アルゴリズム 34－6
（1993．8．19）

シミュレーティド・アニーリングによる静的タスク配置へのヒューリスティクスの適用とその評価

堀川和雄†，森 眞一郎†，中島浩†，富田眞治†
†：京都大学 工学部

タスクインタラクショングラフ (TIG) で表現される並列アプリケーションをマルチコンピュータに静的にタスク配置する方法として，局所解を避けてよい結果を出すシミューレーティドアニーリング法がある．しかしこの手法は収束に時間がかかるという欠点があった．本稿では，タスク配置問題固有の性質を基にするヒューリスティクスにより，タスク–プロセッサ配置関数の生成確率を変更し，この点を改善する．ヒューリスティクスとしては負荷均一化を図る2種とプロセッサ間通信削減を図るものを提案し，ランダムなTIGに対してこれらの手法を適用した結果，負荷均一化を図るヒューリスティクスは初期に特に効果的で，プロセッサ間通信を削減するヒューリスティクスは収束が近付いた段階で効果的であると分かった．

Simulated Annealing Heuristics for Static Task Assignment

Kazuo HORIKAWA†, Shin-ichiro MORI†,
Hiroshi NAKASHIMA†, Shinji TOMITA†

†: Department of Information Science
Faculty of Engineering, Kyoto University
Yoshida-hon-machi, Sakyo-ku, Kyoto 606–01 Japan

E-mail: {horikawa, moris, nakasima, tomita}@kuis.kyoto-u.ac.jp

Simulated annealing is a method that performs effectively and avoids local optima traps for static task assignment of parallel applications, which are represented by task interaction graph(TIG), on a multicomputer. But this method takes long time for the convergence. We speed it up by modifying the generation of neighbor mapping function with heuristics based on the characteristics of task mapping problem. Two heuristics for load balancing and one for reducing interprocessor communications are proposed. They are applied for some random TIG. The former are very effective on the beginning, and the latter is effective near the convergence in the simulated annealing process.

1 はじめに

タスク配置問題とは，マルチコンピュータ上で並列アプリケーションを実行させようとした場合に，どのプロセッサでどのタスクを実行したらよいのかを決定する問題である．本稿ではシミュレーティドアニーリング (simulated annealing:SA) を用いた解法にヒューリスティクスを適用する方法を提案する．

並列計算機で並列アプリケーションを実行するとき，そのアプリケーションの並列処理の最小単位をタスクと呼ぶ．タスク配置の目的は，1) プロセッサ間通信を削減し，2) 負荷を分散し，3) 並列性を得るところにある．

本稿で扱う並列アプリケーションは，後に述べるタスクインタラクショングラフ (task interaction graph:TIG) で表すことができるもののみを扱う (図1)．このモデルでは，タスクは原子的な処理単位であり，SIMD 的に動作する．即ち，タスクは相互の依存関係無しに独立に処理を行い，その後，通信関係のあるタスク間で相互に計算した値を交換するといった処理を繰り返し行う．

並列アプリケーションのプログラム実行時間はタスク配置によって大きく左右されるので，実行時間の短くなるタスク配置を求めることが重要となる．

タスク配置問題を解く方法として，Kernighan-Lin のグラフ2分割手法やその応用 [1][2] に代表されるヒューリスティックな解法や，グラフ理論的なアプローチ [3] などがあった．また SA を使用する方法も以前から検討されている [4][5]．

SA は汎用的な最適化手法で，理論的には必ず最適解に到達することが保証されている．しかし SA には，温度と呼ばれるパラメタのスケジューリングが難しいことと，解の収束に長大な時間が必要であることの2つの問題がある．

本稿では，SA の隣接解生成を改善して収束に至る繰り返し数を削減することにより，SA を用いたタスク配置法の収束を早める手法を提案する．

以下，2章で静的タスク配置問題の定式化，3章で SA のタスク配置問題への適用とヒューリスティクスの使用の説明，4章でその評価，5章で全体のまとめをおこなう．

2 静的タスク配置問題の定式化

本章ではまず，対象となる並列アプリケーションと並列計算機をモデル化する．並列アプリケーションはタスクインタラクショングラフ (task interaction graph:TIG) で，タスク配置の対象となる並列計算機アーキテクチャはプロセッサコミュニケーショングラフ (processor communication graph:PCG) で，それぞれモデル化できる．その次に，配置問題が最小とすべき目的関数を定義する [6][7]．

2.1 TIG

TIG は各タスクの計算に必要な時間とタスク間で交換されるデータの量を表す．TIG は無向グラフ $G_T(V,E)$ で表され，0 から $N-1$ までのラベルづけされた $|V|=N$ 個の節を持つ (図1)．

TIG の節は並列アプリケーションの原子的なタスクを表し，節の重み w_i はタスク i の計算コストを表す．

TIG の枝はタスク i,j 間の通信関係を表し，(i,j) の2つ組で表される．枝の重み $e_{(i,j)}$ (ただし $(i,j)\in E$) はタスク i,j 間の通信量を表す．

2.2 PCG

PCG は各プロセッサ間の通信に関する特性を表す完全な無向グラフ $G_P(P,D)$ である．PCG は0から $K-1$ までのラベル付けされた $|P|=K$ 個の節と $|D|={}_K\mathrm{C}_2$ 個の枝を持つ．

PCG の節はプロセッサを表し，各プロセッサの演算能力は等しいものとする．

PCG の枝は，(p,q) のような2つ組で表され，プロセッサ p,q 間の通信を表現する．枝の重み $d_{(p,q)}$ はプロセッサ間通信に要するコストを表す．

プロセッサ p,q 間通信のコスト $d_{(p,q)}$ は，プロセッサ p,q 間で通信されるメッセージ本体の総量 l を用いて，式1で表されるものとする [8][9][10][11]．

$$d_{(p,q)} = T_s + T_l \cdot l \tag{1}$$

また，同一プロセッサ上のタスク間通信に必要な通信時間は，プロセッサ間通信に要する時間と比べて十分無視できるとする．

2.3 タスク配置に与えられる目的関数

タスクからプロセッサへの配置関数を $M: V \to P$ とし，$M(i)$ をタスク i が配置されるプロセッサ番号とする．

プロセッサの実行時間は演算の時間とプロセッサ間通信の時間とに分けられる．演算は配置されたタスクの処理に必要であり，プロセッサ間通信は，配置されたタスクが異なるプロセッサに配置されたタスクと通信を行う場合に必要となる．

プロセッサ p,q 間の通信量 $COMM_{(p,q)}$ は，

$$COMM_{(p,q)} = \sum_{\{\forall i | M(i)=p\}} \sum_{\{\forall j | M(j)=q\}} e_{(i,j)} \tag{2}$$

プロセッサ p,q 間の通信にかかる時間 $T_{comm(p,q)}$ は，

$$T_{comm(p,q)} = \begin{cases} T_s + T_l \cdot COMM_{(p,q)} & (\text{if } COMM_{(p,q)} > 0) \\ 0 & (\text{if } COMM_{(p,q)} = 0) \end{cases} \quad (3)$$

プロセッサ p がすべての通信にかかる時間 $T_{total\_comm}(p)$ は，

$$T_{total\_comm}(p) = \sum_{q \in P} T_{comm(p,q)} \quad (4)$$

とそれぞれ表される．

また，プロセッサ p が計算に要する時間 $T_{total\_calc}(p)$ は，

$$T_{total\_calc}(p) = \sum_{\{\forall i | M(i)=p\}} w_i \quad (5)$$

と表され，各プロセッサが 1 イタレーションに要する時間 $T(p)$ は

$$T(p) = T_{total\_calc}(p) + T_{total\_comm}(p) \quad (6)$$

と表される．従って，システム全体で 1 イタレーションに要する時間 T_{max} は，

$$T_{max} = \max_{p \in P} T(p) \quad (7)$$

であり，この目的関数 (スコア) を最小とするようなタスク配置を求めることが，タスク配置問題の最終目標である．

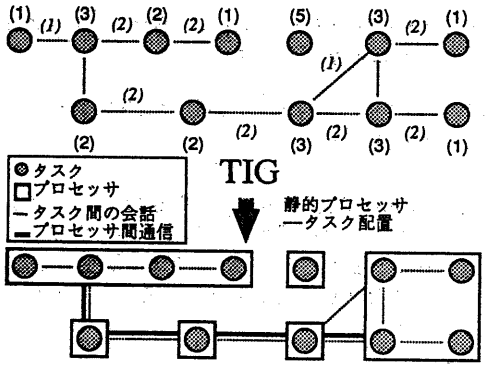

図 1: TIG と静的タスク配置

3 シミュレーティドアニーリング

SA は，広範囲の組み合わせ最適化問題に適用可能な汎用的な最適化手法の一つである [12][13][14].

SA では状態空間の探索において，パラメタにより制御された hill-climbing (解を悪化させるような状態変化) を許容していることにより局所的な最適解 (local optima) に落ち込む危険性を排除している．このパラメタは温度と呼ばれ，最適化されるべき目的関数はエネルギーと呼ばれる．

以下のように記号を定義する．

x ：解

U ：最小とすべき目的関数，解 x のエネルギーは $U(x)$

c ：温度

x_0 を初期解，解の系列を $\{x_n\}$ $n = 0, 1, .., \tau$ とするとき，この解の系列は以下で得られる．

1. **解の生成**
   現在の解 x_n をランダムに微小変化させて新たな解 x'_{n+1} を生成する．x'_{n+1} を隣接解とも言う．ただし $x'_{n+1} \neq x_n$ である．
2. **解の受理判定**
   生成した解 x'_{n+1} を受理するかどうか判定する．
   (a) 受理：確率 a で $x_{n+1} = x'_{n+1}$ とし，
   (b) 非受理：確率 $1-a$ で $x_{n+1} = x_n$ とする．
3. **温度の更新**
   温度の系列は $\{c_n\}$ で与えられる．
4. **終了判定**
   終了条件を判定し，終了しないなら 1 から繰り返す．

ただし，

$$a = \begin{cases} 1 & (\text{if } U(x'_{n+1}) < U(x_n)) \\ \exp\left(-\frac{U(x'_{n+1}) - U(x_n)}{c_n}\right) & (\text{otherwise}) \end{cases} \quad (8)$$

である．

また，温度 c において以下を定義する．

解 現在の解を x，隣接解を y と書く．

生成確率行列 生成確率行列 $G(c) = (g_{xy}(c))$ は現在の解が x のとき解 y を排他的に生成する確率を表す．

受理確率行列 受理確率行列 $A(c) = (a_{xy}(c))$ は現在の解が x で，隣接解として y が生成されたとき，この隣接解を受理する確率を表す．

遷移確率行列 遷移確率行列 $R(c) = (r_{xy}(c))$ は現在の解が x のとき解 y に遷移する確率を表し，以下の性質を持つ．

$$r_{xy}(c) = g_{xy}(c) \cdot a_{xy}(c) \quad (9)$$

$$r_{xx}(c) = 1 - \sum_{y \in \{x \text{ の隣接解 }\}} r_{xy}(c) \quad (10)$$

3.1 SAの静的タスク配置への適用

SAを静的タスク配置に適用する場合，解x，目的関数U，隣接解x'は次のようになる．

解x TIGからPCGへの写像関数Mである．

目的関数U プロセッサへタスク配置したときの実行時間を表すT_{max}を用いる．

隣接解x' Mを微小変化させたM'であり，以下のように生成される．

1. N個のタスクの中から任意の1つのタスクiを選択し，
2. $M'(i) \neq M(i)$なるように，$K-1$個のプロセッサからiが新たに配置されるプロセッサ$M'(i)$を選択する．

即ち隣接解であるタスク配置M'は以下のようになる．またどの隣接解も一様の確率で生成される．

$$M'(i) \neq M(i) \tag{11}$$
$$M'(j) = M(j) \quad (^{\forall}j|j \neq i) \tag{12}$$

SAでは，温度スケジューリングの難しさと，収束に要する長大な実行時間と言う2つの問題点があった．後者では特に低温度での解の非受理が深刻である．本稿ではSAの収束を早めることにだけ注目する．

3.2 SAの高速化

SAの解の収束を早めるために，受理されやすい解や最適解に近付く解をより高い確率で生成するように，解の生成を改善することを考える．

本稿ではタスク配置問題固有の性質を定性的に解析し，目的関数値の変化を予測することにより，解の生成確率を操作する．これを隣接解生成におけるヒューリスティクスの利用と呼ぶ．

ヒューリスティクスの利用により，より少ない繰り返し数で，よりよい解に到達できることが予想できる．半面欠点としては局所最適解から脱出しにくいことと，ヒューリスティクスを用いることによって余計に生ずる計算の影響が考えられる．

3.2.1 ヒューリスティクスの導入方法

元のSAでは隣接解の生成確率は一様であった．

本稿では隣接解の生成確率を非0の範囲で変更する．この方法では隣接解自体は元のSAと同一であり，それぞれの隣接解が選択される確率だけが異なったものとなる．このとき，目的関数値を減少させそうな隣接解が選択される確率を大きくして，目的関数値を大きくしそうな隣接解が選択される確率を小さくするようにする．また，解の受理判定は元のSAと同一の手続きを用いる．

なおこの方法はSAが正しく機能するための以下の条件を満たす

条件：遷移確率行列が

1. 対称的 (symmetric):xがyの隣接解であるときまたそのときのみyはxの隣接解．

$$r_{xy}(c) > 0 \text{ iff } r_{yx}(c) > 0 \tag{13}$$

2. 既約 (irreducible):すべてのxからすべてのyに到達可能．

$$^{\forall}x, ^{\forall}y, ^{\exists}n, r_{xy}^{(n)}(c) > 0 \tag{14}$$

かつ受理確率が

$$a_{xz}(c) \cdot a_{zy}(c) = a_{xy}(c) \tag{15}$$

を満たすとき，適切な温度スケジューリングのもとでSAは確率1で最適解に収束する [15]．

3.2.2 ヒューリスティクス

本稿ではタスク選択とプロセッサ選択とに分けて，ヒューリスティクスを考え，合計3種のヒューリスティクスを提案する．それぞれ次のように名前を付ける．

A. Heavy：タスク選択で負荷の大きいプロセッサに属するタスクほど選ばれやすい．

B.1 Light：プロセッサ選択で負荷の小さいプロセッサほど選ばれやすい．

B.2 Neighbor：プロセッサ選択で隣接プロセッサほど選ばれやすい．

A. タスク選択の方法:Heavy 隣接解の内，ある条件を満たすタスクを選択する解について優先して選択することを考える．Heavyでは，負荷$T(p)$の大きいプロセッサに配置されているタスクを優先して選択することとする．

即ち，

$$0 \leq random < 1 \tag{16}$$

$$s = \begin{cases} \lfloor -\log_2(random) \rfloor & (\text{if } random > 2^{-K}) \\ K-1 & (\text{otherwize}) \end{cases} \tag{17}$$

としたときに，負荷$T(p)$がs番目に大きいプロセッサを選択し，そのプロセッサの中から等確率でタスクを1つ選択する．ただし，$0 \leq s \leq K-1$である．

B. タスクの移動先プロセッサ選択の方法　隣接解の内，ある条件を満たすプロセッサを選択する解について高確率で選択することを考える．本稿では以下の2つを考える．

B.1. 負荷の軽いプロセッサ優先:Light　式16, 17を用いて，負荷$T(p)$がs番目に小さいプロセッサを選択する．

B.2. TIG でのトポロジカルな選択:Neighbor　プロセッサの選択で，現在の解で選択されたタスクが配置されているプロセッサの隣接プロセッサを高確率で選択する．なお，プロセッサpの隣接プロセッサ集合ADJ[p]は以下のように定義される．

$$\mathrm{ADJ}[p] = \{q|^{\exists}i, {}^{\exists}j, p = M(i), q = M(j), (i,j) \in E\} \tag{18}$$

隣接プロセッサとは，物理的なプロセッサ間リンクのことではなく，それらのプロセッサ間に通信関係が存在することを意味している．

タスクをプロセッサ間で移動させるとき，通信を減少させるか通信を増やさないようすることが望まれる．

隣接プロセッサを選択する場合には，プロセッサ間通信が減少したり変化しないことが多い．このため Neighbor では，プロセッサ選択において隣接プロセッサが選択されやすいようにする．

具体的には，確率 b で 1. を，確率 $1-b$ で 2. を実行する．

1. 選択されたタスクが配置されている隣接プロセッサをランダム等確率で選択する．
2. 全プロセッサからランダムに選択する．

以上の3つのヒューリスティクスの特徴をまとめると，Heavy, Light は負荷を均等化しようとするヒューリスティクスであり，Neighbor はプロセッサ間通信を削減しようとするヒューリスティクスであると言える．

実際に本稿で提案するヒューリスティクスを SA に適用するとき，表1に示すヒューリスティクスの組合せを用いる．

3.3 関連研究

SA の高速化の方針として 1) ハードウェアの高速化，2) 並列化による高速化 [14][13]，3) SA 自体の高速化，の3種の方法がある．本稿で提案する方法は SA 自体の高速化である．

SA 自体の高速化として，生成する解をすべて受理する SA が提案されている [16]．この方法は，全ての隣接解について目的関数値の変化を解の生成の前に計算しておき，解の生成確率にこれを反映させる．そのかわりに解の受理確率を 1 とする．

表 1: ヒューリスティクスを組み合わせた方法

方法名	タスク	プロセッサ	パラメタ
ORG	-	-	
NE+	-	Neighbor	$b = 0.9$
NE	-	Neighbor	$b = 0.5$
HV	Heavy	-	
LT	-	Light	
HL	Heavy	Light	

-:ヒューリスティクスを用いない

即ちこの手法では隣接解の生成行列 $G'(c) = (g'_{xy}(c))$ と受理行列 $A'(c) = (a'_{xy}(c))$ は次式で表される．

$$g'_{xy}(c) = \frac{r_{xy}(c)}{\sum_{z \in \{x \text{ の隣接解}\}} r_{xz}(c)} \tag{19}$$

$$a'_{xy}(c) = 1 \tag{20}$$

しかし，この方法をタスク配置問題に適用させようと考えたとき，隣接解数は $N \cdot (K-1)$ であるので，そのそれぞれについて G' を SA の毎世代計算しなければならないこの方法は現実的ではない．

4 実験

本稿で提案するヒューリスティクスを適用する SA の評価を実験的な解析に基づいて行う．

SA を AP1000 のセルプロセッサ上で実行した (SPARC:25MHz)．疑似乱数を用い，各々の検査項目に対して10種の乱数の種に対して実験した．

まず初期配置がランダムであるものについて本手法を適用して評価する．その次にランダムな初期配置にヒューリスティクスの強い反復改善法 (前処理と呼ぶ) を用いたあと，提案する方法を適用して評価する．

TIG　表2に表されるランダムグラフを3つ生成した．

各タスクの計算量w_iを式21とし，タスク間の通信量を$e_{(i,j)} = 1$とする．

これらのパラメタは電力潮流計算で非線形連立方程式を Jacobi 法で解くプログラムを反映するものである．

$$w_i = 18 \cdot (|\{^{\forall}(i,j)|(i,j) \in E\}| + 3) \tag{21}$$

PCG　表3の2つの PCG を扱う．

また通信パラメータを，$T_s = 40, T_l = 4$ と定める．これは既存のマルチコンピュータ AP1000 を参考にした [10][11]．

表 2: 実験に用いた TIG

名前	$\|V\|$	$\|E\|$
256t	256	512
512t	512	1024
1024t	1024	2048

表 3: 実験に用いた PCG

名前	$\|P\|$
32p	32
64p	64

4.1 前処理無しの場合

まず初期配置が各タスクをランダムにプロセッサに割り付けたものに対して実験を行った.

温度スケジューリングは,

$$c_n = c_0 \cdot \alpha^{\lfloor n/\kappa \rfloor}$$
$$\alpha = 0.95, \quad \kappa = \tau/100 \qquad (22)$$

を用いた.

SA の長さτは$\tau = 10000, 20000, 40000, 80000$ の 4 つを試し, 初期温度 c_0は経験的に $c_0 = 200$ とした.

表 4は方法, τ, TIG と PCG ごとの最終的なタスク配置で得られるスコア T_{max} を正規化した U' を表している. ただし,

$$U' = \frac{T_{max}}{(\sum_{i \in V} w_i)/K} \qquad (23)$$

である. また表 4は括弧内の値は ORG との比である. 図 3に表 4 の 256t64p についてだけ抽出したのグラフを示す.

HV, LT, HL は ORG に比べてよい結果を出している. NE+, NE は ORG とほとんど差はないか, 逆に ORG に劣る. τ が小さいときほどこれらの差は大きい.

ORG と HL のスコアの差を$\tau = 80000$ のときに比較した場合, 256t64p で 10% で最大, 1024t32p で 1% と最小になり, Nが増えるにつれ, また K が減るにつれ, 差は小さくなる傾向があり, K の影響の方が大きい.

図 2は n に対する U'の変化を表す. HV, HL は急激に U'を小さくして $n = 5000$ あたりから平坦になるのに対し, LT は徐々に U'を低下させているのが読み取れる.

図 4は生成される隣接解が受理された確率を表す. ORG, NE+, NE はほぼ 70〜80% の受理率であるが, HV, HL の受理率は 約 30% と低いものになっている. LT は HV, HL と同程度の U'が得られているのに, 80%程度の高い受理率である. また K, Nは変化しても受理率はそれほど変化しなかった.

図 5は SA の実行時間を表す. インプリメントに大きくよるのだが, 本稿で用いたプログラムでは実行時間は受理率に大きく依存し, 受理率が低いと実行時間も多くかかった. また繰り返し回数に比例して実行時間が長くなる.

4.2 前処理ありの場合

次に初期配置に前処理を行い, ある程度スコアを低下させた後, 提案する方法を始める実験する. 前処理を行わない場合には, τが小さいときに ORG, NE+, NE は反復改善法にも劣るためである.

具体的に前処理とは以下の処理を言う.

1. まずランダム配置を行い,
2. 次に $c_0, \cdots, c_{1000} = 0$ において HL を 1000 世代経過させた後,
3. ORG, NE+, NE は $c_0 = 10$, HV, LT, HL は $c_0 = 100$ として, ランダム初期配置で用いたものと同じ温度スケジューリングを用いて実行.

前処理後はエネルギーをかなり低下しているので, 低い温度から SA を始めるのが妥当である. しかし, HV, LT, HL は目的関数値を効率よく低下させる能力は高い反面, 局所解から脱出する能力が元の SA と比べて低いため, 比較的高い温度で SA を開始する.

256t64p への適用結果のグラフを図 6に示す. NE+ から HL までそれぞれ 2.9%, 2.7%, 3.5%, 3.0%, 3.7%ずつ, ORG よりよい結果を出している. 前処理無しと比較してみると, HV, LT, HL ではあまり変化が生じないが, NE+, NE が目覚しくよい結果を出している.

256t64p の受理率を図 7に示す. ORG, NE+, NE は 1/6 から 1/8 程度に, HV, HL は 1/3 程度に, 前処理無しの受理率と比較して低下しているが, LT はほとんど低下していない. N, Kに対する変化は前処理無しと同じ振舞であった.

4.3 議論

前処理無しの目的関数値 U' については, HV, HL が非常によく, ついで LT がよい. これにより, 初期段階では負荷均等化を計るヒューリスティクス Heavy, Light が有効であると分かる.

また, 目的関数 (式 7) の性質から特に Heavy が有効である. K が大きくなるか, N が小さくなるとスコアの差が大きくなるのは, プロセッサに配置される平均タスク数が小さいと, それだけ各プロセッサへ割り当てられるタスク数の分散が大きくなるためと考えられる.

HV, HL の受理率の低さは, 早い世代において急激にスコアを低下させているため, 中間以降の世代で生成される解のほとんどが目的関数値を悪化させる解となり, 受理されにくくなっているためと思われる. U' がゆっくり低下していることも LT の受理率の高さを裏付けている.

以上から Heavy は急激に負荷均等化を行い，Light は徐々に負荷均等化を行うといえる．

SA の実行時間は受理率に大きく依存し，ヒューリスティクスの使用の影響は少ない．

前処理ありの目的関数値について NE+, NE が HV, LT, HL と大差無くよい結果を出したことから，収束が近付いて，各プロセッサの負荷がほぼ均等になってきたときに，プロセッサ間通信を削減する Neighbor が有効であるといえる．

前処理ありの受理率について，LT 以外の受理率がかなり低くなっているのは，前処理によってかなりスコアを下げられているためであると考えられる．

LT だけが常に受理率が高いのは，比較的局所解から脱出する能力が高いためと考えられる．

全体的にまとめると，

1. 負荷の均等化を行う Heavy, Light が初めは効果的に目的関数を低下する．
2. 負荷がほとんどのプロセッサで均一になっていくと，通信による負荷を削減する Neighbor が有効になる．
3. Light, Neighbor は局所解から脱出する能力が高く，Heavy は低い．

5 おわりに

TIG で表される並列アプリケーションをマルチコンピュータに静的にタスク配置する方法として，SA を利用するときに，収束するまでに長大な時間がかかることが問題であった．

本稿では SA の解の生成を工夫することにより収束を早める手法を提案した．タスク配置問題固有の性質を解析することにより，隣接解の生成確率を変化させ，ヒューリスティクスとしては負荷均一化を図る Heavy, Light，およびプロセッサ間通信削減を図る Neighbor を提案した．

ランダムな TIG に対してこれらの手法を適用した結果，Heavy, Light は初期に特に効果的であり，Neighbor は目的関数の収束が近付いた段階でよい成果を出した．

本稿で示した以外にもヒューリスティクスは無数に考えられる．また実際のタスク配置手法として用いるときに，ヒューリスティクスの組合せ，それぞれのヒューリスティクスで用いられるパラメタのチューニング，理論的な温度スケジューリングなどが課題として考えられる．

謝辞

並列計算機 AP1000 の実行環境を御提供戴いている (株) 富士通研究所，プログラムの開発環境の御支援を戴いている横河・ヒューレット・パッカード (株) の川島貴子氏，千住晃夫氏ならびに板鼻弘太郎氏に対し，ここで深く感謝致します．

また本研究に対し，日頃より御討論頂く富田研究室の諸氏に感謝致します．

参考文献

[1] Kernighan, B. W. and Lin, S.: An Efficient Heuristic Procedure for Partitioning Graphs, *The Bell System Technical Journal*, Vol. 49, No. 2, pp. 291 – 307 (1970).

[2] Ercal, F., Ramanujam, J. and Sadayappan, P.: Task Allocation onto Hypercube by Recursive Mincut Bipartitioning, *Journal of Parallel and Distributed Computing*, Vol. 10, pp. 35 – 44 (1990).

[3] Lo, V. M.: Algorithms for Static Task Assignment and Symmetric Contraction in Distributed Computing Systems, in *International Conference on Parallel Processing*, Vol. II, pp. 239 – 244 (1988).

[4] Sheild, J.: Partitioning Concurrent VLSI Simulation Programs onto a Multiprocessor by Simulated Annealing, *IEE proceedings*, Vol. 134, No. 1, pp. 24 – 30 (1987).

[5] Ramanujam, J., Ercal, F. and Sadayappan, P.: Task Allocation by Simulated Annealing, in *Third International Conference on Supercomputing*, Vol. III, pp. 471 – 480 (1983).

[6] Sadayappan, P., Ercal, F. and Ramanujam, J.: Cluster Partitioning Approaches to Mapping Parallel Programs onto a Hypercube, *Parallel Computing*, Vol. 13, pp. 1 – 16 (1990).

[7] Bultan, T. and Aykanat, C.: A New Mapping Heuristic on Mean Field Annealing, *Journal of Parallel and Distributed Computing*, Vol. 16, pp. 292 – 305 (1992).

[8] 大河内俊夫, 金野千里, 猪貝光祥：数値シミュレーション向き高水準言語 DEQSOL の分散メモリ型並列計算機向けトランスレーションに関する一考察, 並列処理シンポジウム JSPP'93, pp. 39 – 46 (1993).

[9] 岡本秀輔, 緑川博子, 飯塚肇：分散記憶型マルチプロセッサ nCUBE2 の性能評価, Technical Report 52, 成蹊大学 (1991).

[10] 清水俊之, 堀江健志, 石畑宏明：高速メッセージハンドリング機構-AP1000 における実現-, 並列処理シンポジウム JSPP'92, pp. 195 – 202 (1992).

[11] 清水俊之, 堀江健志, 石畑宏明：AP1000 の性能評価-メッセージハンドリング, 放送, バリア動機の効果-, 92-ARC-95, 情報処理学会 (1992).

[12] Kirkpatrick, S., Gelatt, C. and Vecci, M.: Optimization by Simulated Annealing, *Science*, Vol. 220, No. 4598, pp. 671 – 680 (1983).

[13] Aarts, E. and Korst, J.: *Simulated Annealing and Boltzmann Machines*, Wiley (1989).

[14] van Laarhoven, P. J. M. and Arts, E. H. L.: *Simulated Annealing: Theory and Applications*, D. Reidel (1989).

[15] Faigle, U. and Schrader, R.: On the Convergence of Stationary Distributions in Simulated Annealing Algorithms, *Information Processiong Letters*, pp. 189 – 194 (1987).

[16] Greene, J. W. and Supowit, K. J.: Simulated Annealing Without Rejected Moves, *IEEE Trans. Comput.-Aided Design Integrated Circuits*, Vol. CAD-5, No. 1, pp. 221 – 228 (1986).

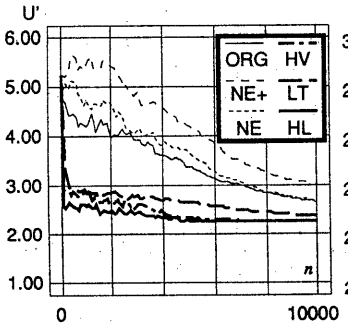

図 2: スコアの変化 (前処理無し, 1024t64p, $\tau = 10000$)

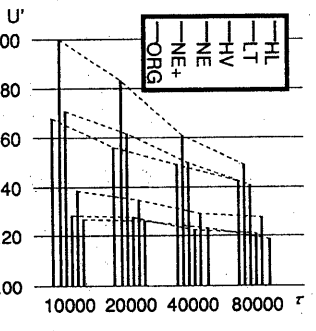

図 3: 正規化スコア (前処理無し, 256t64p)

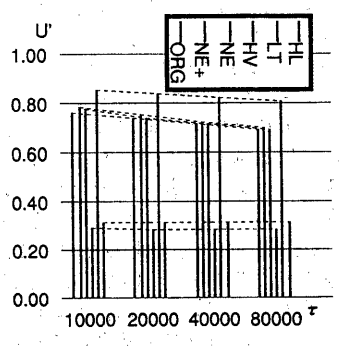

図 4: 受理率 (前処理無し, 1024t64p)

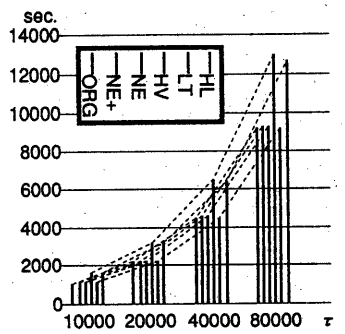

図 5: SA の実行時間 (前処理無し, 1024t64p)

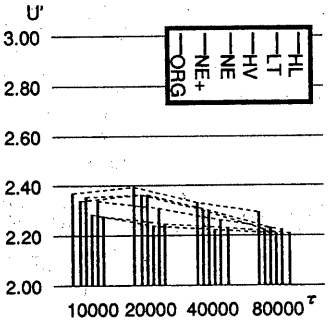

図 6: 正規化スコア (前処理有り, 256t64p)

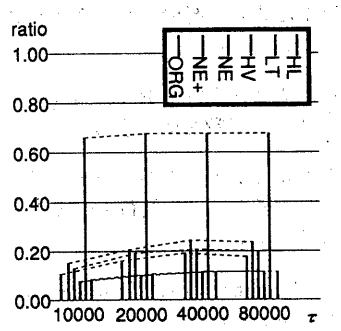

図 7: 受理率 (前処理有り, 256t64p)

表 4: 正規化スコア (前処理無し)

方法	問題 \ τ	10000	20000	40000	80000	問題 \ τ	10000	20000	40000	80000
ORG		3.89(1.00)	3.80(1.00)	3.74(1.00)	3.69(1.00)		2.68(1.00)	2.57(1.00)	2.50(1.00)	2.43(1.00)
NE+		3.90(1.00)	3.82(1.01)	3.72(0.99)	3.67(0.99)		3.01(1.12)	2.84(1.11)	2.61(1.05)	2.49(1.03)
NE	256t	3.89(1.00)	3.80(1.00)	3.72(0.99)	3.67(0.99)	256t	2.71(1.01)	2.62(1.02)	2.51(1.00)	2.41(0.99)
HV	32p	3.67(0.94)	3.61(0.95)	3.58(0.96)	3.53(0.96)	64p	2.28(0.85)	2.28(0.89)	2.23(0.89)	2.21(0.91)
LT		3.70(0.95)	3.62(0.95)	3.58(0.96)	3.53(0.96)		2.39(0.89)	2.35(0.92)	2.30(0.92)	2.28(0.94)
HL		3.63(0.93)	3.60(0.95)	3.56(0.95)	3.50(0.95)		2.27(0.85)	2.27(0.88)	2.23(0.89)	2.19(0.90)
ORG		3.36(1.00)	3.31(1.00)	3.28(1.00)	3.25(1.00)		2.31(1.00)	2.27(1.00)	2.20(1.00)	2.16(1.00)
NE+		3.37(1.00)	3.31(1.00)	3.28(1.00)	3.24(1.00)		2.46(1.06)	2.33(1.03)	2.22(1.01)	2.14(0.99)
NE	512t	3.37(1.00)	3.31(1.00)	3.28(1.00)	3.25(1.00)	512t	2.33(1.01)	2.26(1.00)	2.19(1.00)	2.15(0.99)
HV	32p	3.25(0.97)	3.22(0.97)	3.19(0.97)	3.16(0.97)	64p	2.06(0.89)	2.03(0.90)	2.01(0.91)	1.97(0.92)
LT		3.25(0.97)	3.21(0.97)	3.18(0.97)	3.16(0.97)		2.14(0.92)	2.10(0.93)	2.07(0.94)	2.05(0.95)
HL		3.22(0.96)	3.19(0.96)	3.17(0.97)	3.14(0.97)		2.03(0.88)	2.00(0.88)	1.97(0.90)	1.96(0.91)
ORG		2.89(1.00)	2.88(1.00)	2.87(1.00)	2.86(1.00)		2.02(1.00)	1.98(1.00)	1.94(1.00)	1.91(1.00)
NE+		2.90(1.00)	2.88(1.00)	2.87(1.00)	2.86(1.00)		2.06(1.02)	2.00(1.01)	1.95(1.00)	1.90(1.00)
NE	1024t	2.90(1.00)	2.88(1.00)	2.87(1.00)	2.85(1.00)	1024t	2.04(1.01)	1.98(1.00)	1.94(1.00)	1.91(1.00)
HV	32p	2.85(0.99)	2.85(0.99)	2.84(0.99)	2.83(0.99)	64p	1.86(0.92)	1.83(0.93)	1.81(0.93)	1.79(0.94)
LT		2.85(0.99)	2.84(0.99)	2.83(0.99)	2.83(0.99)		1.90(0.94)	1.88(0.95)	1.86(0.96)	1.84(0.96)
HL		2.85(0.99)	2.85(0.99)	2.84(0.99)	2.83(0.99)		1.84(0.91)	1.82(0.92)	1.79(0.92)	1.77(0.93)