並列処理による進化計算の高速化について

筒　井　茂　義

Ⅰ．はじめに

生物進化にヒントを得た探索手法である進化計算は，1970 年代から基礎研究が始まった。1990 年代になり多くの研究者の注目を浴びるようになり活発な研究が行われるようになった。進化計算の大きな特徴は，解空間が非常に大きく，従来手法では高速計算機を用いても数十年かかるような問題や，問題の構造が分からず数学的に定式化できないため従来手法では解けない問題などに対して，近似解を比較的高速に得られるという点にある。この手法では，正しい解とともに，人間が思いつかないような解に行き着くこともあり，「創発的探索」とも呼ばれる。

進化計算は，2000 年代になって工学分野を中心に，実問題への適用が急速に進んでいる。身近な応用の一例をあげると新幹線のぞみ 700 系統の先端形状の最適化設計をあげることができる [1)]。この例では高速走行で発生する騒音と空気抵抗を小さくする形状設計を進化計算の手法によってなされ，大きな成功を収めた。

進化計算の実問題への応用の推進力の一つには，近年のコンピュータ性能の飛躍的な向上があげられる。高価で高速なコンピュータを用いなくても，安価な高性能 PC を利用することにより，従来困難であった実問題への応用が比較的容易に行えるようになった。従来数時間を要していた計算が数分で行えると，例えば配送計画の最適化問題に実時間で対応できる解を得ることに利用できる。

以下本稿では，一般的なユーザにおけるコンピュータ環境で進化計算の並列処理による高速化について，筆者が取り組んできた並列進化計算の手法 [2), 3), 4), 5)] について述べ，今後の研究課題について考察する。まずⅡ章では，進化計算の最新の動向について概観する。つぎに，Ⅲ章では，並列進化計算の各種モデルを概観する。Ⅳ章では，Java アプレットによるネットワーク環境型の並列進化計算モデルについて，また，Ⅴ章ではマルチコア計算機による並列進化計算モデルについて述べる。最後にⅥ章ではグラフィックスプロセッサ（Graphics Processing Unit; GPU）を用いた超並列進化計算の試みについて述べ，その検討課題について考察する。

Ⅱ．進化計算の概要

1．探索手法としての進化計算の特徴

今日，コンピュータは家電製品から自動車，事務処理，社会インフラのシステム制御など，あらゆる分野で用いられ，もはや使われていることが意識されないほどコンピュータはあらゆる分野に遍在し，我々の生活になくてはならないものとなった。

しかし，コンピュータは原理的な限界を持っている。それは，コンピュータはプログラムによって動作し，それ以外のことは基本的にできないということである。プログラムは人間によって作成される。コンピュータはプログラマが考えた以上のことはできない。コンピュータに思考，知性，適応といった，知的な能力を持たすことができないのか？コンピュータが発明された当初から，このような「人工知能」の研

阪南論集　社会科学編　Vol. 45 No. 3

究は，コンピュータ技術者の夢であった。しかし現実には，本当の意味での人工知能の実現は大変難しいということは明らかである。

さて，ここでコンピュータから離れて，生物進化を考えてみよう。約 40 億年前に原始生物が誕生したと考えられている。この過程で地球の大きな変動を何度も乗り越え，変動する環境に厳しい生存競争を通して適応することで生物は進化を遂げ，高等な生物へと進化した。当初は生物の変異は突然変異のみであったが，有性生物の出現により「交叉」という新たな変異の手段を獲得し，生物の多様化と進化が促進された。

生物進化における生物のこの優れた「適応機構」をコンピュータで模擬し，プログラムにより問題を直接解くのではなく，この適応過程を得た結果として解を見つけよう，このような発想で新たな「知的プログラミング」を構成しようというのが進化計算の源流の一つである「遺伝的アルゴリズム」（Genetic Algorithm，GA）の考え方である。ここで，生物に似せるものが問題の「解候補」である。すなわち，複数の解候補を生物の個体（Individual）にみたてて，その集団（Population）を「進化」させ，より良い解候補の集団に進化させる。進化計算の基本的な手順を単純 GA（Simple GA）と呼ばれるものを例に図 1 に示す。

単純 GA では，まず最初に複数の個体からなる初期集団 $P(t)$（ただし，$t=0$）を生成する。初期個体は一般にランダムに生成されるので，初期集団 $P(0)$ は低品質の解候補の集合である。つぎにそれらを評価（Evaluation）し，その中からよい個体を選択（Selection）する。選択された個体をペアリングし交叉（Crossover）と突然変異（Mutation）からなる遺伝的操作（Genetic Operator）を適用し次世代集団を生成する。これを設定した品質の解（個体）が得られるまで繰り返す。

進化計算は GA の他，本章 2 節で述べるように多くの手法があり，現在では多くの実際の問題に応用され，従来の手法では解くことが困難であった問題の解法に適用されている。ここで，進化計算の特徴をまとめると以下のようになる。

(1) 解の探索は集団で行う。すなわち，探索手法から分類すると，多点探索である。これは山登り法やシミュレートアニーリングと大きく異なる点である。これにより局所解にトラップされることが少なくなる。

(2) ブラックボックス探索ができる。生物は生存できるかどうかは環境への適応度である。これと同様進化計算では解候補の優劣の情報を用い，数学的な定式化ができない問題にも対処できる。

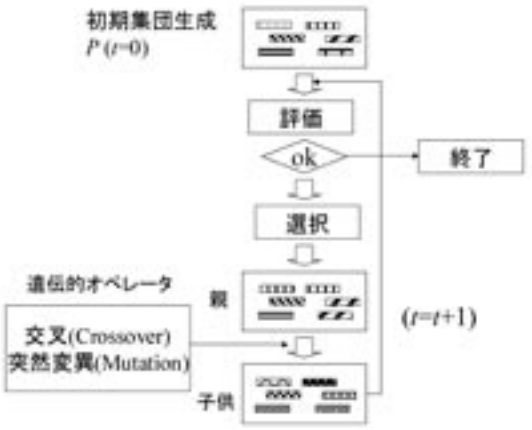

図 1　進化計算の手順（単純GAの場合）

(3) 上記(2)の特性は非常に大切な特徴であり，微分不可能な不連続な問題や非線形問題の解法にも有利となる。数学的に定式化できる問題でも，組合せ爆発により高速計算機を用いても実用的な時間で解けない解空間の大きな問題に対しても対応できる。

(4)「進化」の結果として解を得る。これにより人間が思いつかないような解を得るという「創発的探索」ができる。

進化計算の応用事例を表1に示す。このように，進化計算は主に工学分野での応用が多い。しかし，最近では金融分野や，社会経済分野でも応用が始まっている。

2．進化計算の研究の発展過程

進化計算は，本章2節で述べたように生物進化にヒントを得た探索から始まっているが統計的探索手法との融合など，その枠組みが広まっている。進化計算の最大の特徴は，集団で探索するという点にある。現在では，集団で探索する手法を総称して進化計算（Evolutionary Computation）と呼ばれている。以下では，進化計算の枠組みで捉えられる代表的な手法を簡単に述べる。

表1　進化計算の応用事例

設計･計画問題	プリント基板の最適な位置配置
	LSI のレイアウト設計
	ニューラルネットワークの合成
	通信網の動的予測ルーティング
	ジェットエンジンの設計
	エアコン制御の設計
	ハードウェア進化
	購入計画問題
	レンズ系の設計
制御問題	エレベータ群管理
	ガスパイプライン制御
	プロセス制御
	配電系統の損失最小化制御
スケジューリング問題組合せ最適化問題	ジョブショップスケジューリング
	各種工程スケジューリング
	製品投入スケジューリング
	運送業乗務員スケジューリング
	巡回セールスマン問題
学習･その他	ロボットの行動学習
	概念形成問題
	戦略獲得問題
	DNA 配列のシグナルパターンの抽出
	DNA の配座解析
	画像解析･復元
	タンパク質の構造推定

進化計算の源流は一般に，1960年代にRechenbergに提案され，その後Schwefelらにより発展させられた進化戦略（Evolutionary Strategy，ES），同じく1960年代のL. J. Fogelに始まる進化プログラミング（Evolutionary Programming，EP）および1950年代から研究が始まり1975年のHollandの研究[6]で基本的な枠組みができた遺伝的アルゴリズム（Genetic Algorithm，GA）の三つが挙げられる。これらは，いずれも生物進化をヒントとする探索，適応化手法である。しかし，これらは1980年代までお互いに遭遇することなく研究されてきた。

GAの研究グループにより1985年に米国において，国際会議第1回International Conference on Genetic Algorithms（ICGA）[7]が開催され，以降2年おきに開催されてきた（1999年からよりGenetic Programming Conferenceと統合して現在の毎年開催されるGECCOとなっている）。当初ICGAは国際会議とはいえ，ローカルな研究仲間の会議であったが，1989年ごろからESの研究者も加わりはじめ，1990年代にはGA，ESを含む進化計算の中心的国際会議となった。

一方，ヨーロッパにおいては，もう少し広い枠組みを狙いとした国際会議第1回Parallel Problem Solving from Nature（PPSN）がSchwefelらの提唱によりESの発祥国のドイツで1990年に開催され，GAの研究者も多く参加している。以降，2年おきに開催され，現在では進化計算に関するハイレベルな国際会議の一つとしての地位を確立している。1994年になって第1回IEEE International Conference on Evolutionary Computation（ICEC）が開催された。1999年から新たに，IEEEとSociety of EPとの共催の形でCongress on Evolutionary Computation（CEC）が毎年開催されている。

このように1990年代になって従来の異なっ

たコミュニティが互いに本格的に遭遇し，研究交流が国際会議を通して始まった。この交流は，例えばGAのコミュニティが重視している交叉がESに取り入れられるというように方法論的イミグレーションを伴う。

1990年代に入り，従来の生物進化をベースとする手法に加え，群れの集団行動をモデルとするアントコロニー最適化（Ant Colony Optimization，ACO），粒子群最適化（Particle Swarm Optimization，PSO）や進化計算に統計的手法を融合する分布推定型アルゴリズム（Estimation of Distribution Algorithm，EDA）[8), 9)] などの手法が進化計算の仲間として加わってきた。

各コミュニティはときには強い自己主張を行い，またときには排他的になりつつも，これらにより進化計算コミュニティの多様性が維持され，相互に影響しながら発展してきた。

以上，進化計算の発展過程を簡単に述べたが，この過程を大局的に見ると研究の大きなブレイクスルーはほぼ10年を単位として進んでいる。多くの技術の発展，例えばインターネット技術の進展を見ても，一見急速な技術変化をしているように見えるが，大局的に見ると研究の節目はほぼ10年程度要している。興味ある一致である。

Ⅲ．進化計算の並列化方式の分類

一般に，コンピュータによる計算の並列化に当たっては，どのような単位でプログラムを並列化するかが重要となる。どのようなプログラムでも，逐次法でしか実行できない部分と並列に実行できる部分からなる。逐次法で処理されるプログラムの作業量の割合をrとし，残りの$(1-r)$がp個のプロセッサで並列実行されるとすると，並列化による高速化は，高々，

$$speedup = \frac{1}{r + \frac{1-r}{p}} \qquad (1)$$

である。これはアムダールの法則（Amdahl's law）として知られている式である。式（1）に従うと，例えばr=0.5の場合，プロセッサ数を無限にしても，2倍の高速化しか得られないことになる。このことから分かるように，並列化に当たっては，逐次法でしか実行できない部分を極力0に近づけなければ，プロセッサ数を増やしても高速化は得られない。並列化による高速処理を実現するには，このことを念頭にプログラム構造の設計が重要となる。

進化計算はⅡ章でも見たように，もともと複数の個体に関する処理手順であり並列化に適しているといわれている。一般に進化計算の並列化方式には，以下のような4つのモデルが代表的である[10)]。

(1) 単一集団によるマスタ／スレーブモデル：このモデルでは，マスタプロセッサにより集団が管理され，スレーブプロセッサにより各個体の評価が行われる。一般に，進化計算では個体の評価に要する計算量が大きく，この部分の並列化の効果が大きい。これはいわば効率的な負荷分散方式である。

(2) 分散進化計算モデル：このモデルでは，進化の単位である集団を複数個配置する。それぞれの集団の処理が各プロセッサに割り当てられる。各集団は各プロセッサで独立に進化するが，一定の世代間隔で各集団の個体を集団間で交換する。このモデルは，島モデルとも呼ばれ，海洋の島々での進化過程にヒントを得たものである。分散進化計算モデルでは，各集団は異なった進化をすることと定期的な集団間での個体交換により，全体として多様性を持った集団の進化が期待でき，この結果として良好な解が高速に得られることが期待できる。

(3) セル型進化計算モデル：小さな集団を2次元あるいは3次元上に配置し，隣り合う集団間にまたがって遺伝的オペレータを適用する。これは大陸における生物集団間の交流による進化過程にヒントを得たモデルである。大規模並列計算機に向いた方式である。

(4) ハイブリッドモデル：(1)～(3)の基本モデルを組み合わせたモデルであり，多くの変形方式が考えられる。

以上のモデルに加えて，本研究では，以下の並列化モデルを検討した。

(5) 個体レベルの並列化モデル：この方式では，各個体の一連の進化オペレーションを独立したプロセッサで行わせる。

以上の５つの並列化モデルには，使用する計算機環境や用いる進化計算の方式により各種の実現方式があり，与えられた計算機環境で最適な並列化方式を見つけることが並列進化計算の研究課題である。Ⅳ章で述べるネットワーク型並列計算は，ネットワーク環境上のプロセッサを用いたマスタ／スレーブモデルに関するものである。Ⅴ章で述べるマルチコアプロセッサによる並列進化計算は，マルチコアプロセッサを用いて個体レベルの並列化モデルを実現したものである。また，Ⅵ章の GPU を用いた超並列進化計算は，個体レベルの並列化と分散進化計算モデルを組み合わせたものである。

最後に並列進化計算の研究の目的を再確認して本章を締めくくろう。並列進化計算の目的は大きく次の二つにまとめることができる。第一は，決められた品質の解を高速に得ること，第二は，決められた時間内で，よりよい品質の解を得ることである。本論文では前者，すなわち，決められた品質の解を高速に得ることを目的とし，その並列化方式について述べている。

Ⅳ．Java アプレットによるネットワーク型並列進化計算

本章では，筆者が開発した高性能 ACO アルゴリズムであるカニングアントシステム（cunning Ant System，*c*AS）をネットワーク環境で並列化した並列 *c*AS（parallel *c*AS, *p*-*c*AS）を２次割当て問題（Quadratic Assignment Problem，QAP）の解法に適用した方式ならびに結果について述べる。

１．カニングアントシステム（*c*AS）と２次割当て問題（QAP）

ACO は，実際のアリの採餌行動における経路生成過程にヒントを得た探索手法であり，巡回セールスマン問題（Traveling Salesman Problem，TSP）など多くの組合せ最適化問題に適用され，有効な結果が得られている。アリはフェロモンを介したコミュニケーションを行いながら群れで行動し，ある種の秩序を形成する。ACO では，この秩序形成過程を探索に用いる。

ACO の基本モデルは，Dorigo らによる Ant System（AS）[11] と呼ばれるアルゴリズムに代表される。その後，多くの改良型 ACO アルゴリズムが提案されている [12]。

通常の ACO アルゴリズムでは，各エージェント（アリ）により生成される解はフェロモン濃度 $\tau_{ij}(t)$ に基づいて確率的に生成される。ただし，$\tau_{ij}(t)$ は，解の順序表現におけるノード i，j 間のフェロモン濃度である。著者が提案した *c*AS[13), 14)] では，カニングアント（cunning ant, *c-ant*）と呼ぶエージェントを導入している。*c-ant* は，従来のエージェントと異なり，新しい解候補を生成の際に，過去の探索において存在する解の一部分（部分解）を借用し，解の残りの部分の生成にのみ従来と同様フェロモン濃度を利用する（図２参照）。このようなエージェントを抜け目のないアリという意味でカニングアントと呼んだ。狡をするエージェントという意味も言外に込められている。

*c*AS は経路を一部借用するという方法により，探索過程においてポジティブフィードバックによる集中化を緩和し，初期収束を抑止する。この効果により *c*AS は優れた探索性能を示している。

さて，ここで解く問題は２次割当て問題（Quadratic Assignment Problem，QAP）である。QAP は，L 個からなる部門を L 箇所のロケーションに式（2）で定義される値が最小になるような順列 ϕ を求める問題である。これは，企業における事業所や工場の立地の最適配置問

阪南論集　社会科学編　Vol. 45 No. 3

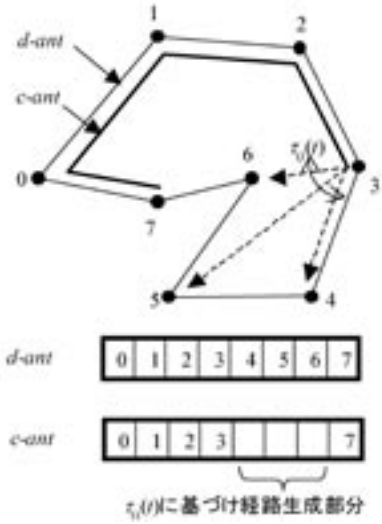

図２　カニングアントの解生成法

題やビルの部門の最適配置問題などを一般化したもので，組合せ最適化問題の中では，最も困難な問題に属する。

$$f(\phi)=\sum_{r=1}^{L}\sum_{s=1}^{L}b_{rs}a_{\phi(r)\phi(s)} \qquad (2)$$

式（2）において，$A=(a_{pq})$ および $B=(b_{rs})$ はそれぞれ $L\times L$ の距離マトリックスとフローマトリックスであり，ϕ は $\{1, 2, \cdots, L\}$ の順列である。マトリックス A と B は，それぞれ，場所 p，q 間の距離，部門 r，s 間の流量を表している。QAP は評価式が距離と流量の積となっており，両情報が相互に絡み合うため，TSP に比較して格段に困難な問題である。

２．ネットワーク環境における *c*AS による QAP の解法

１）*c*AS の並列化のためのネットワーク基本モデル

本研究における *c*AS の実装には Java を用いている。Java には RMI や Jini などネットワーク環境における分散処理のためのパッケージが準備されているが，本研究では *c*AS の並列化の基本アーキテクチャとして，クライアント／サーバモデルに基づくアプレットベースの基本モデルを構築した。この構造を図３に示す。

クライアントで実行されるプログラムはアプレットのクラスとしてサーバから転送される。すなわち，クライアントのプログラムは，ブラウザから Java アプレットとして実行される。クライアントとサーバ間の通信は，TCP ソケットプログラミングを用い，両者の情報交換は Serializable インタフェースを実装し

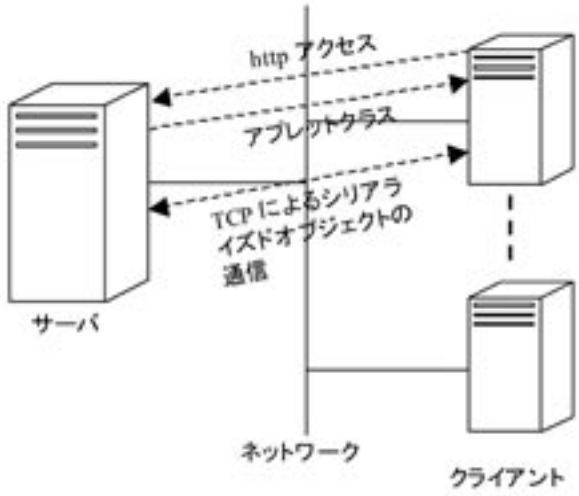

図３　*c*AS の並列化のためのネットワーク基本モデル

たオブジェクトを交換して行う。具体的には，ObjectInputStream，ObjectOutputStream オブジェクトの交換として通信する（このオブジェクトの詳細については Java 2 API[15] 参照）。

以上の方式により，ネットワークに接続され，Java アプレットの実行環境を有する任意のクライアントがサーバに http プロトコールによりアクセスすることで，簡単に並列計算環境を構築することができる。

なお，クライアント側で QAP の距離マトリックスやフローマトリックスなどのデータファイルの参照が必要となる。しかし，アプレットではファイル参照ができないので，これらのファイルは Java の一つのクラスに変換しておくことを前提としている。これにより各クライアントで必要となるファイルのデータは http プロトコールによりクライアントからアクセスできる。なお，サーバには Web サーバとしての機能を必要とする。

2）*p-c*AS の実装

表 2 は，*c*AS に局所探索としてタブーサーチを結合した場合の処理時間の分布を各サイズのテスト問題に対して示したものである。実行環境は 2 個の Opteron 280（dual core，2.4GHz，Socket940）プロセッサからなる WindowsXP マシン（主メモリ：2 GB）である。この表より，探索の総時間のほぼ 99% 以上が局所探索に費やされていることが分かる。そこで，今回の *p-c*AS では，局所探索を並列化して高速化を図るという方式を，図 3 で示したネットワーク基本モデルを用いて実装した。

*p-c*AS の実行環境は上述の Opteron マシンを 2 台用い（以下，それぞれ，マシン A，マシン B と呼ぶ），両マシンを 1000BASE-T スイッチングハブで接続する。それぞれのマシンは 4 個のプロセッサを有する。マシン A をサーバとして用い，クライアント機能はマシン B に割り当てる。マシン A の Web サーバ用ソフトには，Apache[16] を用いた。

図 4 は，上記の環境での局所探索の並列化の方式である。用いるアント数 m（進化計算では一般に集団サイズ）は m=5 である。そこで，4 個の *c-ant* の局所探索をクライアントで行うこととし，このためマシン B には Java アプレットが実行できるブラウザを，4 つの独立したプロセスとして起動する。厳密には各ブラウザプロセスは，OS により 4 つのプロセッサ間で動的に切り替わって実行されるが，これらのプロセスは 4 つのプロセッサに論理的に割り当てられると考えてよい。また，残り 1 つの *c-ant* の局所探索はサーバプロセッサに割り当てる。なお，図 4 では，クライアントプロセッサ数を N_c として一般化して記述している。上記の説明は，同図において，N_c=4，m=5 とした場合に相当し，局所探索を適用するために各プロ

表２　局所探索を用いた*c*ASにおけるQAPでの計算時間の分布
（単位はミリ秒で，25回の実験の平均値を示している）

QAP	サンプリング	ローカルサーチ	τ_{ij} の更新	その他	合計
tia40b	3.8	6833.8	1.3	12.3	6851.2
（%）	0.1%	99.7%	0.0%	0.2%	100.0%
tia50b	4.9	13465.8	2.3	13.3	13486.2
（%）	0.0%	99.8%	0.0%	0.1%	100.0%
tia60b	7	23448.7	3	15.7	23474.4
（%）	0.0%	99.9%	0.0%	0.1%	100.0%
tia80b	10.6	56688.7	5.9	19.9	56725
（%）	0.0%	99.9%	0.0%	0.0%	100.0%
tia100b	14.8	114411.7	9	21.4	114456.9
（%）	0.0%	100.0%	0.0%	0.0%	100.0%
tai150b	29.9	422079.8	19.9	46.1	422175.6
（%）	0.0%	100.0%	0.0%	0.0%	100.0%

阪南論集　社会科学編　Vol. 45 No. 3

セッサに割り当てられるエージェントの数 e は 1 である。

3．結果と考察

図 5 に *p-c*AS の結果と考察結果をまとめる。データは 25 回の実験の平均である。ここで，*speedup* は，（並列化しない場合の *c*AS の実行時間）/*p-c*AS の実行時間）を示す。クライアントとサーバ間の通信オーバーヘッド（$T_{comm.}$）が無ければ *speedup* の値は 5 に近い値となる。しかし同図に示しているように，通信オーバーヘッド（$T_{comm.}$）が存在する。この通信オーバーヘッドのために tai40b および tai50b では *speedup* 値は 1 より小さく，また tai60b でほぼ 1 となっている。一方，より大きな問題である tai80b，tai100b および tai150b では，*speedup*

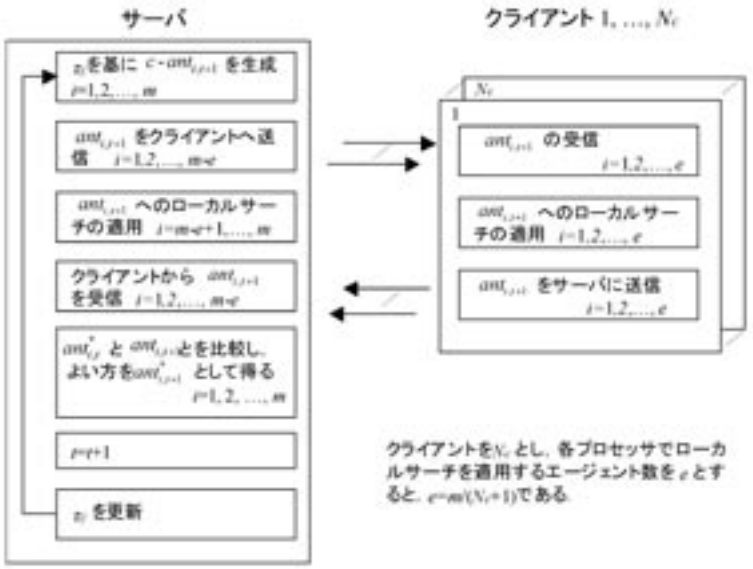

図４　並列化*c*AS（*p-c*AS）の概要

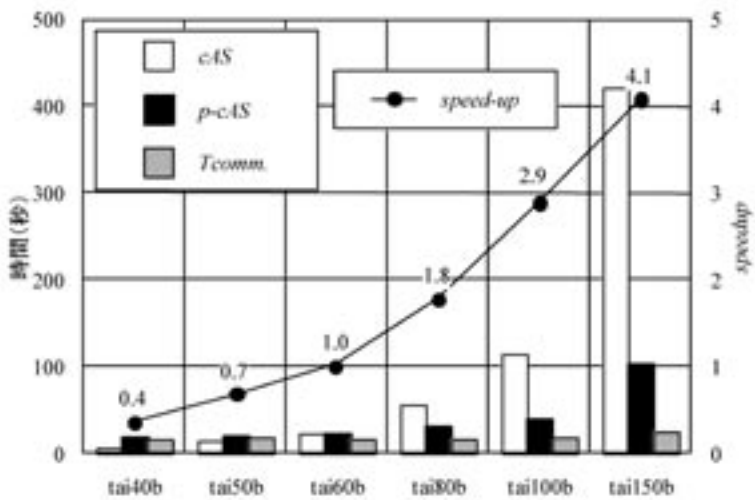

図５　*p-c*ASによる高速化の結果

は，それぞれ，1.8，2.9 および 4.1 となっている。これは，大きな問題では全体の時間に占める T_{comm} の割合が小さくなるためであり，大きな問題での並列化による高速化への効果が大きいと言える。なお，この並列化方式では，より複雑な局所探索を用いる場合には更なる高速化の効果が得られる。

最後に，本並列化方式の特徴をまとめると，Java VM と Web ブラウザを有すれば，ネットワークに接続されている任意のコンピュータが，並列処理として動作できるという点である。

Ⅴ．マルチコアプロセッサによる並列進化計算

近年，マルチコア型の高性能 CPU が開発され，一般の安価な PC にも利用できるようになってきた。このタイプの計算機では，2，4，6 といった数の演算装置を有し，マルチスレッドプログラミングにより，並列進化計算が比較的容易に実現できる。マルチコア型 PC による並列処理では，4 章のネット型の並列計算で見たような通信オーバーヘッドが少ないため，効率的な並列化が実現できることは容易に考えられる。また並列化の粒度も小さくとることができる。しかしここでも，進化計算の並列化に当たっては，その並列化モデルを十分考慮しなければ高性能化は得られない。

本章では，順序表現向きの分布推定アルゴリズム（EDA）の一つとして筆者が先に提案した Edge Histogram Based Sampling Algorithm（EHBSA）[17] の改良型モデル（enhanced EHBSA，以下 eEHBSA）のマルチコア PC による並列化を，TSP をテスト問題として行った結果について述べる [4]。

1．改良型 EHBSA（*e*EHBSA）

1）エッジの分布

*e*EHBSA では，EHBSA と同様，分布推定モデルとして，集団全体に含まれる個体のエッジの分布をマトリックス形式で表現した EHM（Edge Histogram Matrix）を用いる。図 6 は，問題サイズ $L = 5$ の TSP に対して，集団サイズ $N = 5$ とした場合の集団 $P(t) = \{s^t_1, s^t_2, \cdots s^t_L\}$ の EHM の例である。EHM の各エントリー e_{ij} の整数部分は，集団におけるエッジ $i \rightarrow j$ の総数である。小数点以下の値（同図では，0.1 としている）は，個体生成時における外乱を与えるバイアスであり，アルゴリズムのパラメータの一つである。ここでは，対称 TSP の場合のようにエッジ $i \rightarrow j$ とエッジ $j \leftarrow i$ とが同等である場合を示しているが，スケジューリング問題のようにエッジ $i \rightarrow j$ とエッジ $j \leftarrow i$ とが異なる場合にも同様に EHM は非対称マトリックスとして表現できる。

2）個体生成法

図 7 に一つの新個体を生成する方法を示す。新個体は以下のようにして生成される。

（ⅰ）現在の集団から「テンプレート」に用いる個体を一つ選ぶ。

（ⅱ）EHM を用いて生成するノード数 l_s を式 (3)[4] で示す確率密度関数 $f(l_s)$ より決定し，$l_p = L - l_s$ 個のノードをテンプレートから

$$s^t_1 = (1, 2, 3, 4, 5)$$
$$s^t_2 = (2, 4, 5, 3, 1)$$
$$s^t_3 = (4, 5, 3, 2, 1)$$
$$s^t_4 = (5, 1, 4, 2, 3)$$
$$s^t_5 = (3, 2, 4, 5, 1)$$

(a) $P(t)$

$$\begin{pmatrix} 0 & 3.1 & 2.1 & 2.1 & 3.1 \\ 3.1 & 0 & 4.1 & 3.1 & 0.1 \\ 2.1 & 4.1 & 0 & 1.1 & 3.1 \\ 2.1 & 3.1 & 1.1 & 0 & 4.1 \\ 3.1 & 0.1 & 3.1 & 4.1 & 0 \end{pmatrix}$$

(b) EHM ^{t}

図６　EHMの一例

新個体のメモリ領域にコピーする（Partial Solution Segment，PSS）。コピーの開始位置 p_{top} はランダムに決定する。

（iii）残りの l_s 個のノードは，EHM をもとに式（4）に基づいてサンプリングする（Sampling Based Segment，SBS）。

先の EHBSA の研究[17]では，l_s の決定に n カットポイント法を用いていたが，この方法では，l_s の平均値 $E(l_s)$ が L/n（$n = 2, 3, \cdots$）に制限されるという問題があり，改良型 EHBSA では，$E(l_s)$ が $L \times \gamma$（$\gamma \in [0, 1]$）となるパラメータ γ を導入した。

$$f(l_s)=\begin{cases}\dfrac{1-\gamma}{L\gamma}\left(1-\dfrac{l_s}{L}\right)^{\frac{1-2\gamma}{\gamma}} & \text{for } \gamma\in(0,0.5]\\ \dfrac{\gamma}{L(1-\gamma)}\left(\dfrac{l_s}{L}\right)^{\frac{2\gamma-1}{1-\gamma}} & \text{for } \gamma\in(0.5,1]\end{cases} \quad (3)$$

（iii）のサンプリングでは，現在位置のノードが i であるとき，i の次の位置のノード j は，次式の確率 p_{ij}

$$p_{ij}=\begin{cases}\dfrac{e_{ij}}{\sum_{s\in F(i)}e_{is}} & \text{If } j\in F(i)\\ 0 & \text{otherwise}\end{cases} \quad (4)$$

で決定される。ただし，$F(i)$ は，現在のノード i からつぎに選べるノードの集合である。この式に基づくサンプリングの計算量は一般に $O(L^2)$ になり，問題サイズが大きくなるとサンプリングに要する時間が問題となる。このような場合，候補リスト[18]を使うと計算量を $O(L)$ に近づけることができる。

EHBSA では，1 世代に一個体を生成するモデルを用いたが，本研究では，図 8 に示すように，1 世代で N 個体を生成する方式とし，効率的な世代交代モデルに改良している。同図において，新しい個体 I'_i は，集団 $P(t)$ の個体 I_i をテンプレートとして用いる（$i = 1, 2, \cdots, N$）。各 i のペア（I_i, I'_i）（$i = 1, 2, \cdots, N$）を比較し，良い個体を次の世代 $P(t + 1)$ のメンバーとする。I_i と I'_i とを比較するこの方式は，Mahfoud の deterministic crowding 法[19]のように多様性維持に効果がある。また，この方式では，図 7 で示した γ も重要な設計パラメータとなる。

3）*e*EHBSA のシングルスレッド性能

*e*EHBSA の並列化されていない場合の基本性能を見るために，TSP を用いて評価した結果を述べる。ここでは，（1）小規模問題：berlin52，pr76，（2）中規模問題：att532，rat783，（3）大規模問題：fl3795，rl5934 の 3 つの規模の問題を取り上げる。実験条件は以下の通りである。（1）の小規模問題に対しては，集団サイズを $2L$，最大解生成数を $2L \times 10000$ とし，局所探索は適用しない。（2）の中規模問

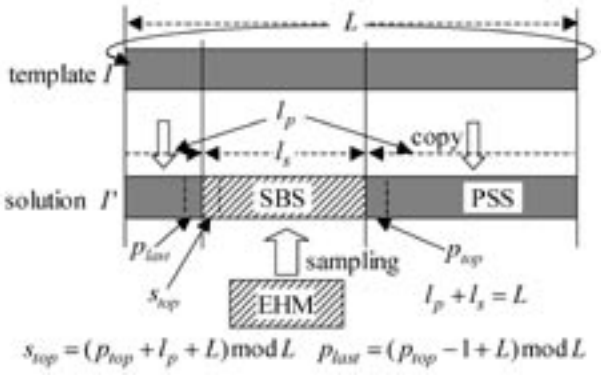

図 7　個体の生成法

題に対しては，集団サイズを L=15，最大解生成数を $L\times1000$ とし，局所探索には 3OPT を適用する。(3) の大規模問題に対しては，集団サイズを 4 とし，最大解生成数を $L\times10$ とし，局所探索には TSP における最強の局所探索とされている Lin-Kernighan（LK）ヒューリスティックを適用する。

プログラミング言語には Java を用い，マシンには Core i7 965 プロセッサを用いる。LK コードには Concorde[20] を用い，JNI を用いて *e*EHBSA コードと結合した。それぞれの問題に対して γ を 0.1 から 1.0 まで 0.1 刻みで実験を行った。各実験における試行回数は 20 回とした。各規模の問題に対して，#OPT（20 回の実験中で最適解を発見した回数）および T_{avg}（最適解を発見した実験において，最適解を発見するのに要した時間の平均）による結果をそれぞれ，図 9 に示す。

これらの結果から *e*EHBSA では，γ の適切な設定が重要であることが分かる。また，いずれの規模の問題においても，$0.2 \leqq \gamma \leqq 0.4$ において #OPT=20 であり，また，T_{avg} も小さい値となっていることが分かる。

2．マルチスレッドプログラミングの実現法

*e*EHBSA の並列化方式として，同期マルチスレッドモード（synchronous multi-thread mode，SMTM）と非同期マルチスレッドモード（asynchronous multi-thread mode，AMTM）の 2 つの並列化スレッドプログラミングを考える。使用言語は Java である。使用プロセッサは，5.1.3 項で述べた Core i7 965 である。このプロセッサは 4 つのコアを内蔵している。BIOS においてハイパースレッディング機能を有効に設定すると OS から見かけ上 8 個のコアが存在するが，ここではその機能は用いないとした。したがって，本研究では，4 つのスレッドを生成して *e*EHBSA の並列実行を評価する。*e*EHBSA の繰り返し過程は以下となる。

(1) $t \leftarrow 0$. 個体 I_i（i = 1，2，…，N）からなる初期集団 $P(t)$ を生成。
(2) $P(t)$ の個体を評価（局所探索を適用する場合は適用後評価し，I_i も変更）。
(3) $P(t)$ から EHM を作成。
(4) 新個体 I'_i（i = 1，2，…，N）を，EHM および I_i の部分解部から生成。
(5) 各 i（i = 1，2，…，N）に対して I_i と I'_i とを比較し，I'_i の方がよければそれを I_i と置き換え，集団 $P(t)$ を更新する。
(6) $t \leftarrow t + 1$。
(7) 終了条件を満たせば終了，満たさなければ (3) へ。

各個体の一つの処理が一つのスレッドの処理となる。スレッドは 4 つとしているので，集団サイズ分の処理をこれらの 4 つのスレッドが処理を繰り返す。SMTM では，各ステップの処

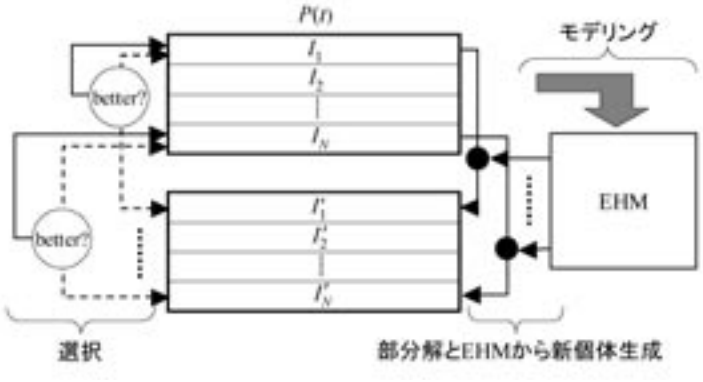

図８　世代交代モデル

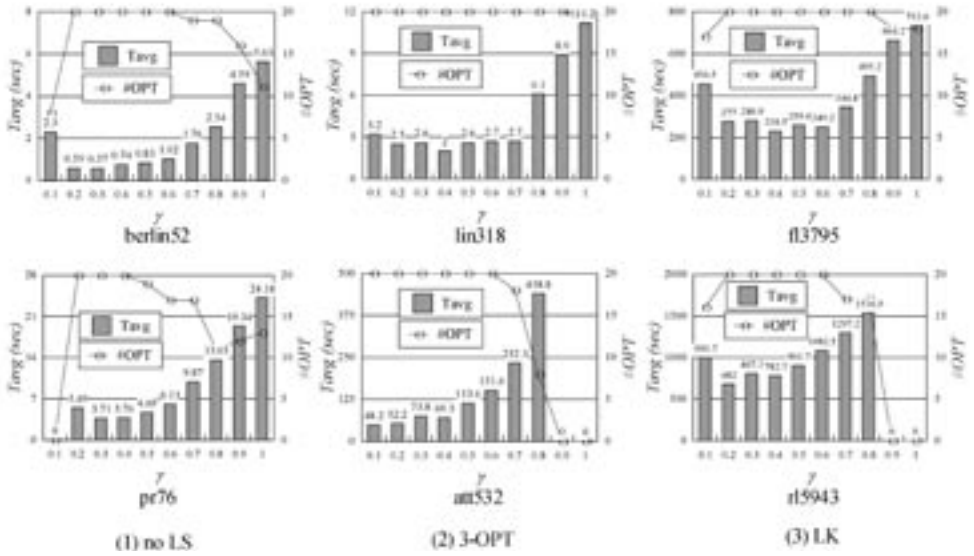

図９　*e*EHBSAのシングルスレッド性能

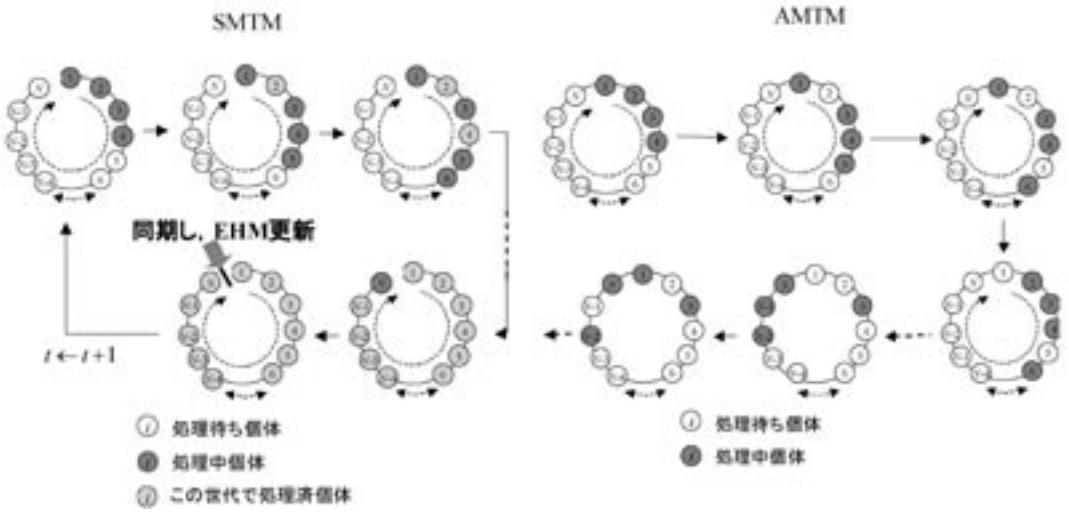

図10　AMTMおよびAMTMにおける4つのスレッドによる個体の処理過程の例

理を同期して行う。それに対して，AMTM では上記（3）まで同期モードで動作し，それ以降は独立に未処理の個体を並列実行する。このとき，EHM の更新では，全ての個体が更新されてからではなく，I_i と I'_i の入れ替えとなったときには，現在の EHM から I_i の属するエッジを取り出し（それらのエッジの数を –1 する），I'_i に属するエッジを加える（それらのエッジの数を +1 する）。この仕組みにより，AMTM では処理の待ち合わせに伴う遅延は発生しない。ただし，処理が非同期に行われるので，各個体に対して各ステップの処理は完全な世代単位には行われなくなる。図 10 に 4 つのスレッドによる各個体の処理過程をそれぞれ SMTM および AMTM に対して示す。

3．結果と考察

各種サイズの TSP を用いて，2 つの並列化方式の結果を表 3 に示す。実験回数は 20 とした。この実験では 20 回の実験で全て最適解を発見している（#OPT=20）。したがって，T_{avg} は，最適解を見つけるまでに要した時間（秒）の 20 回の実験の平均である。*speedup* は，並列処理を行わない場合の T_{avg} をマルチスレッド方式による場合の T_{avg} で割ったものである。同表には，各 T_{avg} の 95% 信頼区間ならびに SMTM と AMTM の *speedup* の比較の *p*-値も示した。

SMTM の結果を見ると，局所探索を用いない小規模問題と局所探索を用いる中，大規模問題で大きな違いがあることが分かる。中，大規模問題では，速度比が 3.1 ～ 3.7 であるのに対して，局所探索を用いない小規模問題では速度比が 1.5 ～ 2.5 と高速化の度合いが小さくなっている。一方，AMTM の結果を見ると，中，大規模問題では SMTM の結果と大きな違いがなく，同程度の高速化の結果が得られている。大きな違いは，小規模問題の場合である。この場合にも，SMTM の場合と異なり AMTM では 3.4 ～ 4.4 と高速化の結果が得られている。局所探索を適用しない場合にはサンプリングに要する時間の割合が大きく（図 11 参照），また，*e*EHBSA ではこの時間は確率的に変動する。したがって，SMTM ではスレッド間の待ち時間が長くなる。AMTM ではこのような待ち時間はない。

Ⅵ．GPU を用いた超並列進化計算の試みと課題

ここ 2，3 年前より，いわゆるグラフィックボードとして一般の PC に用いられている画像処理用演算装置（Graphics Processing Unit, GPU）を用いて流体力学，医療用画像分析，統計データ処理などの科学技術計算の並列計算を実現する並列計算（GPU Computation，以下

表３　マルチコア計算機による並列進化計算の結果

テスト問題	局所探索	シングル *e*EHBSA		並列 *e*EHBSA						*p*-値
				SMTM			AMTM			
		T_{avg} (sec)	*SE*	T_{avg} (sec)	*SE*	*speedup*	T_{avg} (sec)	*SE*	*speedup*	
		信頼区間		信頼区間			信頼区間			
oliver30	no	0.082	0.004	0.05	0.08	1.5	0.02	0.04	3.4	5.93E-07
		[0.074, 0.091]		[0.045, 0.063]			[0.020, 0.004]			
gr48		0.6	0.04	0.29	0.01	2.1	0.16	0.01	3.8	1.31E-09
		[0.53, 0.67]		[0.26, 0.31]			[0.15, 0.17]			
berlin52		0.57	0.03	0.29	0.02	2	0.13	0	4.3	1.58E-07
		[0.51, 0.64]		[0.25, 0.33]			[0.12, 0.14]			
pr76		3.71	0.14	1.46	0.07	2.5	0.86	0.11	4.3	8.01E-04
		[3.40, 4.01]		[1.31, 1.61]			[0.74, 1.20]			
lin318	3-OPT	2.57	0.64	0.84	0.06	3.1	0.76	0.04	3.4	0.31
		[1.23, 3.91]		[0.70, 0.98]			[0.68, 0.84]			
pcb442		12.49	2.21	3.12	0.28	4	2.92	0.26	4.3	0.606
		[7.85, 17.12]		[2.54, 3.70]			[2.38, 3.47]			
att532		73.84	12.29	21.84	3	3.4	19.41	2.78	3.8	0.549
		[48.11, 99.56]		[15.57, 28.11]			[13.81, 25.02]			
rat783		139.48	7.57	38.77	2.72	3.6	38.37	2.75	3.6	0.918
		[123.63, 155.32]		[33.07, 44.47]			[32.62, 44.12]			
fl3795	LK	280.88	34.7	105.55	19.08	2.7	85.51	12.28	3.3	0.384
		[208.24]		[65.60, 145.50]			[59.81, 111.21]			
rl5934		807.27	76.04	252.44	26.51	3.2	209.62	16.91	3.9	0.183
		[648.12, 966.43]		[196.96, 307.93]			[174.22, 245.01]			

SE：標準誤差

阪南論集　社会科学編　Vol. 45 No. 3

GPU 計算）の研究が注目され，CPU に対して数 10 ～ 100 倍の高速化が図られたという有望な結果が報告されている [21), 22)]。GPU 計算による進化計算の並列化の研究も今後大きく発展するものと思われる。

2009 年 7 月の ACM 主催の進化計算の国際会議 GECCO-2009 では，進化計算への GPU 計算の適用に関するワークショップ 2009 Computational Intelligence on Consumer Games and Graphics Hardware（CIGPU-2009）が開催され，筆者らの研究も含めて 8 件の研究が発表された。2010 年には，IEEE 主催の進化計算の国際会議 CEC において，CIGPU-2010 が開催される。

本章では，2 次割当て問題（quadratic assignment problem，QAP）の解法に GPU 計算を用いて並列進化計算を実行した筆者らが試みた研究結果 [5)] について述べ，今後の研究課題について考察を行う。

1．GPU 計算による超多スレッドプログラミングの一実現法

本研究で用いた GPU は，NVIDIA GeForce GTX285 である。図 12 に本 GPU のアーキテクチャを示す。Thread Processor（TP）演算装置の単位であり，8 個の TP を単位として 1 つの Multi-processor（MP）を構成している。MP 内の TP は，レジスタ並みの高速アクセスが可能な Shared Memory（SM）を介してデータを共有することができる。SM のサイズは 16KB である。MP の数は 30 である。異なる MP に属する TP 間のデータ共有は，VRAM 経由となり速度が遅くなる。プログラミング環境としては，CUDA（compute unified device

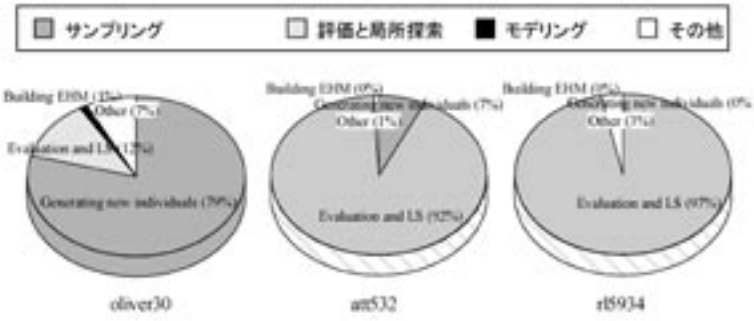

図11　eEHBSAにおける処理時間の分布

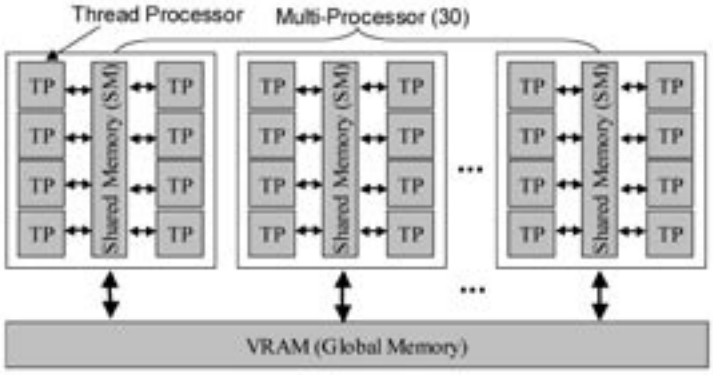

図12　NVIDIA GeForce GTX285のアーキテクチャ

architecture）と呼ぶC言語を拡張した統合開発環境があり，コンパイラやライブラリなどから構成されている[21]。Windows環境では，Microsoft Visual Studioに組み込んで使うことができ，本研究では，この方法を用いた。

CUDAでは，合計8 × 30 = 240個のTPを動作待ち時間なく有効に動作させる（メモリアクセス遅延を隠す）ために数千という超多スレッドプログラミングを前提としている。このため，*grid*，*block*，*thread* という概念を導入し，物理的な構成を直接的に意識せずに超多スレッドプログラミングができる環境を提供している。スレッドの切り替えはハードウェアにより行われ，切り替え時間のオーバーヘッドは発生しない。この考え方はPentium IVで導入されたハイパースレッドの機構と類似している。

GPU計算を進化計算の並列化に応用する際の大きなネックは，SMのサイズにある。個体の集団をVRAMにおくと，各TPからのメモリアクセスの遅延が大きいため，高速化に不利となる。このため，SM（16KB）の活用が重要となる。本研究では，一つのサブ集団$P(t)$をSMに置くという分散型GAモデルを考案し，高速化を図ることを考えた。QAPは大きな問題でも問題サイズが高々150程度である。したがって，順列表現である染色体の遺伝子には *unsigned char* 型を使用することができ，個体表現に要するSMの消費量を節約できる。

各サブ集団の世代交代モデルでは，図8で述べたのと同様，一つの個体の処理がCUDA環境で1つのスレッドとなるよう工夫している。サブ集団サイズをNとすると，SMの消費量は，$2L \times N$バイトとなる。今回は$N = 128$に固定し，最大問題サイズを$L = 56$とした。交叉にはPMXオペレータ，突然変異には2つの任意の遺伝子座の値を入れ替えるSWAPオペレータを使用した。局所探索は適用していない。各サブ集団の世代交代モデルを図13に示す。

各MPにおけるサブ集団の個体は，図14に示すように500世代毎にVRAMを介してシャッフルする。この意味で，GPU内の進化モデルは，分散GAにおける島モデルの一形態といえる。個体数の総数は，$128 \times 30 \times k$（$k = 1, 2, \cdots$）であり，これがCUDAプログラミング環境における総スレッド数になる。

2．結果と考察

QAPの各テスト問題に対して，GPU計算によるスレッド並列化の結果と，Core i7 965プロセッサにおいてシングルスレッドで実行した結果とを，最適解を発見するのに要した時間（T_{avg}）で比較したものを表4に示す。ここで，CPUにおけるGAのモデルとして，論理的にGPU計算の場合と同じとした場合（GA-1）と島間の個体の交換をチューニングしたモデル

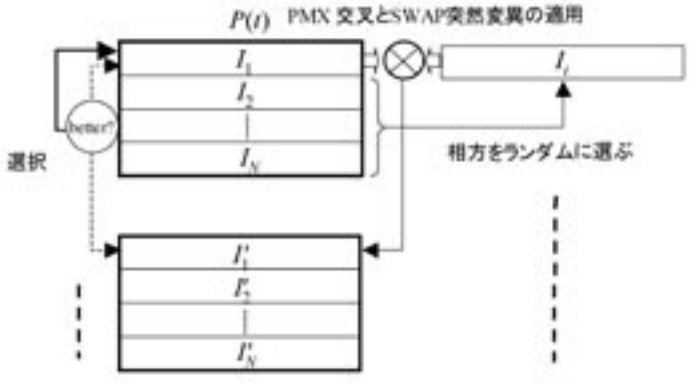

図13　GPU計算における各サブ集団の世代交代モデル

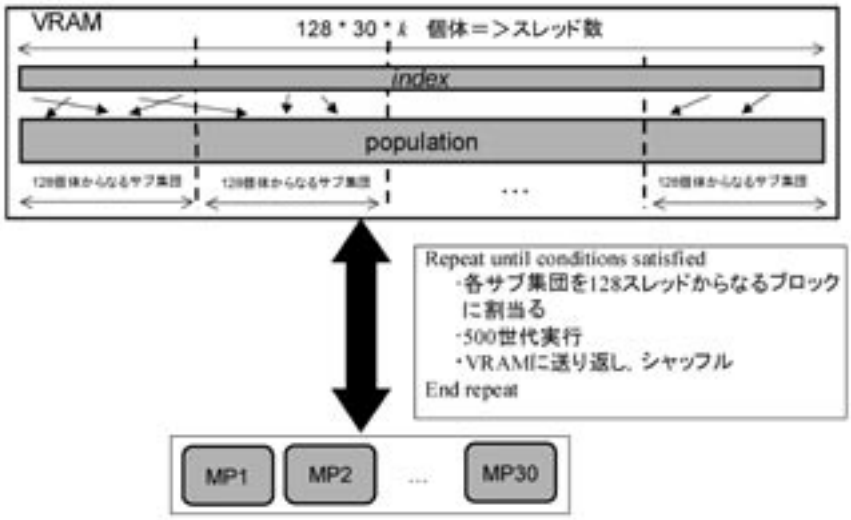

図14　GPU計算における進化計算の構成

（GA-2）の結果を示した。当然GA-2の結果の方が，GA-1の結果よりも優れている。GPU計算のGA-2に対する *speedup* の値として3～12という結果が得られた。

画像処理や流体力学などへの応用では，CPUの数10倍～100倍以上の高速化が期待できる[10]。今回の結果はこれらの分野と比較すると十分とはいえないが，今後計算モデルの改良により，さらなる速度の向上は十分期待できる。以下では，進化計算においてGPU計算を用いる際に課題となる点について考察し，今後への研究課題を述べたい。

進化計算にGPUを用いる際に課題となる事項としては以下をあげることができる。

- 進化計算では，処理が確率的に行われ，これにより多くの分岐命令が現れる。GPUではスレッドはブロックごとにMPに割り当てて実行されるが，各ブロック内のスレッドはワープに分割され（GTX285の場合，32スレッド単位）てSIMD風に処理が行われるので，スレッド間でTrueとFalseの処理がシリアルに行われ，このときMP内のTPにアイドル状態が発生する場合がある。また，分岐がなく同じ処理が行われる場合でも，各スレッド間での処理時間が異なると，全体の処理時間は長い方に合わされ，TPにアイドル状態が発生する。したがって，このような問題を少なくする計算モデルを工夫する必要が

表4　GPU計算による並列進化計算の結果

QAPテスト問題	GPU計算				シングルスレッドによるCPU計算								*speedup*	
					GA-1				GA-2					
	総個体数	*#OPT*	T_{avg} *(sec)*	*SE*	総個体数	*#OPT*	T_{avg} *(sec)*	*SE*	総個体数	*#OPT*	T_{avg} *(sec)*	*SE*	GA-1	GA-2
tai20b	128×30×1	10	0.064	0.001	128×30×1	10	0.428	0.012	128×30×1	10	0.422	0.013	6.6	6.7
tai25b	128×30×1	10	0.169	0.005	128×30×1	10	1.386	0.043	128×30×1	10	1.286	0.046	7.6	8.2
kra30a	128×30×5	10	2.002	0.551	128×30×2	9	9.651	1.514	128×30×4	9	11.87	1.038	5.9	4.8
kra30b	128×30×5	9	1.332	0.244	128×30×5	8	23.399	3.831	128×30×4	9	16.745	3.721	12.6	17.6
tai30b	128×30×3	10	0.947	0.182	128×30×3	10	22.649	2.16	128×30×1	10	7.203	1.984	7.6	23.9
tai35b	128×30×4	10	2.51	0.234	128×30×3	10	22.649	2.16	128×30×1	10	7.203	1.984	2.9	9
ste36b	128×30×4	10	3.337	0.334	128×30×4	10	33.274	4.13	128×30×2	10	14.675	1.213	4.4	10
tai40b	128×30×1	10	1.088	0.028	128×30×1	9	6.016	0.162	128×30×1	10	5.811	0.153	5.3	5.5

SE：標準エラー

ある。

- 各 MP で共有される高速なメモリである SM が一般に小さく，大きな問題に適用する場合に問題分割などの十分な工夫が必要となる。各 MP 間のデータ共有は VRAM 経由となる。基本演算命令の実行時間は 4 クロックであるのに対して，VRAM アクセスの遅延は 400 から 600 クロックである（GTX285 の場合）。このため MP 間でのデータ共有や交換の方法に工夫が必要となる。
- 各スレッドで用いる乱数系列の管理に工夫を要する。
- スレッドで用いるローカル変数にはレジスタが割当てられ，高速な処理が行える半面，進化計算で多用される配列がローカル変数として使えない。したがって，ローカル変数の効率的な使用には十分な工夫が必要となる。

以上の点に関しては今後の GPU の進歩にも期待できる。事実，NVIDIA は公表している次期 GPU（Fermi）では，SM の容量増加や VRAM アクセスに対するキャッシュの導入などが図られている [23]．しかし，GPU を並列進化計算に用いる研究テーマとしては，数百もの演算ユニットを如何に有効活用するかということが本質であり，進化計算のモデルまでさかのぼった研究を行うことが重要である。

Ⅶ. むすび

以上本論文では，一般的なユーザにおけるコンピュータ環境で，筆者が取り組んでいる並列化による進化計算の高速化に関する三つのアプローチを取り上げ，並列化の効果を述べた。また，今後の研究課題について考察した。

いずれも有効な結果が得られたが，特に GPU による並列進化計算は，高速化の度合いも大きく，今後の研究に期待が持てると思われる。現時点で利用可能な GPU はハードウェアの制限もあり，並列進化計算手法として必ずしも十分であるとはいえないが，この分野でのハードウェアの技術進歩は CPU 技術の進歩に劣らず著しいものがある。したがって今後は，この進歩に十分期待でき，そのような進歩を利用する並列化手法の研究が一層進むものと思われる。

CPU と GPU とでは，それぞれ本来の利用目的も異なることから当然得て不得てな点を持っている。したがって，全ての計算を GPU で行わせるのは必ずしも効率的とはいえない。この観点からは，複数の GPU による負荷分散や GPU とマルチコア CPU による並列計算との機能分担方式，さらにはネットワークで結合した並列計算との融合など，新しい並列進化計算のモデルの検討など重要な研究課題がある。今後これらの研究課題に鋭意取り組んでいきたい。

謝　辞

本研究を進めるにあたり，前学長の大槻眞一先生にはご支援とともにいつも励ましのお言葉をいただきました。ここに紙面を借りて深く感謝の意を表します。

GPU 計算の研究に関しては，大阪府立大学藤本典幸教授との共同研究の成果の一部です。なお，本研究の一部は，文部科学省基盤研究（C），課題番号 19500199 の補助金に基づいて行われました。

参考文献

1）伊藤：N700 系新幹線車両の概要～遺伝的アルゴリズムを用いた先頭車両形状設計～，計測自動制御学会北陸支部講演会（2009）.

2）S. Tsutsui：Cunning Ant System for Quadratic Assignment Problem with Local Search and Parallelization, Proc. of the 2nd Intl. Conf. on Pattern Recognition and Machine Intelligence, pp. 269-278, Springer LNCS 4815（2007）

3）筒井 茂義，劉 力綺，小島 基伸：カニングアントを用いた ACO の 2 次割当て問題への適用とその並列化，情報処理学会論文誌：数理モデル化と応用，Vol. 49, No. SIG 4（TOM 20），pp. 45-56，情報処理学会（2008）.

4）S. Tsutsui：Parallelization of an Evolutionary

Algorithm on a Platform with Multi-core Processors, Proc. of the 9th Intl. Conf. on Artificial Evolution (EA'09), Springer (2009).

5) S. Tsutsui and N. Fujimoto : Solving Quadratic Assignment Problems by Genetic Algorithms with GPU Computation : A Case Study, Proc. of the GECCO 2009 Workshop on Computational Intelligence on Consumer Games and Graphics Hardware CIGPU-2009, pp. 2523-2530, ACM (2009).

6) J. H. Holland : Adaptation in Natural and Artificial Systems, The University of Michigan Press (1975).

7) J. J. Grefenstette (Ed). Proc. of the First Intl. Conf. on Genetic Algorithms. Lawrence Erlbaum Associates (1985).

8) H.Mühlenbein and G. Paaß : From recombination of genes to the estimation of distribution I. binary parameters, Proc. of the Parallel Problem Solving from Nature (PPSN IV), pp. 178–187 (1996).

9) P. Larranaga and J.A. Lozano (eds) : Estimation of distribution algorithms, Kluwer Academic Publishers (2002).

10) E. Cantu-Paz : Efficient and Accurate Parallel Genetic Algorithms, Kuwer Academic Publishers (2000).

11) M. Dorigo, V. Maniezzo, and A. Colorni : The Ant System : Optimization by a Colony of Cooperating Agents, IEEE Trans. on SMC-PartB, Vol. 26, No. 1, pp. 29–41 (1996).

12) M. Dorigo and Stützle, T. : Ant Colony Optimization, MIT Press, Massachusetts (2004).

13D3) S. Tsutsui : *c*AS : Ant Colony Optimization with Cunning Ants, Proc. of the 9-th Intl. Conf. on Parallel Problem Solving from Nature (PPSN IX), pp. 162-171, Springer LNCS (2006).

14) 筒井：*c*AS：カニングアントを用いた ACO の提案, 人工知能学会論文誌, Vol. 22, No. 1, pp. 29-36, 人工知能学会 (2007).

15) Sun Microsystems, Inc. : Java 2 Platform, Standard Edition, v1.4.2 at API Specification. java.sun.com/j2se/1.4.2/docs/api.

16) Apache Software Foundation : Apache HTTP Server Project. httpd.apache.org.

17) 筒井：エッジヒストグラムを用いる順序表現向き確率モデル GA の提案, 人工知能学会論文誌, Vol. 18, No. 4, pp. 173-182, 人工知能学会 (2003).

18) J. L. Bentley : Fast algorithms for geometric traveling salesman problems, ORSA Journal on Computing, Vol. 4, pp. 387–411 (1992).

19) S. Mahfoud : A Comparison of Parallel and Sequential Niching Methods, Proc. of the 6-th Intl. Conf. on Genetic Algorithms, Morgan Kaufmann (1995).

20) Concorde TSP Solver, http : //www.tsp.gatech.edu/concorde.html (2006).

21) NVIDIA : CUDA Programming Guide 2.3, www.nvidia.com/object/cuda develop.html (2009).

22) 青木：フル GPU による CFD アプリケーション, 情報処理, Vol. 50, No. 2, pp. 107–115 (2009).

23) NVIDIA : Next Generation CUDA Architecture, www.nvidia.com/object/fermi architecture.html (2009).