衡な状態に近づけていく手法で, 特に, diffusion 法や, Cheng-Zhing Xu らによって提案された generalized dimention exchange(GDE) 法といった手法は, [1] [2]

- 比較的単純なアルゴリズムの繰り返しで, 完全に近い負荷分散を行う事が可能.
- 各プロセッサはグローバルな情報を用いず, 近隣のプロセッサの情報のみを参照して負荷分散を行うため, 対故障性が強い.

など, 並列処理プロセッサにとって望ましい利点が挙げられる.

しかし, 反復改良アルゴリズムに関する過去の研究は hypercube や mesh,torus といった一部のネットワーク構造についてのものが多く, 古くから並列処理の分野で研究対象とされている tree や, また CCC といった構造について研究した物は少なく, それらのネットワークに対する有効性についても殆ど言及されていない.

1.2 本研究の目的

本研究では, 並列処理プロセッサの負荷分散アルゴリズムとして, 動的負荷分散の手法である反復改良アルゴリズムに注目し, tree, CCC 構造ネットワークによって構成される並列処理システムの負荷分散に適したアルゴリズムの検討を行い, これらの構造において最適に負荷分散を行うための条件を求める.

さらに, 各ネットワークの負荷分散において生じる特徴を調べ, それらに対して考察し, 改良したアルゴリズムの提案と検証を行う.

1.3 本論文の構成

本論文の構成は以下の通りである.

第 1 章は序論であり、本研究の背景と研究の目的を述べる。

第 2 章は反復改良アルゴリズムについて述べ, 並列処理ネットワークの負荷分散という観点から各アルゴリズムを検証する.

第 3 章は並列処理において代表的なネットワーク構造である tree 構造のうち, 特に完全二分木構造の負荷分散についてシミュレーション実験と理論的解析を行い, 検証と考察を行う.

第 4 章は hypercube 構造の変形である CCC(Cube-Connected Cycle) 構造の負荷分散についてシミュレーション実験と理論的解析を行い, 検証と考察を行う.

第 5 章は本研究のまとめを行なう.

第 2 章

反復改良アルゴリズム

2.1 はじめに

動的負荷分散の手法は, 古くから多くの研究者によって研究されている. 代表的なものとしては, 過負荷なプロセッサから負荷を移動する**送り手主導**, あるいは負荷が一定以下であるプロセッサに負荷を集める**受け手主導**アルゴリズム [8], **Gradient Model** (GM) [9], また負荷の移動を確率モデルにより予測するもの [10] [11] [13] などが挙げられる.

これらの手法は用途や目的, あるいは条件によって研究されてきたものであるが, 第 1 章で並列処理プロセッサの負荷分散において望ましいと述べた, 完全に近い負荷分散を行うアルゴリズムという点では該当しない物が多い.

反復改良アルゴリズムの多くは, この目的に合致するアルゴリズムであるが, 中には並列処理ネットワークでの実行に不適である物も存在し, 並列処理ネットワークの負荷分散に適用するにあたっては, 各アルゴリズムの特徴を考慮しなければならない.

本章では, 反復改良アルゴリズムとして diffusion 法, dimention exchange (DE) 法, generalized DE(GDE) 法, Simulated Annealing を挙げ, 各アルゴリズムの特徴から, 並列処理ネットワークの負荷分散に適したアルゴリズムについて検討する.

2.2 反復改良アルゴリズム

動的負荷分散, 特に反復改良アルゴリズムにおいては, プロセッサが持つ負荷は, 均等な大きさを持つ, 最小の負荷単位である**タスク**の集合により構成されると仮定される. 負荷の大きさはタスクの個数として, 負荷の移動はタスクをいくつ移動するかとして表現される. 以降, 本論文ではプロセッサの持つ負荷については上記の仮定を適用するものとする.

反復改良を利用した負荷分散アルゴリズムの代表的なものには, 以下のものがある.

2.2.1 diffusion 法

diffusion 法では, 並列処理プロセッサを構成する各プロセッサは, 式 2.1に従い, 自分と接続されている最近傍のプロセッサと負荷を交換する.

$$w_i^{t+1} = w_i^t + \sum_{j \in A(i)} \alpha_{ij}(w_j^t - w_i^t) \tag{2.1}$$

ここで, w_i^t とは, 式 2.1を t 回繰り返した時点での, i 番目のプロセッサが持つ負荷である. $A(i)$ は, プロセッサ i の最近傍のプロセッサの集合, α_{ij} は, プロセッサ i と j の間で交換する負荷の量を決める係数で, diffusion パラメータと呼ばれる. 図 2.1に, diffusion 法による負荷分散の様子を示す. 図の丸はプロセッサ, 中の数字はプロセッサが持つ負荷 (=タスク数) を表す.

中央の太丸が, 式 2.1におけるプロセッサ i とすると, $A(i)$ は四方のプロセッサになる. i は四方のプロセッサそれぞれと自分の負荷を比較して負荷を送るか受け取るかを決定し, 負荷の交換を行う.

diffusion 法は, この手順をプロセッサ全体で繰り返し行っていくことで負荷を均等にしていく. このアルゴリズムは, 一つのプロセッサが一回の動作で複数のプロセッサとの通信を行う場合は, ネットワーク全体で同時に負荷の交換を行うことが出来るが, 並列処理システムに使われるプロセッサは通常, 同時にただ一つだけのプロセッサとの通信が可能である物が多い.

この場合は, ネットワークの次元数分だけアルゴリズムを繰り返さなければネットワーク全体での負荷の交換を行う事が出来ない.

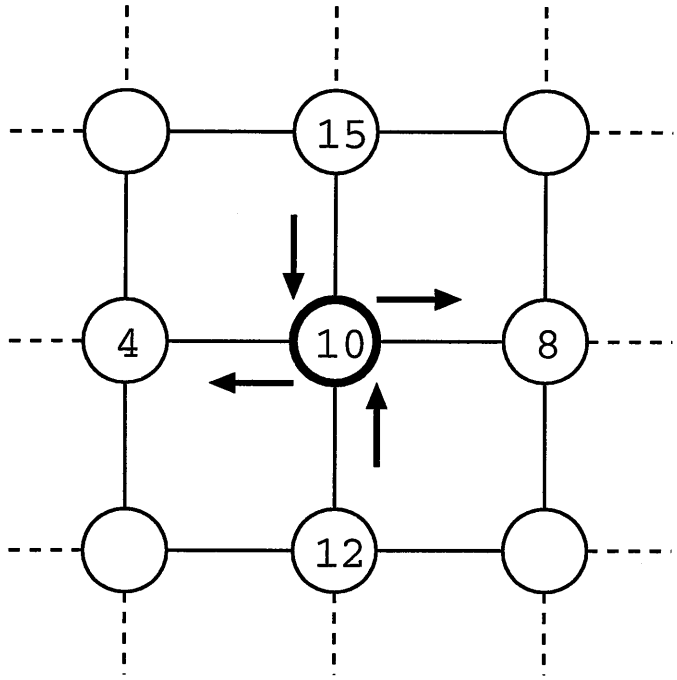

図 2.1: diffusion 法の負荷分散の様子

2.2.2 DE 法

DE(Dimention Exchange) 法は当初, hypercube のための負荷分散方式として考案されたアルゴリズムである. DE 法の負荷分散の様子を図 2.2に示す. 各々のプロセッサは x,y,z 軸のいずれか一方の隣接するプロセッサと, 負荷を半分ずつ交換する. 負荷の移動を式にすると, 式 2.2のようになる.

$$w_i = \frac{1}{2}(w_i + w_j) \tag{2.2}$$

そうして次々と, 全ての次元のプロセッサに対して負荷を半分ずつ交換するという作業を終えると, 負荷分散が完了する. hypercube の場合, この一連の手順を一回行う事で完全な負荷分散が可能である事が証明されている.

hypercube 以外の構造でも,**edge coloring** という概念の導入により,DE 法の適用が可能となる [4]. edge coloring とは, プロセッサ間を繋ぐ線に, 同じ番号の線が一つのプロセッサに複数繋がらないようにするという規則で番号をつけていくというものである. 幾つかの具体例を図 2.3, 図 2.4, 図 2.5に示す.

edge coloring の定義から, coloring により付けられた番号が同じ線同士は, 同時刻に通信を行っても, 全てのプロセッサはただ一つだけのプロセッサと通信をすることになるので, この番号の順に負荷の交換を行うと, hypercube の時に各次元毎に負荷の交換を行ったのと等価になる. edge coloring の番号全てで一回交換を行う事を **sweep** と呼び, アルゴリズムの収束速度は, 負荷が分散されるまでに何回 sweep が実行されたかで評価する.

edge coloring は, ring や mesh といった規則的な構造の場合はすぐに決定されるが, 一般には NP 完全問題であり, LAN により構築された分散システムのような場合, coloring の決定そのものが困難であったり, coloring が出来ても非常に多くの color を使ってしまったりすることがある.

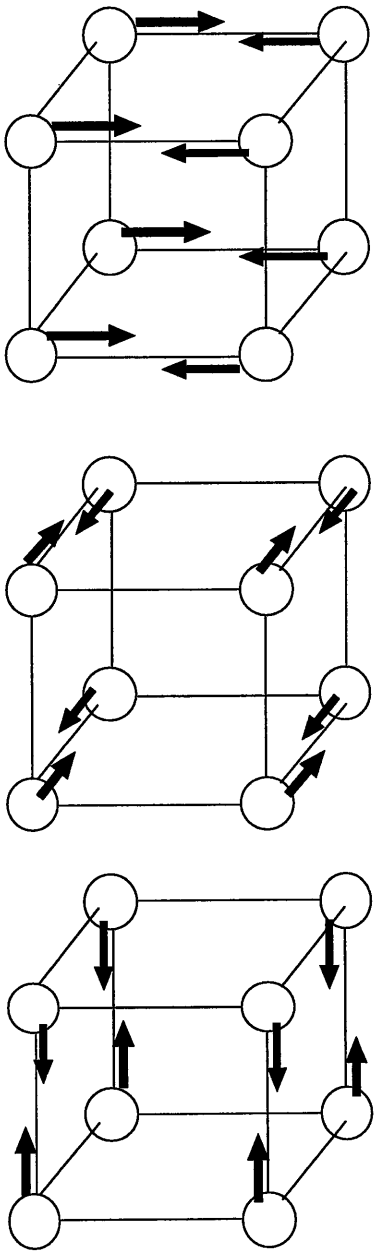

図 2.2: DE 法の負荷分散の様子

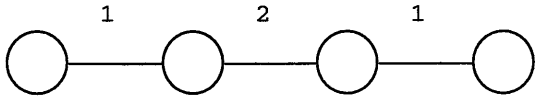

図 2.3: array 構造の coioring 例

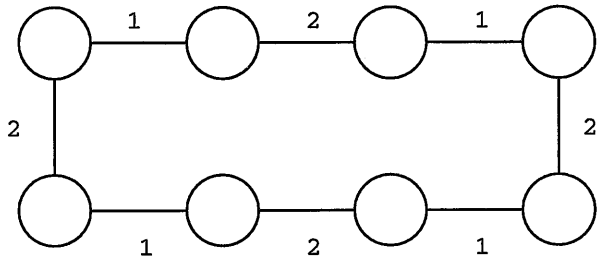

図 2.4: ring 構造の coloring 例

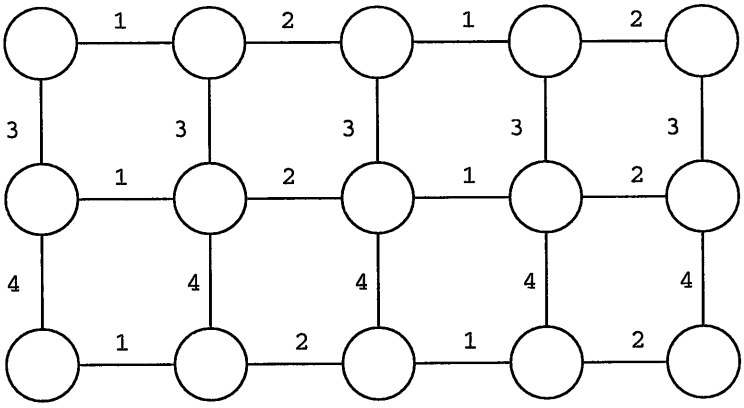

図 2.5: mesh 構造の coloring 例

2.2.3 GDE 法

generalized DE(GDE) 法は,DE 法の改良アルゴリズムとして提案された手法である [1] [2].

hypercube の場合, 同時に交換出来るプロセッサの組が一様であることから, 隣接するプロセッサと負荷を半分ずつ交換することで負荷が分散されるが, それ以外の構造に対して負荷分散を行う際には, もっと異なる比率で交換する場合が高速に負荷分散を行える事が示されており [1], その交換比率として λ を導入したのが GDE 法である.

隣接するノード i,j の持つ負荷を w_i, w_j とすると, GDE 法では式 2.3に従い負荷を交換する.

$$w_i = (1-\lambda)w_i + \lambda w_j \tag{2.3}$$

この λ を交換パラメータと呼び, $\lambda = \frac{1}{2}$ の時に DE 法と等価になる.

2.2.4 Simulated Annealing

Simulated Annealing は負荷分散に限らず, NP 完全問題のような最適解を求めるのが困難である問題の解をヒューリスティックに求めるためのアルゴリズムである.

アルゴリズムの挙動は, 実際の物理現象のアニーリングに良く似ている. 最初に, 温度 T というパラメータが決定される.T にはある程度大きな数値が代入される. 次に, プロセッサ上の負荷をランダムに少し変化させて, それが負荷分散に対して良い結果を与えたかどうかをコストの変化量 δH により評価する. コストは,Simulated Annealing で取り扱っている問題が, 求める結果に近づく程減少し, 遠ざかる程増加するように定義する.

もしも $\delta H < 0$ であるなら, 状態が改善されたので, その負荷の移動を受諾 (accept) する. δH が増加した時にも, $e^{\frac{-\delta H}{T}}$ の確率でその状態を受諾する. これは, 解が極小解に陥らないようにするための配慮である. そうして, 解を受諾するか拒否するか決めたら, 温度 T を一定の値だけ下げる. そうして, ある値まで温度 T が下がった時の解を, 最終的な解として受諾する.

2.2.5 並列処理ネットワークに適したアルゴリズム

2.2で示された 3 種のアルゴリズムについて, 並列処理プロセッサの動的負荷分散という立場からの適合性を考えると, 2.2.4の Simulated Annealing はどちらかと言うと解を求

めるのが困難である問題に対して強いアルゴリズムであり, 並列処理プロセッサに特別に向いているとは言い難い.

2.2.2のDE法については, hypercube以外の構造ではGDE法の方がより高速に負荷を分散する事が分かっているため除外する.

2.2.1のdiffusion法と,2.2.3のGDE法は, どちらも並列処理ネットワークの負荷分散に適していると言えるが, diffusion法の場合, 構成要素のプロセッサが同時通信が可能ではないと仮定した場合, 本質的にはGDE法と等価になると考えられる. 式2.1は,

$$w_i^{t+1} = \sum_{j \in A(i)} ((1-\alpha_{ij})w_i^t + \alpha_{ij}w_j^t) \tag{2.4}$$

と書く事が出来, 各colorに応じて交換パラメータλを変えたGDE法でシミュレート出来る. そこで, 本論文ではGDE法を研究対象として取り挙げ, 負荷分散の特徴について議論する.

GDE法にもcoloringの問題があるが, 並列処理プロセッサは基本的に規則的な配列をされるため, 現在問題にしている用途に限って言えば問題にはならないものと思われる.

2.3 GDE 法の収束速度の評価

2.3.1 収束速度と交換パラメータ λ の関係

反復改良アルゴリズムの評価に当たっては, アルゴリズムの収束速度の評価が問題になってくる. GDE 法を採用するにあたって, , 交換パラメータ λ と収束速度の関係について考える [1].

いま, ある並列処理ネットワーク P の持つ各プロセッサ $p_i(1 \leq i \leq n)$ が持っている負荷を $w_i(1 \leq i \leq n)$ とする. ここで, edge coloring によって番号 1 がつけられたエッジ同士で通信を行い負荷を交換した後の負荷を $w'_i(1 \leq i \leq n)$ とする. この時, $(w_1, w_2, w_3, ..., w_n)^T$ と $(w'_1, w'_2, w'_3, ..., w'_n)^T$ との間の関係は,

$$(w'_1, w'_2, w'_3, ..., w'_n)^T = M_1(\lambda)(w_1, w_2, w_3, ..., w_n)^T \tag{2.5}$$

と書き表す事が出来る. ここで $M_1(\lambda)$ は, 番号 1 の coloring が行われたエッジ同士で結ばれたプロセッサ間での負荷の移動の関係を表す $n \times n$ の正方行列であり, 対称行列になる.

同様に, 番号 i の coloring が行われたエッジ同士で結ばれたプロセッサ間での負荷の移動の関係を表す行列を $M_i(\lambda)$ とすると,c 色で coloring されたネットワークにおいて, 全体で 1 回負荷分散を行う関係を表す行列 $M(\lambda)$ は,

$$M(\lambda) = M_c(\lambda) \times ... \times M_2(\lambda) \times M_1(\lambda) \tag{2.6}$$

となる. ここで $M(\lambda)$ は GDE 行列と呼ばれ, $n \times n$ の正方行列であり, 並列処理ネットワークの構造と edge coloring によって定まる.

よって, 並列処理ネットワーク P の持つ各プロセッサ $p_i(1 \leq i \leq n)$ が初期配置として持つタスクを w_i^0 とし,t 回 GDE 法を適用した後にプロセッサ p_i が持つ負荷を w_i^t とすると,w_i を要素とするベクトル $W^t = (w_1^t, w_2^t, ..., w_n^t)^T$ と,$t+1$ 回 GDE 法を適用した際の負荷を要素とするベクトル W^{t+1} の間には, 式 2.6 で述べた $M(\lambda)$ を用いると,

$$W^{t+1} = M(\lambda)W^t \tag{2.7}$$

という関係が成立する.

よって, 初期配置として与えられた負荷のベクトルを $W^0 = (w_1^0, w_2^0, ..., w_n^0)^T$ とすると,W^0 と W^t には,

$$W^t = M(\lambda)^t W^0 \tag{2.8}$$

の関係が成立する.

GDE 法の収束速度は,t を大きくしていった時に,$M(\lambda)^t$ がどれだけ小さい t で一定の値に収束するかで決まる.GDE 行列 $M(\lambda)$ は,$M(\lambda)$ の固有値と固有ベクトルを用いて,

$$M(\lambda)K = K\Gamma \tag{2.9}$$

と表せる. ここで K は GDE 行列 $M(\lambda)$ の固有ベクトルから成る $n \times n$ の正方行列,Γ は $M(\lambda)$ の固有値 $\gamma_1, \gamma_2, ..., \gamma_n$ を対角成分とする対角行列である.

したがって,$M(\lambda)^t$ は K と Γ を用いて,

$$M(\lambda) = K\Gamma K^{-1} \tag{2.10}$$

$$M(\lambda)^t = K\Gamma^t K^{-1} \tag{2.11}$$

と表せる. よって $M(\lambda)^t$ は,$M(\lambda)$ の固有値 $\gamma_1, \gamma_2, ..., \gamma_n$ を用いて,

$$M(\lambda)^t = \gamma_1^t C_1 + \gamma_2^t C_2 + ... + \gamma_n^t C_n \tag{2.12}$$

と書く事が出来る. つまり,$M(\lambda)^t$ の収束速度は, 式 2.12 の各固有値のべき乗がどれだけ速く収束するかによって決まる.

ここで,GDE 行列が確率行列であることから, 第一固有値 γ_1 は常に 1 になる [1] という特徴があり, $\gamma_1 > \gamma_2 > ... > \gamma_n$ なので,GDE 法の収束速度は,$M(\lambda)$ の第二固有値 γ_2 の大きさに最も影響を受ける, ということになる.

したがって, ある並列処理プロセッサに GDE を適用する場合最も最適な交換パラメータ λ は, その並列処理プロセッサの構造と edge coloring から得られる GDE 行列 $M(\lambda)$ の第二固有値を最小にする λ を選ぶことで得られる.

簡単のため, 固有値は実数としたが, 一般的には複素数になる. その場合は, その絶対値 ($\gamma_i = \alpha_i + i\beta_i$ のとき, $\sqrt{\alpha_i^2 + \beta_i^2}$) を以上の議論の γ_i と読み換えれば同じ議論が成立する.

2.3.2 edge coloring と収束速度の関係

ある並列ネットワークを想定した時, そのネットワークの coloring は基本的に複数のパターンが存在する. 図 2.6, 図 2.7は, 同じネットワークに異なる edge coloring をほどこした場合の例である.

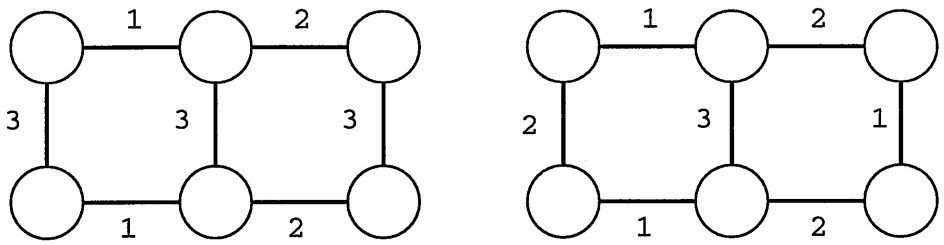

図 2.6: 異なる coloring の例:coloring のパターンが違う場合

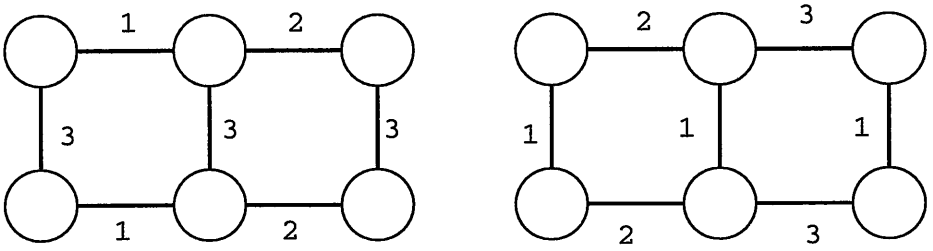

図 2.7: 異なる coloring の例:coloring のパターンが同じで番号が違う場合

図 2.6の例では,2 つの coloring のパターンが完全に異なるのに対して, 図 2.7の例は基本的な coloring のパターンは同じで, 割り当てられている番号が異なる.

ここで,3 色で edge coloring されているネットワークにおいて, 図 2.7の例のように基本的な coloring は同じだが割り当てられている番号が違うようなネットワークに対して, 以下の定理が成り立つ.

定理: 3 色で edge coloring されたあるネットワークの GDE 行列 M(λ) の固有値は, その edge coloring のパターンが基本的に同じで, 割り当てられている番号が違うもの同士では同じになる.

証明) 3 色で coloring されたあるネットワークの GDE 行列を $M(\lambda) = M_3(\lambda)M_2(\lambda)M_1(\lambda)$ とした時, その色違いのパターンで coloring されたネットワークの GDE 行列は, 上式の右辺の各行列を互いに入れ替えることで得られる. この時, その組み合わせは, $M_1(\lambda)M_2(\lambda)M_3(\lambda)$, $M_1(\lambda)M_3(\lambda)M_2(\lambda)$, $M_2(\lambda)M_1(\lambda)M_3(\lambda)$, $M_2(\lambda)M_3(\lambda)M_1(\lambda)$, $M_3(\lambda)M_1(\lambda)M_2(\lambda)$, $M_3(\lambda)M_2(\lambda)M_1(\lambda)$ の 6 通りが考えられる. この 6 通りの GDE 行列の全てで固有値が等しい事を以下の命題を用いて証明する.

命題 1: 対称行列 A, B, C において, C が正則であるとき, 積 ABC の固有値と積 CAB の固有値は等しい.

証明) 行列積 ABC の固有値を γ, 固有ベクトルを$\boldsymbol{x}$とすると,

$$(ABC - \gamma I)\boldsymbol{x} = 0 \tag{2.13}$$

が成り立つ. ここで,C は正則なので逆行列 C^{-1} が存在し,

$$ABC - \gamma I = ABC - \gamma C^{-1}C = C^{-1}(CBA - \gamma I)C \tag{2.14}$$

となる. 式 2.14の右辺を式 2.13に代入すると,

$$(ABC - \gamma I)\boldsymbol{x} = C^{-1}(CAB - \gamma I)C\boldsymbol{x} = 0 \tag{2.15}$$

と表せるため, ここで $C\boldsymbol{x} = \boldsymbol{y}$ と置くと式 2.15は,

$$(CAB - \gamma I)\boldsymbol{y} = 0 \tag{2.16}$$

となる. よって, 積 ABC の固有値 γ は同時に積 CAB の固有値である. (証明終)

命題 1 より, $M_1(\lambda)M_2(\lambda)M_3(\lambda)$, $M_2(\lambda)M_3(\lambda)M_1(\lambda)$, $M_3(\lambda)M_1(\lambda)M_2(\lambda)$ の固有値は等しい事が分かる. 同様に, $M_3(\lambda)M_2(\lambda)M_1(\lambda)$, $M_2(\lambda)M_1(\lambda)M_3(\lambda)$, $M_1(\lambda)M_3(\lambda)M_2(\lambda)$ の固有値もお互い等しい.

次に, この 2 組の固有値が等しい事を以下の命題で証明する.

命題 2: 対称行列 A, B, C において, 積 ABC の固有値と積 CBA の固有値は等しい.

証明) ある行列 M とその転置行列 M^T の固有値は等しいので, 積 ABC と $(ABC)^T$ の固有値はそれぞれ等しい. ここで, A, B, C は対称行列なので, $(ABC)^T = C^TB^TA^T = CBA$ となる. よって, 積 ABC と積 CBA の固有値はそれぞれ等しい. (証明終)

命題 2 より,$M_1(\lambda)M_2(\lambda)M_3(\lambda)$ と $M_3(\lambda)M_2(\lambda)M_1(\lambda)$ の固有値はお互い等しい. よって, 前出の 6 通りの GDE 行列において, その固有値は全て等しくなる.(証明終)

定理より,GDE 法において 3 色で edge coloring が可能なネットワークを考える場合, 割り当てられている番号が違うだけの coloring のパターン同士ではその GDE 行列の固有値が同じになるため. アルゴリズムの収束速度は等しくなると考えられる.

2.4 まとめ

本章では, 反復改良アルゴリズムの中から,

- diffusion 法
- DE 法
- GDE 法
- Simulated Annealing

について述べ, 並列処理プロセッサの負荷分散という観点から評価を行い, 構成要素プロセッサの全てのリンクで同時通信が出来ないネットワークを考えて,GDE 法を研究対象として採用した.

また,GDE 法の収束速度の評価法として, ネットワークの構造と coloring から与えられる GDE 行列 $M(\lambda)$ の固有値を用いた方法を述べた.

第3章

tree構造プロセッサの動的負荷分散

3.1 はじめに

tree構造は古くから良く知られ, 負荷分散のみならず並列処理の分野で広く研究に使われてきた構造である. 動的負荷分散の分野でも, Tree Walking Algorithm(TWA) のような,tree構造の性質を考慮した負荷分散アルゴリズムが提唱されている. [7] しかし, 反復改良アルゴリズムをtree構造に対して適用した研究報告は非常に少ない.

本章では,tree構造の一つである完全二分木構造を例に取り, 反復改良アルゴリズムのGDE法による負荷分散のシミュレーションを行い, アルゴリズムの収束速度と交換パラメータ λ の関係について評価, 検討を行う.

また, 完全二分木構造特有と思われる特徴について調べ, それに対する考察を行う.

3.2 tree 構造における GDE 法の適用

2.2で議論したように, 負荷分散の対象となる回路が規則的なアーキテクチャによって形成されている場合には GDE 法が容易に適用出来る. 一般に, 完全 n 分木に対して edge coloring を適用するには, 最低で $n+1$ 色を必要とする. 従って, 本章で評価の対象とする完全二分木は 3 色で edge coloring が可能であり, その例は図 3.1 のようになる.

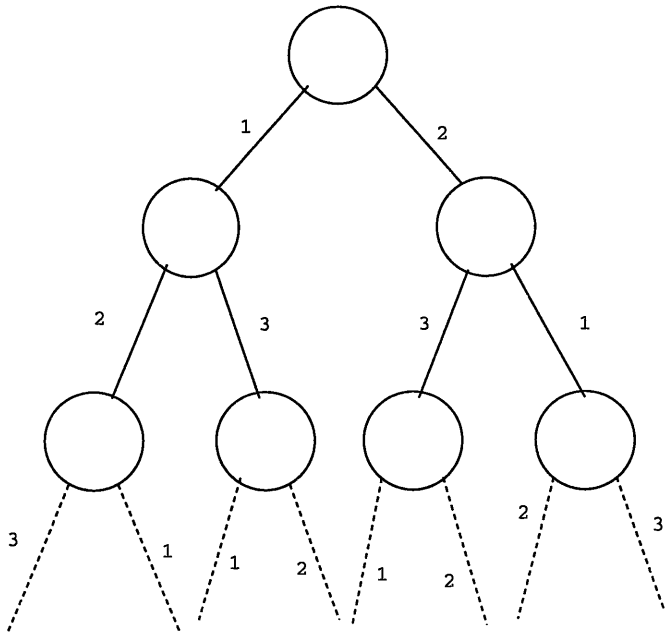

図 3.1: 完全二分木の edge coloring の例

完全二分木の場合,edge coloring のパターンは基本的にこの例の番号を入れ替えたものか, 子の木を回転させたパターンのみが存在する.

本章では, 二分木の edge coloring は全て図 3.1 と同じものに統一するものとする.

3.3 シミュレーション実験

最初に, 高さが 3, 4, 5, 6 である完全二分木 (プロセッサ数はそれぞれ 7, 15, 31, 63) の完全二分木構造ネットワークにおいて, その構造と edge coloring から, おのおのの GDE 行列 $M(\lambda)$ を求め,S 言語 [12] により, 各行列の第二固有値を最小にする交換パラメータ λ の値を算出した.

さらに, 下記の条件で, 完全二分木構造プロセッサの負荷分散のシミュレーションを,GDE 法のアルゴリズムを用いて行い, 最適な λ の値を求めた.

3.3.1 シミュレーション条件

- 各プロセッサには初期配置として, プロセッサ全体の平均が 10, 100, 1000 となるようなランダムな負荷を配置. ただし, 負荷の大きさの上限は, 平均の 2 倍とする.
- 初期配置のサンプル数は, おのおの 200 通りとする.
- 取り扱う負荷は整数とする. 計算上, 移動するタスクの数が小数になった場合, 式 3.1, 式 3.2に従いタスクの切り上げ, 切り捨てを行う.

$$w_i = \lceil (1-\lambda)w_i + \lambda w_j \rceil (w_i \geq w_j) \tag{3.1}$$

$$w_i = \lfloor (1-\lambda)w_i + \lambda w_j \rfloor (w_i < w_j) \tag{3.2}$$

- λ は,0.5 から 0.99 の範囲で 0.01 ずつ変化させる.
- アルゴリズムの停止条件は, 隣接するプロセッサ間の全てで, タスク数の差が 1 以下になった時とする.
- アルゴリズムの収束速度は, アルゴリズムの停止までに sweep が何回実行されたか (=何回, プロセッサ全体に対してアルゴリズムが実行されたか) で評価する.

3.3.2 実験結果と考察

S 言語により求めた, 各構造の第二固有値と交換パラメータ λ の関係を図 3.2に示す. 縦軸は第二固有値の大きさ, 横軸は交換パラメータ λ の値である.

図 3.2を見ると, 高さ 3 の完全二分木における第二固有値の最小値が最も小さく, 高さが増す毎に第二固有値の最小値が上昇してくるのが見える.2.3.1でも述べた通り, 第二固有値は GDE 法の収束速度に最も影響を与える要素である.

従って, 第二固有値の最小値が増大するということは, 収束に要する時間が, ネットワークの規模が増すにつれて長くなるという事を意味する.

また, 第二固有値を最小とする交換パラメータ λ の値が, 完全二分木の高さが増すごとに 1 に近くなっていくという特徴が見られる. この現象は, 最適な λ がノード数に依存するという,C. Z .Xu らが mesh,ring,torus などの構造に対して行った研究と一致する [1].

シミュレーションの結果を図 3.3, 図 3.4, 図 3.5, 図 3.6, 図 3.7, 図 3.8, 図 3.9に示す.

図 3.3, 図 3.4, 図 3.5, 図 3.6 はそれぞれ, 完全二分木の高さが 3,4,5,6 の時の負荷分散の結果である. 縦軸が, アルゴリズムが停止するまでに行われた sweep の回数, 横軸は使用した λ の値である.

また, 図 3.3, 図 3.4, 図 3.5, 図 3.6の結果を, プロセッサに割り与えられた平均のタスク数を基準に書き直したもの (縦軸スケールをデータに合わせている) が図 3.7, 図 3.8, 図 3.9 である.

どのプロセッサ数の場合も, 平均タスク数が 10 の時は数回の sweep 数で負荷分散が完了しているが, 負荷分散された結果そのものはばらつきが大きく, 完全と呼べるものではなかった. これは, タスク数は整数のみとして扱い, 隣接するプロセッサとのタスクの差が全て 1 以下になった時点でアルゴリズムを停止するという条件のためである.

この条件により, アルゴリズムは最悪の場合, 並列処理プロセッサ内の直径分だけの負荷のずれを残して終了する. よって, プロセッサが持つ負荷の平均タスク数が, 並列処理プロセッサの直径に比べて充分大きい時でないと, 本アルゴリズムは完全に近い負荷分散を行えないことが分かる. 完全二分木構造の場合, 最も小規模なプロセッサ数 7 の場合でも, 最も離れている 2 プロセッサ間の距離は 4 となるため, 平均タスク数が 10 程度の負荷を分散するには本アルゴリズムは適さないと言える.

実験に用いた完全二分木において, シミュレーションが最速に収束した λ の値とその時の sweep 数,, 第二固有値の評価から求めた最適な λ の値およびその時の第二固有値を

表にしたものが表3.1である.

これらの結果から, 最速に収束する λ は, 二分木の高さが増大するにつれて全体的に上昇することが分かる. この結果は, 図3.2において, 二分木の高さが増すごとにGDE行列 $M(\lambda)$ の第二固有値が増大しているという解析結果と良く一致する.

また, 平均タスク数が増えるにつれて収束に時間がかかることが分かる.

	高さ3	高さ4	高さ5	高さ6
γ_2を最小にするλ	0.63	0.69	0.73	0.76
第二固有値γ_2	0.4384	0.7332	0.8732	0.9386
λ:task 10	0.67 ～ 0.69	0.67 ～ 0.68	0.75 ～ 0.76	0.77
sweep:task 10	(3.525)	(4.710)	(5.830)	(6.455)
λ:task 100	0.69	0.78	0.80	0.82
sweep:task 100	(6.33)	(11.67)	(19.805)	(31.105)
λ:task 1000	0.66	0.73	0.77	0.80
sweep:task 1000	(8.805)	(18.785)	(36.135)	(63.620)

表 3.1: 完全二分木のプロセッサ数と λ の関係

シミュレーションの結果と第二固有値を用いた分析を比較すると, 両方の結果はある程度近い交換パラメータ λ を最適としているものの, 完全に一致しているとは言い難い. これは, アルゴリズムの停止条件を, 全ての近接ノードとの負荷の差が1以下としたため, 完全な負荷分散がなされていない状態においてもアルゴリズムが停止してしまう事が影響していると思われる.

また, シミュレーションに用いた条件や完全二分木の大きさが同じ場合でも, アルゴリズムの収束速度に大きな違いが見られた.

図3.10は, 高さ5の完全二分木の, 交換パラメータ λ が最適値0.80の時の200通りの初期配置に対するシミュレーション結果について,sweep数の出現頻度を図にしたものである. 平均のsweep数は表3.1の通り19.805だが, 最小値は10, 最大値は28と, 3倍近くの差が見られる. これは, タスクの初期配置の異なりが, アルゴリズムの収束速度に対して影響を及ぼしたためと考えられる.

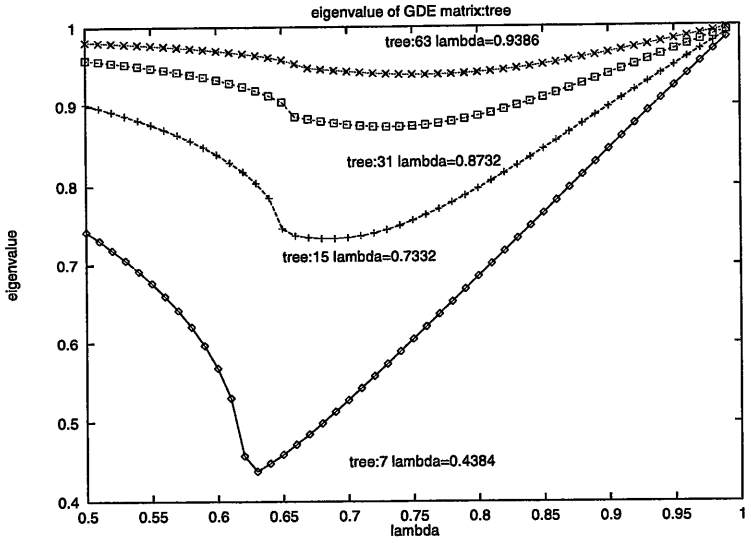

図 3.2: GDE 行列の第二固有値と λ の関係:完全二分木の場合

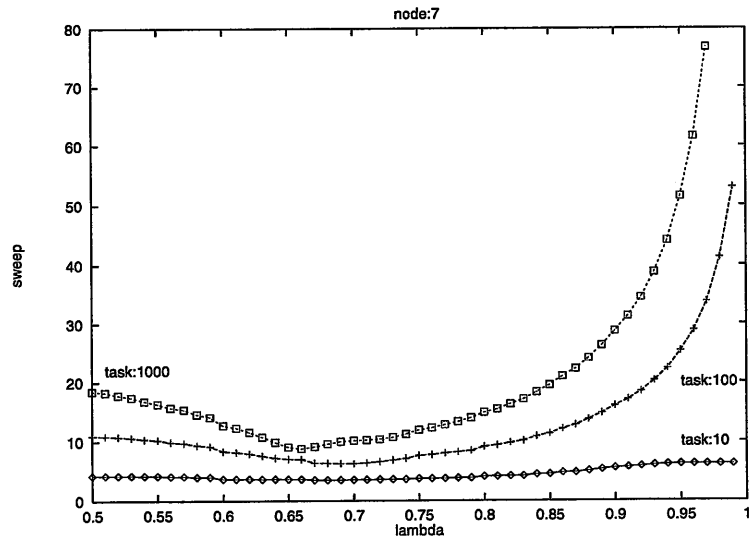

図 3.3: 完全二分木の負荷分散の例:高さ 3 の場合

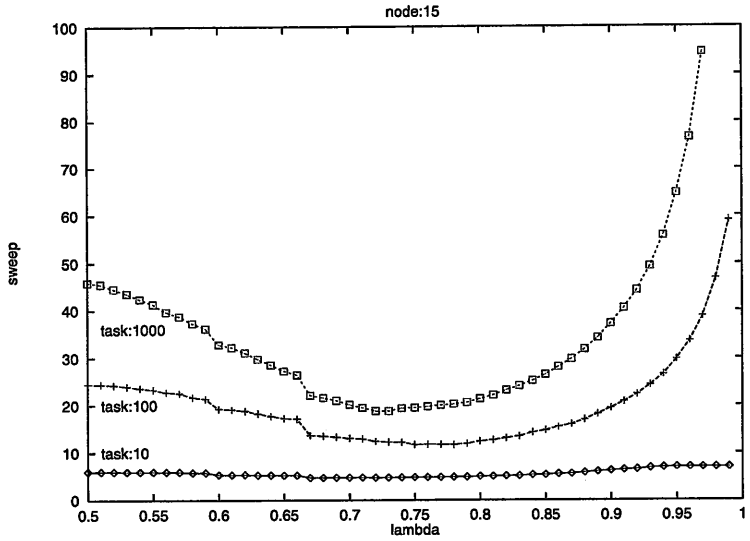

図 3.4: 完全二分木の負荷分散の例:高さ 4 の場合

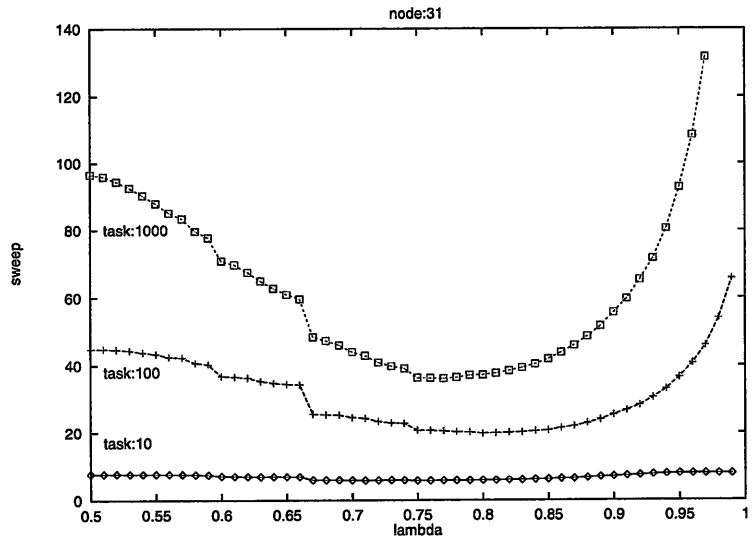

図 3.5: 完全二分木の負荷分散の例:高さ 5 の場合

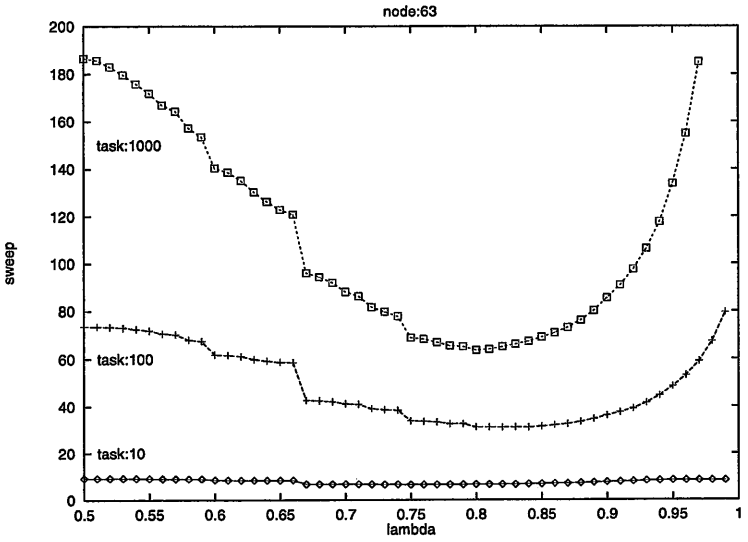

図 3.6: 完全二分木の負荷分散の例:高さ 6 の場合

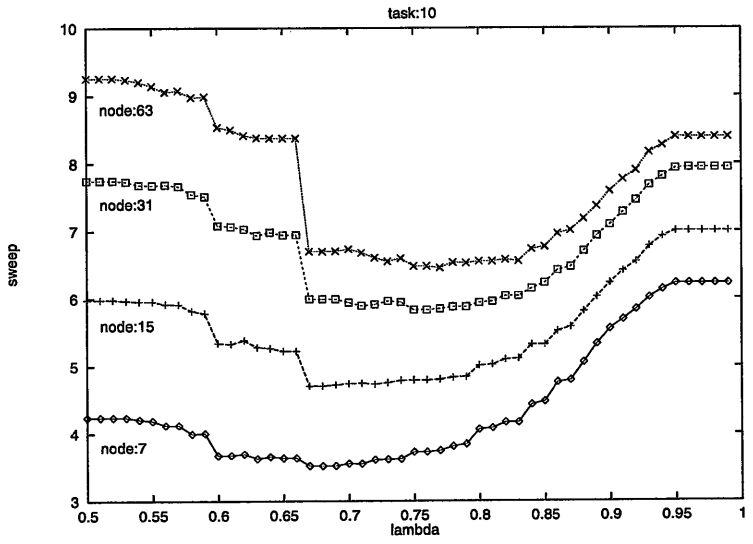

図 3.7: 完全二分木の負荷分散の例:平均タスク数が 10 の場合

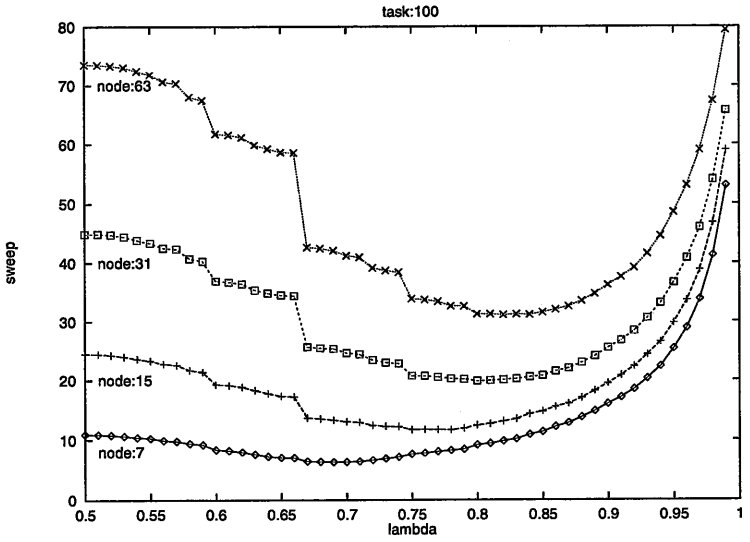

図 3.8: 完全二分木の負荷分散の例:平均タスク数が 100 の場合

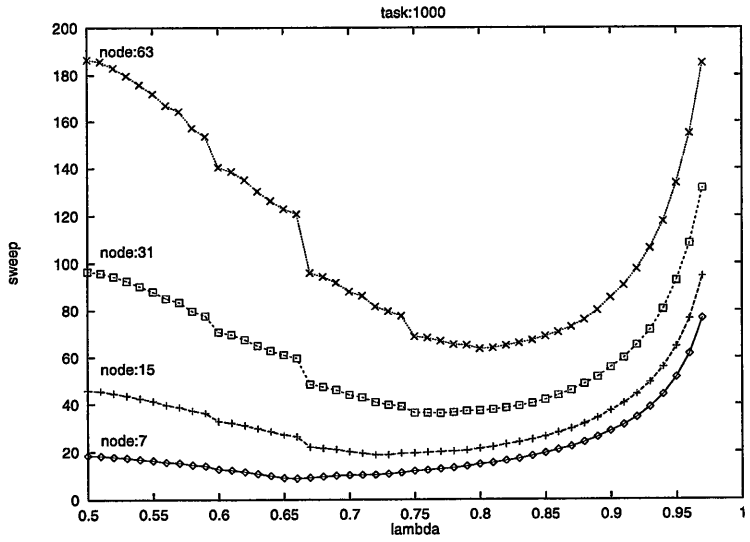

図 3.9: 完全二分木の負荷分散の例:平均タスク数が 1000 の場合

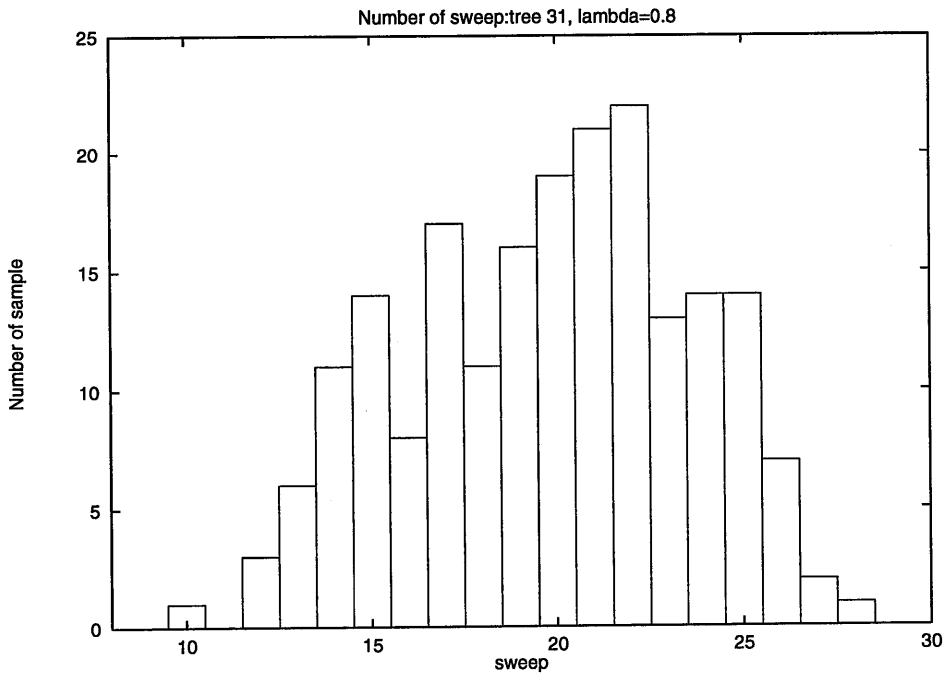

図 3.10: 負荷分散のばらつきの頻度

3.4 収束速度に関する問題

3.3.2で述べた通り,3.3の実験において, 実験条件が全く同じ状態においても, 異なる初期配置における収束速度の結果が著しく異なるという現象が見られた. この事から, 完全二分木の場合, 負荷の初期配置はGDE法アルゴリズムの収束速度に大きな影響を与えることが予想される.

その可能性を検証すべく, シミュレーションによる実験を行った.

3.4.1 シミュレーション条件

- 高さ4の完全二分木構造を対象.
- 平均タスク数は100とする.
- 初期配置はランダムに配置する. その上で, ある特定のノードに平均タスク数の5倍である500のタスクを配置する.
- 初期配置のサンプル数は200通りを使用.

3.4.2 実験結果

図3.11に実験結果を示す. 縦軸は収束に必要としたsweepの数, 横軸は意図的に負荷を加えたプロセッサである. 番号は, 二分木のrootを1として, 高さ優先で左から順につけていった.

同じ色で彩色されている棒グラフは, 二分木の中で同じ高さに存在するプロセッサの結果である事を意味する. これを見ると, 同じ高さのプロセッサでも, 初期配置により負荷の偏りが出た時の収束速度が明らかに異なる事が分かる.

理由としては, 過大な負荷を分散させる際に,rootの部分の通信性が悪いため, 一番最初にrootに効率的に負荷を送るようなcoloringがなされているプロセッサに負荷が多く存在する場合, 素早く全体のプロセッサに負荷の分配がなされるためと考えられる.

以上の実験結果から, 完全二分木構造のプロセッサをGDE法で負荷分散する際には, 初期配置の負荷の偏りを考慮して, 過負荷になっているプロセッサがただちにrootに負荷

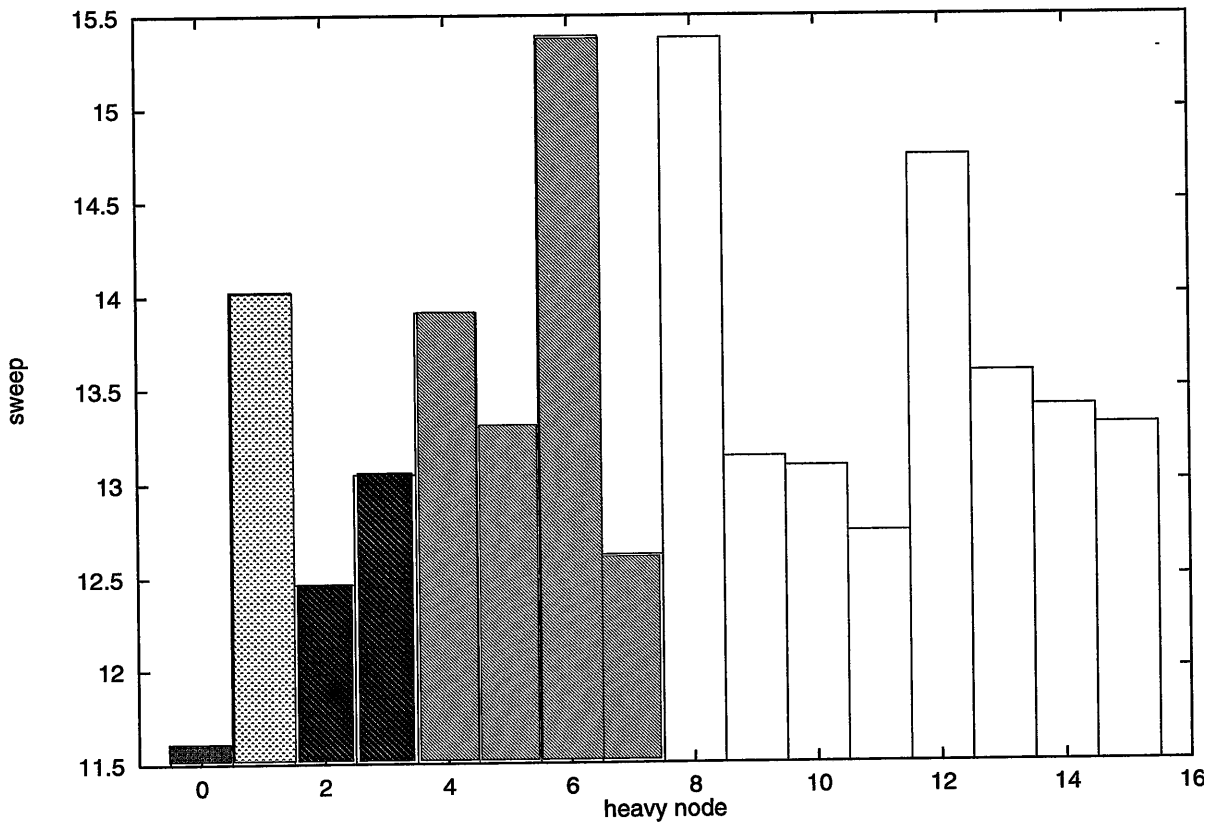

図 3.11: 局所的に負荷を加えた場合の収束速度の変化

を送る事が出来るようにcoloringの番号を決定する事で,アルゴリズムがより高速に収束すると思われる.

3.5 まとめ

本章では,tree構造の一つである完全二分木構造プロセッサの負荷分散において,反復改良アルゴリズムであるGDE法を適用した際の有効性について調べた.

GDE行列の固有値による評価とシミュレーションによる解析から,高さ3,4,5,6の完全二分木構造においてアルゴリズムが最速に収束する交換パラメータλを求めた.

負荷分散の完全性という点では,プロセッサの持つ負荷の平均タスク数が少ない場合守られなくなるが,負荷の平均タスク数が並列処理プロセッサの直径に比べて多い場合は完全性が増す.

また,同じ条件でも初期配置の違いによって負荷分散の速度が大きく異なる場合がある事が確認されたが,これは初期配置と二分木のedge coloringに関係がある事が示された.

第4章

CCC構造プロセッサの動的負荷分散

4.1 はじめに

CCC(Cube-Connected Cycle)構造とは,非常に良く知られている構造であるhypercube構造の,各プロセッサをring構造で置換して得られる構造である.CCCもtreeと同様,反復改良アルゴリズムを適用した負荷分散に関する研究報告は非常に少ない.

本章では,hypercube構造のうち,3次元hypercubeの置換により得られるCCC構造プロセッサに対して,反復改良アルゴリズムのGDE法による負荷分散のシミュレーションを行い,最適な交換パラメータλを求める.

また,CCC構造の場合における特徴について議論し,それに対する評価と考察を行う.

4.2 CCC 構造における GDE 法の適用

本章では,CCC のうち 3 次元 hypercube のノードを ring で置換した構造について取り扱う. 3 次元 hypercube を元にした CCC は, 基本的に 3 色での edge coloring が可能である.

3 次元 hypercube のノードを ring に置換する際, プロセッサの配置の仕方によっては同じノード数でも何種かの構造が存在する. そこで, 本論文では, 回路全体の構造をなるべく対称的にするため, hypercube の次元数である 3 の倍数の 3, 6, 9 のノードを持つ ring に置換して得られる CCC を考える.

図 4.1に,hypercube のノードを, ノード数 3 と 6 の ring に置換した例を示す.

第 3 章 における完全二分木構造の場合と異なり, CCC 構造の場合, 一つの構造に対して複数の edge coloring が存在する場合がある.

本論文で扱う, 3 次元 hypercube のノードをノード数 3, 6, 9 の ring で置換する CCC では, ノード数 3 の ring で置換したものについては基本的に 1 通りの edge coloring のみを持つが, ノード数 6, 9 のものについては, 二通り以上の edge coloring が存在する.

図 4.2は, ノード数 6 の ring により作った 3 次元 hypercube の edge coloring の例である. 図 4.2 の左側の coloring は, 最初に hypercube 構造の部分を coloring してしまい, その結果に合わせて ring の部分を coloring した例であり, 右側は最初に ring の部分を coloring してしまい, それに合わせて hypercube 構造の部分を coloring したものである.

ノード数 9 の ring を持つ CCC においても, 同様の方法による 2 通りの coloring が存在する. 以下の議論ではノード数 6, 9 の ring による CCC においては, これら 2 通りの coloring を別々に扱い実験を行うものとする.

4.3 シミュレーション実験

最初に,ring 部分のノード数が 3(プロセッサ数 24), 6(プロセッサ数 48), 9(プロセッサ数 72) の CCC 構造ネットワークにおいて, その構造と edge coloring から, おのおのの GDE 行列 $M(\lambda)$ を求め, S 言語 [12] により, 各行列の第二固有値を最小にする交換パラメータ λ の値を算出する.

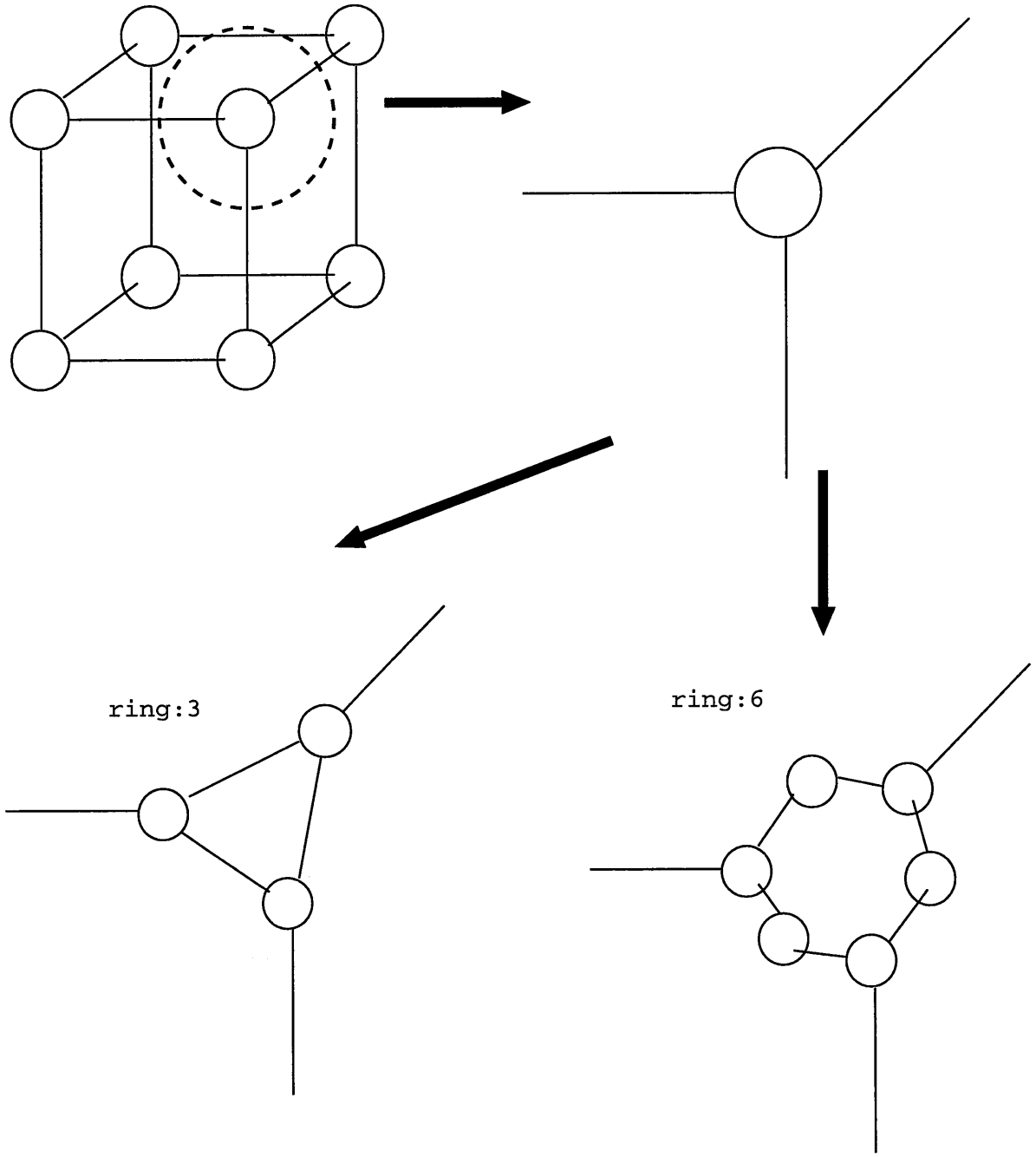

図 4.1: hypercube 構造の置換の例

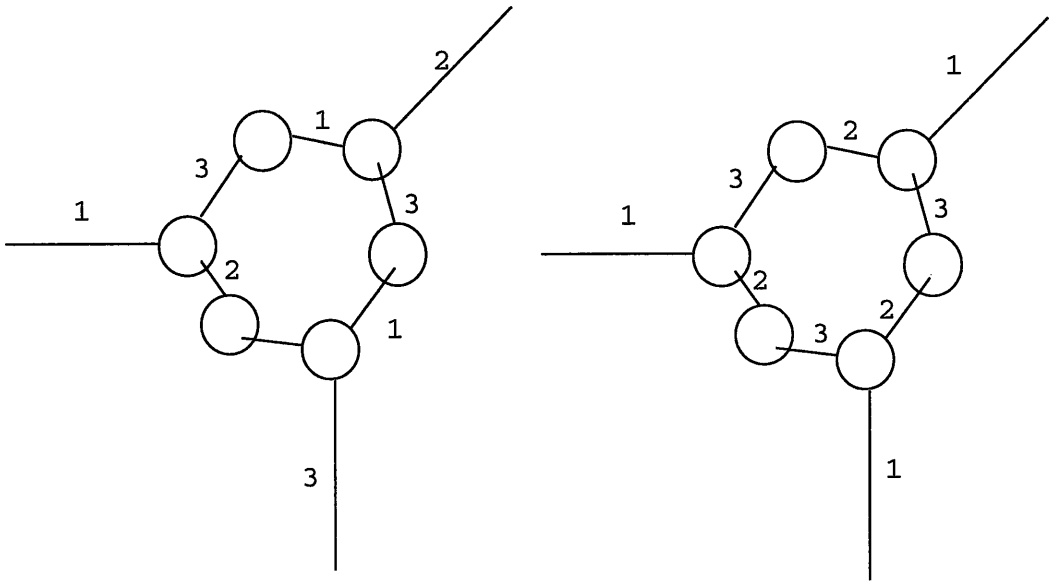

図 4.2: CCC 構造の edge coloring の例

ring 部分のノード数が 6,9 のものについては 2 通りの edge coloring を取り扱い, 最初に ring 構造の部分を coloring してしまい, その結果に合わせて hypercube の部分を coloring したものを 6(1), 9(1), 最初に hypercube の部分を coloring してしまい, それに合わせて ring 構造の部分を coloring したものを 6(2), 9(2) と呼び区別する.

さらに, 下記の条件で,ring 部分のプロセッサ数が 3, 6(1), 6(2), 9(1), 9(2)CCC 構造ネットワークの負荷分散のシミュレーションを, GDE 法のアルゴリズムを用いて行った.

4.3.1 シミュレーション条件

- 各プロセッサには初期配置として, プロセッサ全体の平均が 10, 100, 1000 となるようなランダムな負荷を配置. ただし, 負荷の大きさの上限は, 平均の 2 倍とする.
- 初期配置のサンプル数は, おのおの 200 通りとする.
- 取り扱う負荷は整数とする. 計算上, 移動するタスクの数が小数になった場合, 式 4.1, 式 4.2に従いタスクの切り上げ, 切り捨てを行う.

$$w_i = \lceil (1-\lambda)w_i + \lambda w_j \rceil (w_i \geq w_j) \tag{4.1}$$

$$w_i = \lfloor (1-\lambda)w_i + \lambda w_j \rfloor (w_i < w_j) \tag{4.2}$$

- λ は,0.5 から 0.99 の範囲で 0.01 ずつ変化させる.
- アルゴリズムの停止条件は, 隣接するプロセッサ間の全てで, タスク数の差が 1 以下になった時とする.
- アルゴリズムの収束速度は, アルゴリズムの停止までに sweep が何回実行されたか (=何回, プロセッサ全体に対してアルゴリズムが実行されたか) で評価する.

4.3.2 実験結果と考察

S 言語により求めた, 各構造の第二固有値と交換パラメータ λ の関係を図 4.3に示す. 縦軸は第二固有値の大きさ, 横軸は交換パラメータ λ の値である.

また, シミュレーションの結果を図 4.4, 図 4.5, 図 4.6, 図 4.7, 図 4.8, 図 4.9, 図 4.10, 図 4.11に示す.

図4.4, 図4.5, 図4.6, 図4.7, 図4.8 はそれぞれ,ring 部分のプロセッサ数が 3,6(1),6(2),9(1),9(2) の CCC における負荷分散シミュレーションの結果である. 縦軸が, アルゴリズムが停止するまでに行われた sweep の回数, 横軸は使用した λ の値である.

最速に収束した λ の値を表にしたものが表 4.1である.CCC の場合, 平均タスク数が 10 の時にシミュレーションで得られた最適な交換パラメータ λ が, 平均タスク数 100,1000 の場合に比べて大きく異なるという傾向が見られた. これは,CCC 構造の直径が大きいために, 平均タスク数が少ない場合は完全な負荷分散がなされていないうちにアルゴリズムが終了してしまうためと思われる.

	ring:3	ring:6(1)	ring:6(2)	ring:9(1)	ring:9(2)
γ_2を最小にするλ	0.66	0.73	0.81	0.83	0.80
第二固有値γ_2	0.4074	0.7697	0.7214	0.7924	0.7800
λ:task 10	0.73	0.75	0.75	0.77	0.72
sweep:task 10	(3.605)	(4.99)	(4.815)	(5.27)	(5.39)
λ:task 100	0.80	0.81	0.78	0.85	0.85
sweep:task 100	(7.980)	(13.50)	(10.73)	(15.54)	(14.94)
λ:task 1000	0.80	0.77	0.78	0.85	0.84
sweep:task 1000	(11.625)	(22.895)	(17.405)	(29.95)	(25.47)

表 4.1: CCC のプロセッサ数と λ の関係

また, 図 4.4, 図 4.5, 図 4.6, 図 4.7, 図 4.8の結果を, プロセッサに割り与えられた平均のタスク数を基準に書き直したもの (縦軸スケールが変更されている) が図 4.9, 図 4.10, 図 4.11である.

完全二分木の場合と同様, 平均タスク数が 10 の時は負荷分散された結果そのものはばらつきが大きく, 完全と呼べるものではなかった. 特に, 回路の直径が大きくなる 9(1),9(2) で, 平均タスク数 10 の時の結果が著しく悪くなるという傾向が出た.

また, 今回の実験では,CCC 構造に用いられているプロセッサ数が同じ場合でも,edge coloring によってアルゴリズムの収束速度に違いが見られた. 固有値による分析でも, 例えば 6(1) の最適な分散パラメータ λ は 0.73 であるが,6(2) の最適な λ は 0.81 であり, その時の第二固有値の値も違っている. このことは, 同じ構造を持つ回路でも,edge coloring の違いによって収束速度に変化が見られるという事を示している.

実験の場合,hypercube を優先に coloring した 6(2),9(2) の方が, ring を優先的に coloring した 6(1),9(1) よりも高速に収束するという結果になった.

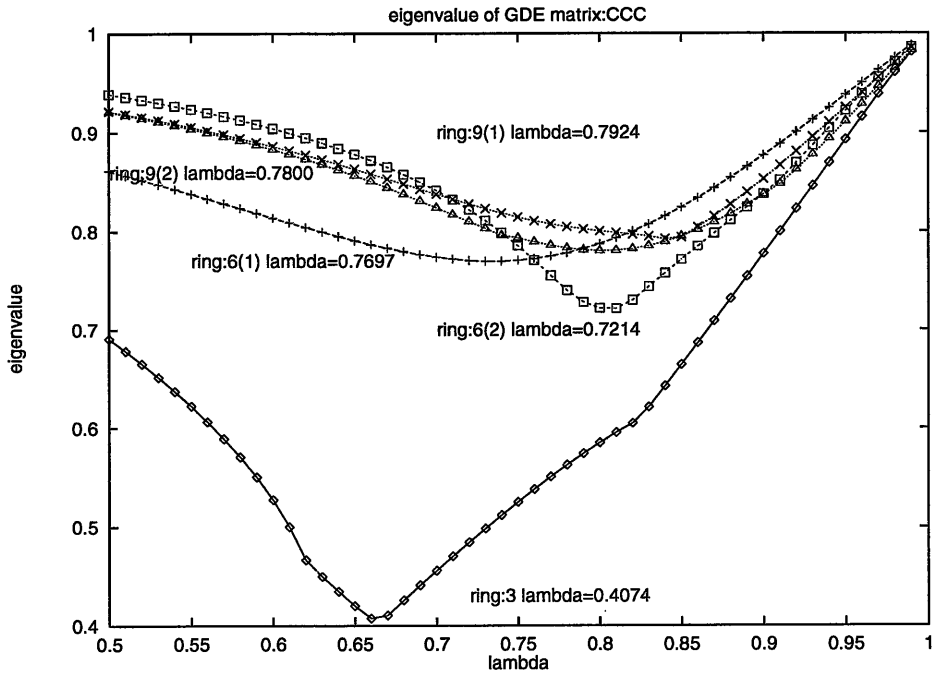

図 4.3: GDE 行列の第二固有値と λ の関係:CCC 構造の場合

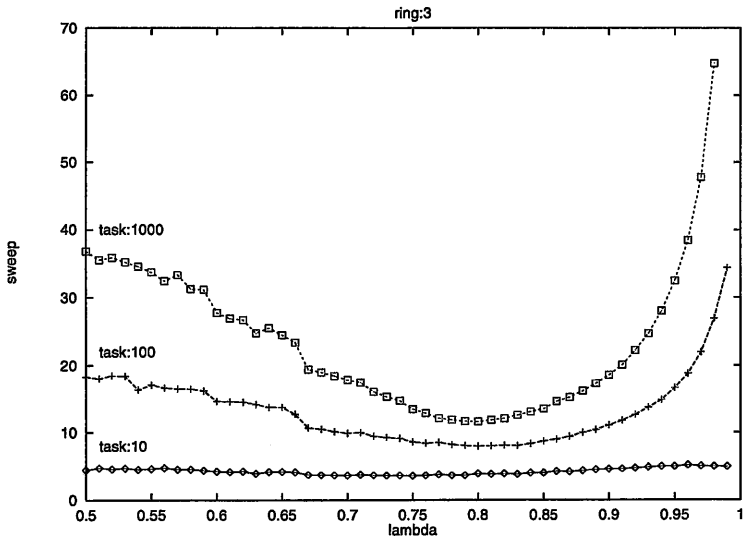

図 4.4: CCC 構造の負荷分散の例:ring のプロセッサ数 3 の場合

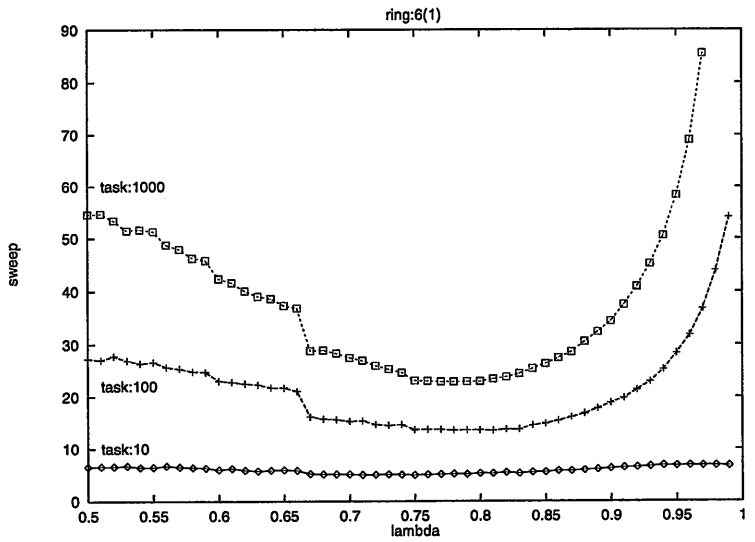

図 4.5: CCC 構造の負荷分散の例:ring のプロセッサ数 6(1) の場合

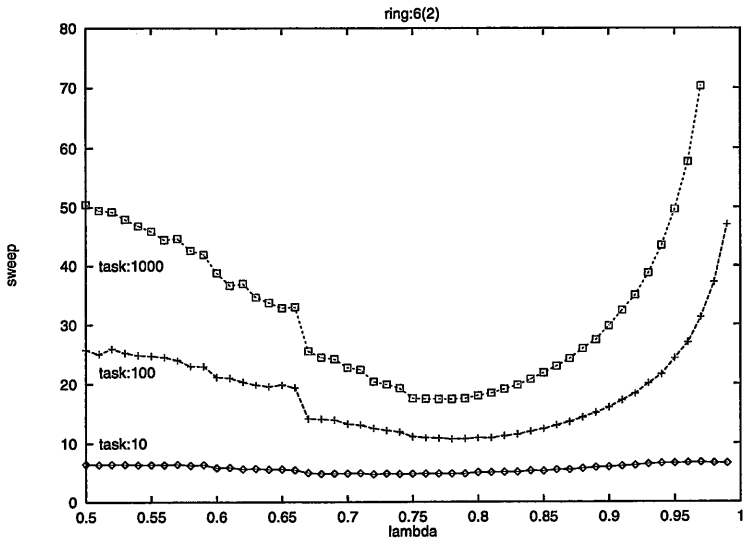

図 4.6: CCC 構造の負荷分散の例:ring のプロセッサ数 6(2) の場合

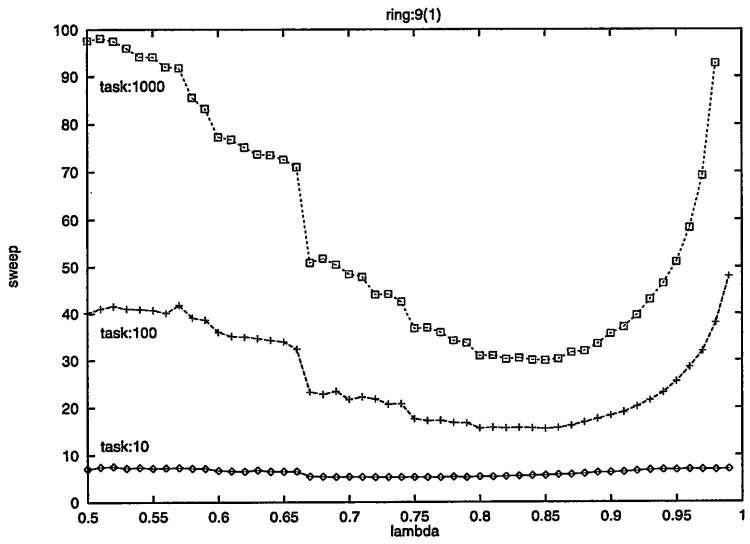

図 4.7: CCC 構造の負荷分散の例:ring のプロセッサ数 9(1) の場合

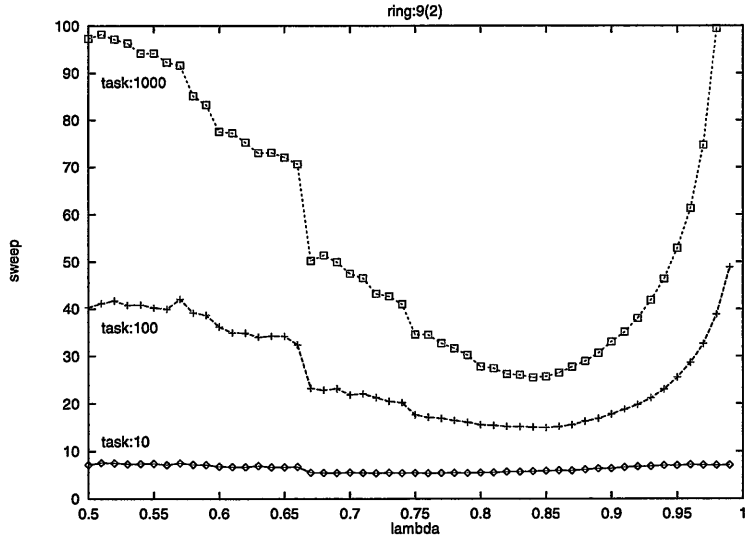

図 4.8: CCC 構造の負荷分散の例:ring のプロセッサ数 9(2) の場合

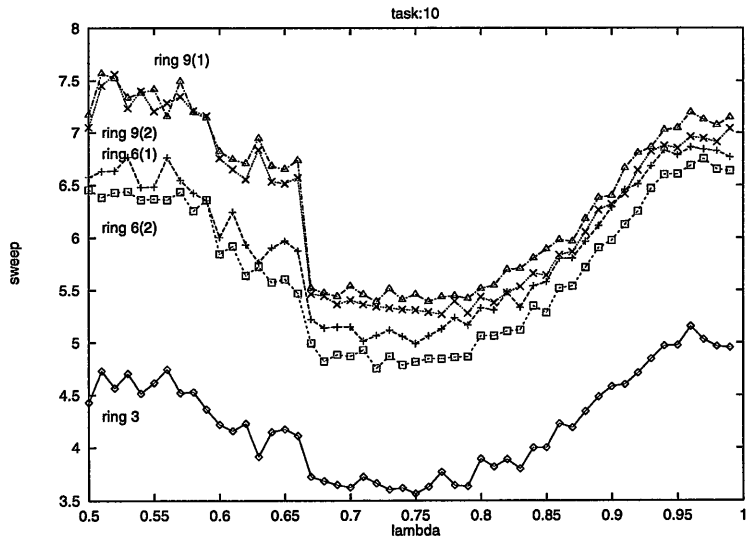

図 4.9: CCC 構造の負荷分散の例:平均タスク数が 10 の場合

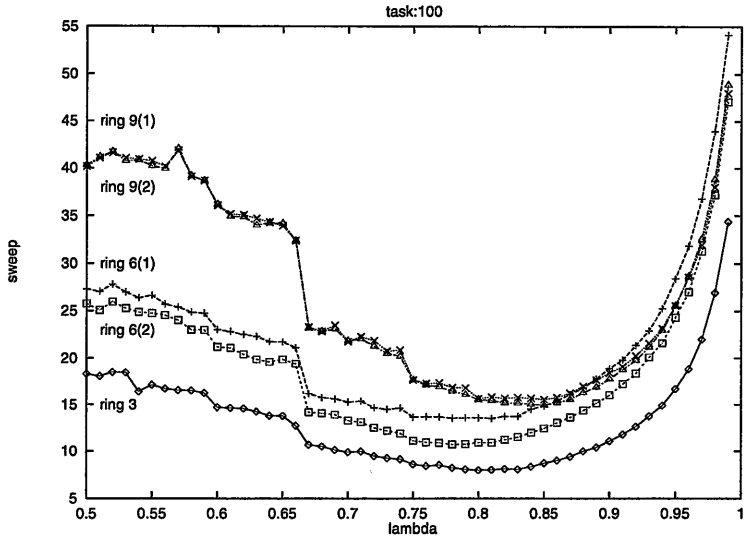

図 4.10: CCC 構造の負荷分散の例:平均タスク数が 100 の場合

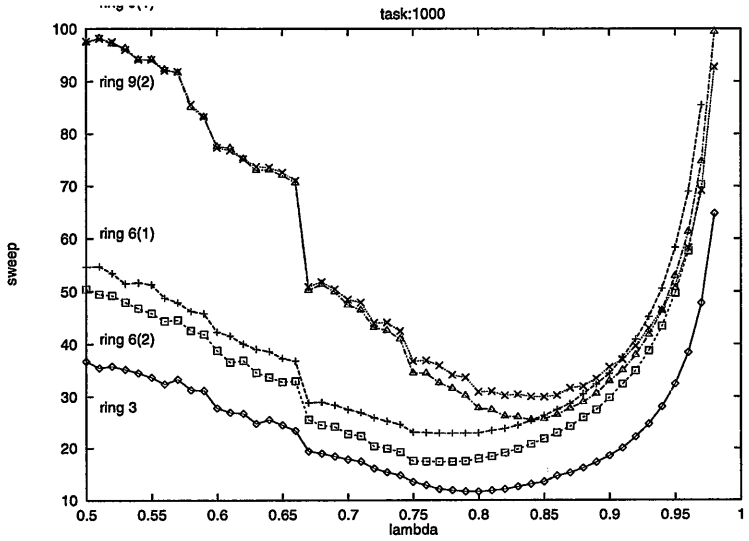

図 4.11: CCC 構造の負荷分散の例:平均タスク数が 1000 の場合

4.4 CCC の構造に関する問題

CCC 構造は,hypercube 構造と ring 構造と言う, 異なる性質を持つ構造を組み合わせて作られた構造である.

この事から,CCC において, 図 4.12のように, 異なる構造である ring 部分と hypercube 部分において異なる λ を適用することで, アルゴリズムの収束速度に影響を与えることが予想される.

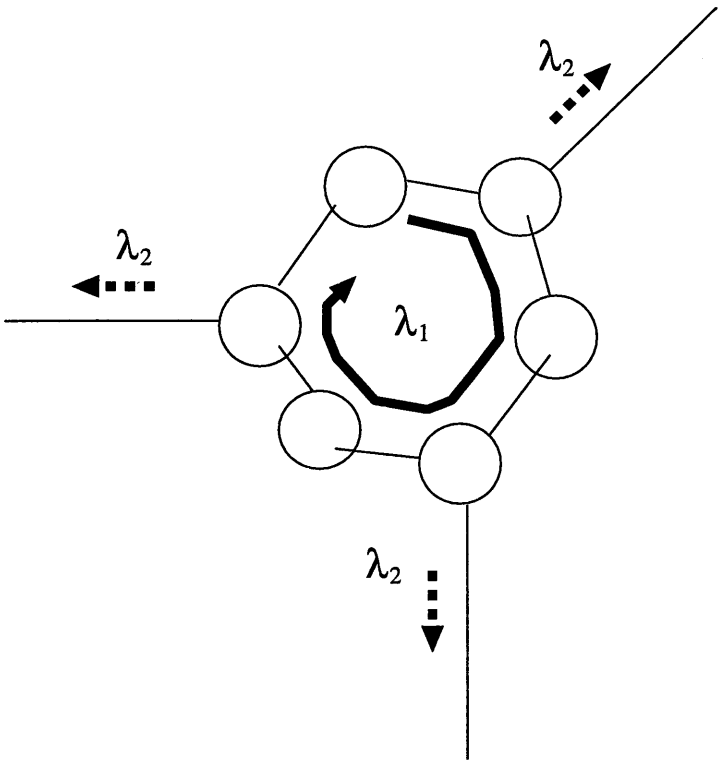

図 4.12: 構造の違いを考慮したアルゴリズム

その可能性を検証すべく, 以下の要領で実験を行った.

4.4.1 実験

ring 部分のプロセッサ数が 3,6(1),6(2),9(1),9(2) の CCC 構造プロセッサにおいて,ring の部分に λ_1,hypercube の edge に相当する部分に λ_2 という二つの交換パラメータ λ を適用して, 負荷分散の実験を行う.

最初に, この 2 つの λ を用いた場合の GDE 行列 $M(\lambda_1, \lambda_2)$ を求め, 各行列の第二固有値を最小にする交換パラメータ λ_1, λ_2 の値を算出した.

また, 以下の条件で負荷分散のシミュレーションを行った.

- 平均タスク数は 100 とする.
- 初期配置はランダムに配置する. タスク数の上限は平均タスク数の 2 倍とする.
- 初期配置のサンプル数は 200 通りを使用.
- λ_1,λ_2 の値を, それぞれ 0.5 ～ 0.99 の範囲で変化.

4.4.2 実験結果

単一の交換パラメータ λ を用いた場合, 並びに二つの交換パラメータ λ_1,λ_2 を用いた場合における GDE 行列の第二固有値の最小値について比較した結果を表 4.2, シミュレーション解析により収束に要した sweep 数と最適な交換パラメータの結果を表 4.3に示す.

	$min(\gamma_2(M(\lambda)))$	λ の最適値	$min(\gamma_2(M(\lambda_1, \lambda_2)))$	λ_1, λ_2 の最適値
3	0.4074	(0.66)	0.322	(0.51,0.80)
6(1)	0.7697	(0.73)	0.7478	(0.64,0.99)
6(2)	0.7214	(0.81)	0.6664	(0.67,0.99)
9(1)	0.7923	(0.83)	0.7823	(0.86,0.75)
9(2)	0.7800	(0.80)	0.6899	(0.72,0.99)

表 4.2: 単一の λ および二つの λ による固有値解析の比較

プロセッサ数	λ_1, λ_2 の最適値	sweep(λ_1, λ_2)	λ の最適値	sweep(λ)
3	(0.52,0.90)	7.38	(0.80)	7.98
6(1)	(0.76,0.93)	12.80	(0.81)	13.515
6(2)	(0.85,0.63)	10.575	(0.78)	10.73
9(1)	(0.89,0.66)	15.21	(0.85)	15.54
9(2)	(0.88,0.66)	14.675	(0.85)	14.94

表 4.3: 収束に要した sweep 数

表 4.2によると,GDE 行列の第二固有値の最小値 $min(\gamma_2(M(\lambda)))$ は, 従来の λ を一つだけ用いる方法よりも λ_1, λ_2 の二つの交換パラメータを用いた提案手法の方が, より小さな第二固有値を示している. よって, 交換パラメータ λ を二つ用いる手法の方が, 数値解析的にはより高速に収束すると思われる.

また, 表 4.3を見ると, 表 4.2の結果通り,2 つの λ を使った方が収束速度が若干増している. この事から,2 つの λ を用いる手法がより高速にアルゴリズムを収束させることが分かる.

4.5 まとめ

本章では,CCC 構造プロセッサの負荷分散において, 反復改良アルゴリズムである GDE 法を適用した際の有効性について調べた.

第二固有値による分析とシミュレーションによる結果から,ring 部分のプロセッサ数が 3,6(1),6(2),9(1),9(2) の CCC でアルゴリズムが最速に収束する λ を, 固有値による解析とシミュレーションから求めた. また, 同じ構造を持つ CCC 構造においても,edge coloring の違いによっては収束速度に変化をもたらす事が分かった.

負荷分散の完全性という点では, 完全二分木と同様, プロセッサの持つ負荷の平均タスク数が少ない場合守られなくなるが, 負荷の平均タスク数が並列処理プロセッサの直径よりも遥かに多い場合は完全性が増す.

また, 異なる構造の部分に対して, それぞれ異なる交換パラメータを用いる方法を提案し, その有効性を確かめた.

第5章

結論

本研究では, 並列ネットワークにおいで重要な機能である負荷分散について述べ, それに適合するアルゴリズムについて評価を行った.

構成要素として全て同じプロセッサが使われる並列処理プロセッサネットワークのような並列処理システムにおいては, 全プロセッサ間でなるべく均等な負荷が保たれるよう, なるべく完全な負荷分散がなされる事が望まれることから, 適合するアルゴリズムとして反復改良を挙げ, またその中から特に, 並列処理ネットワーク向けのアルゴリズムとしてGDE 法を採用し, 理論的な解析方法として, 第二固有値による解析法を述べた.

また,GDE 法によって tree 構造,CCC 構造の動的負荷分散のシミュレーション実験と固有値による解析を行い, 負荷の数がネットワークの直径に比べて充分に大きい程完全に近い負荷分散を行う事を示し, 各ネットワークに対して最速にアルゴリズムを収束させる交換パラメータ λ を求めた. また,tree においては, 負荷の分布によっても収束速度に違いがあることが分かった.

さらに,GDE 法では各ネットワークの edge coloring のパターンによって収束速度に違いが生じる事を示し, それらを考慮した GDE 法の改良法を示した.

今回は GDE 法のみについて議論と評価を行ったが,diffusion 法, TWA 法など, 並列処理ネットワークの負荷分散に有用と思われるアルゴリズムは他にも存在する. こうした, 他のアルゴリズムとの収束速度の性能評価などが, 今後の課題といえよう.

謝辞

本研究を進めるに当って，全般的に御指導いただいた，東北大学工学部 阿曽弘具教授に心より感謝いたします．

本研究をまとめるに当って，適切な御助言をいただいた，東北大学工学部 白鳥則郎教授，川又政征教授に大変感謝いたします．

いつも熱心に御討論していただいた，東北大学工学部助手 大町真一郎氏，森大毅氏，東北大学大学院工学研究科博士課程 菅谷至寛氏をはじめ，東北大学工学部通信工学科阿曽研究室並列グループの皆様に深く感謝いたします．

また，日頃より研究活動を共にし，計算機環境を常に快適な状態に整備していただいた，東北大学工学部通信工学科阿曽研究室の皆様，ならびに同研究室の同級生諸氏に深く感謝いたします．

参考文献

[1] C. Z. Xu, F. C. M. Lau: "Analysis of The Generalized Dimention Exchange Method for Dynamic Load Balancing", Journal of Parallel and Distributed Computing, .

[2] C. Z. Xu, F. C. M. Lau: "The Generalized Dimention Exchange Method for Load-Balancing in k-Ary n-Cube and Variants", Journal of Parallel and Distributed Computing,Vol.24,No.1, pp.72–85,Jan.1995.

[3] C. Z. Xu, F. C. M. Lau: "Iterative Dynamic Load Balancing in Multicomputers", Journal of the Operational Research Society,Vol.45,NO.7, pp.786–796. 1994.

[4] S. Fiorini, R. J. Wilson: "Edge-Coloring of Graphs", Selected Topics in Graph Theory,1978, (L. W. Beineke, R. J. Wilson,eds),pp. 103-125, Academic Press,New York.

[5] S. Ranka, Y. Won, S. Sahni: "Programming a Hypercube Multicomputer", IEEE Software, pp.69-77, Sept. 1988.

[6] G. Cybenko: "Dynamic Load Balancing for Distributed Memory Multiprocessors", Journal of Parallel and Distributed Computing, Vol.7, pp.279-301, 1989.

[7] M. Y. Wu: "On Runtime Parallel Scheduling for Processor Load Balancing", IEEE Transactions on Parallel and Distributed Systems, Vol.8, No.2, pp.173-186, Feb. 1997.

[8] D. L. Eager, E. D. Lazowska, J. Zahorjan: "A Comparison of Receiver Initiated and Sender Inisiated Adaptive Load Sharing", Performance Evaluation, Vol.6, 1986, pp. 53-68.

[9] F. C. H. Lin, R. M. Keller, "The Gradient Model Load Balancing Method", IEEE Trans. Software Eng., Vol.13, No.1, Jan.1987,pp. 32-38.

[10] K. K. Goswami, M. Devarakonda, R. K. Iyer: "Prediction-Based Dynamic Load-Sharing Heuristics", IEEE Trans. Parallel and Distributed Systems, Vol.4, No.6, June.1993, pp. 638-648.

[11] D. J. Evans, W. U. N. Butt: "Dynamic Load Balancing Using Task-Transfer Probabilities", Parallel Computing, Vol.19, No.8, Aug.1993, pp. 897-916.

[12] R. A. Becker, J. M. Chambers, A. R. Wilks （渋谷政昭, 柴田里程訳）: "S 言語 データ解析とグラフィックスのためのプログラミング環境 I,II" 共立出版株式会社, 1991.

[13] 鶴田進: "並列分散処理システムに関する研究", 東北大学大学院工学研究科修士論文, 1997.

研究業績

- “GDE 法による完全二分木構造プロセッサの負荷分散”
  遠藤貴之，阿曽弘具
  1998 年 電気関係学会東北支部連合大会, 2H-8 (1998-8)