分散遺伝的アルゴリズムアプリケーションGAPPA

川崎 高志，廣安 知之，三木 光範

同志社大学 工学部

本研究では，分散遺伝的アルゴリズム GAPPA を開発した．GAPPA は逐次処理だけでなく，共有メモリ型および分散メモリ型並列計算機上で並列処理が行える．特に，分散メモリ型並列計算機では MPI ライブラリでメッセージパッシングにより並列処理を行うように開発されているため，非常に多くのアーキテクチャーで利用が可能である．本研究では GAPPA の構成や実装方法について解説する．さらに，他の GA ライブラリと性能比較した．その結果，GAPPA は非常に良好な結果を示した．

An Application of Distributed Genetic Alogrithms: GAPPA

Takashi KAWASAKI,Tomoyuki HIROYASU, Mitsunori MIKI

Faculty of Engineering, Doshisha University

In this paper, the distributed genetic algorithm library is developed. The developed library is called "GAPPA". GAPPA can be operated not only on sequential computers but also on the parallel computers: shared memory parallel machines and distributed memory parallel machines. Especially, because GAPPA uses the MPI library for message passings, GAPPA can be performed on many types of architectures. In this paper, the concept and the implementation of GAPPA is explained. Comparing GAPPA with some GA Libraries, we achieved successful results.

1 はじめに

遺伝的アルゴリズム (Genetic Algorithms : GA) は生物の進化を模倣した確率的な最適化アルゴリズムである [1)]．この手法は，従来の最適化手法では解くことが困難であった複雑な連続および離散的問題に適用できる上，実装も比較的容易であるという長所がある．しかしながら，GA は膨大な反復計算を必要とするため，計算コストが極めて高いという問題点もある．これまでに多くの GA のためのフレームワークやライブラリが提供されているが，ほとんどが，開発されてからかなりの年月が経過している．そのため，近年，非常に利用が進んでいる並列処理に対応しているものは少なく，かた，MPI で実装された例は非常に少ない．GAPPA は共有メモリ型並列計算機に対応できる．得に分散メモリ型では，MPI によるメッセージパッシングライブラリを使用しているため，多くの並列アーキテクチャーに対応できる．本研究では，GAPPA の概要を述べ，他の GA ライブラリとの性能評価を行う．

2 遺伝的アルゴリズムの実装

2.1 評価

評価は，すべての個体に対して評価値を求める処理である．したがって，評価関数が複雑である場合には，非常に多くの時間を要する．GAPPA では，既に計算済みの適合度を記憶し，個体の情報が変化するまでは再計算を行わないことで処理速度の向上を図っている．解探索の序盤においては，母集団内の多様性は大きいため，その効果はあまり得られないが，中盤から終盤においては，この処理の有意性は高いと考えられる．

2.2 選択

選択は，評価された個体の適合度をもとに，次世代の母集団を生成する遺伝的オペレータである．選択については多くの研究がなされており，様々な手法が提案されている．

2.2.1 スケーリング関数による選択

スケーリングによる選択では，個体の持つ関数値 e に対して，あるスケーリング関数 $F(x)$ を用いて，その評価値 f を $f = F(e)$ として求め，その値の大きさに比例して選択される確率が高くなる．

ただし，スケーリングは序盤においてその淘汰圧が弱くなるという欠点を持っている．

GAPPA では，個体の関数値 e に加えて，さらに全個体の最大値 e_{max} と最小値 e_{min} を採る関数 $F(e, e_{max}, e_{min})$ を使って，$f = F(e, e_{max}, e_{min})$ とし，さらにスケーリング関数が無限の記憶を持つことを許すことにより，他の多くの選択手法をスケーリングで代用することを可能にした．コストは，関数呼び出しのみであり，$O(n)$ である．

2.2.2 ランキング選択

ランキング選択は適合度によって各個体に順位付けし，順位に応じた確率で個体を選択するという手法である．この処理は式 (1) で表せる．GAPPA では，この関数とよく似た挙動を示す関数（式 (2)）を用いて，疑似ランキング選択とした．この関数は，個体数がごく小さい内はランキング選択の代わりとしては代用できないものの，単純な選択方法としては有効であり，この選択手法の適用範囲を狭めるようなものではない．

$$m(g) = \left\lfloor \frac{2n+1-\sqrt{(2n+1)^2-4n(n+1)(1-g)}}{2} \right\rfloor \tag{1}$$

$$m(g) = \lfloor n(1-\sqrt{g}) \rfloor \tag{2}$$

2.3 交叉

交叉は染色体の組み替えにより新しい個体を生み出す遺伝的オペレータである．この処理は，交叉点を示すビットパターンを生成し，これを用いて 2 つの子供個体を生成する．この過程は，非常に単純であるため処理速度向上は考えにくい．しかし，ビットパターンの生成部分については高速化が可能である．特に，m 点交叉におけるビットパターン作成は複雑であり，高速化の余地は大きい．GAPPA の実装では，染色体のビット数分のシャッフル済みの数値リスト (128 ビットならば，0〜127 までのカードをシャッフルしたもの) を用意し，その中から，一つずつ数値を取り出して，該当のビットから染色体の最後までのビットを反転するという方法を採用した．コストは，染色体長 n，交叉点数 m のとき，$O(mn/2)$ である．

2.4 突然変異

突然変異は，染色体上のあるビットを一定の確率で反転させて新しい個体を生み出す遺伝的オペレータである．通常の実装では，すべての個体のすべてのビットに対して適当な乱数を発生させ，その値が突然変異率よりも小さければそのビットを反転させる．したがって，染色体長が長ければそれに伴ってこの処理のコストは高くなる．個体数 n_{pop}，染色体長 n_{ch} のとき，コストは $O(n_{pop}n_{ch})$ であり，計算の負荷は無視できない．

一般に，確率 p で起こりうる事象が n 回連続して起こりうる確率は，二項分布によって得られるので，染色体長 l，突然変異率が p_m のとき，n ビットの突然変異が起こりうる確率 $P_{mutate}(n)$ は (3) 式で与えられ，突然変異が n ビット以下である確率 $P_m(n)$ は，(4) 式で求められる．

$$P_{mutate}(n) = {}_lC_n p_m^n (1-p_m)^{l-n} \tag{3}$$

$$P_m(n) = \sum_{k=0}^{n} P_{mutate}(k) = \sum_{k=0}^{n} {}_lC_k p_m^k (1-p_m)^{l-k} \tag{4}$$

ここで，突然変異の数を決めるために，定義域 [0 1) の乱数 g を発生させるとすれば，(5) 式を満たすような n の値が，突然変異の数である．ただし，$P_m(-1) = 0$ と定義する．

$$P_m(n-1) \leqq g < P_m(n) \tag{5}$$

また，$P_m(n)$ は，染色体長 l，突然変異率が p_m のみに依存するため，この値は，それらのパラメータが設定された時点で確定することが出来る．従って，GAPPA の実装では，突然変異率に変更があった場合にのみこのテーブルを再計算するようにした．実行時には，効率よく n を探し出すことのみが問題となる．この値の探索には二分探索を用いた．突然変異率が $1/l$ の場合の計算コストはほぼ $O(n_{pop})$ である．図 1 に，この手法と通常の突然変異の所要時間を示す．*normal* が一般的な突然変異手法，*GAPPA* が GAPPA の実装である．

2.5 Easy To Use

図 2 は，SGA を実行するのに必要なすべてのコードを含んだプログラムの例である．GAPPA では，継承ベースのオブジェクト指向よりも，容易に使える部品としてのライブラリを提供することに重点を置いている．一般に継承といったオブジェクトベースの概念は理論が難しく，利用にも慣れを必要とする場合が多い．GAPPA は，RAD（Rapid Application Development）にならい，コ

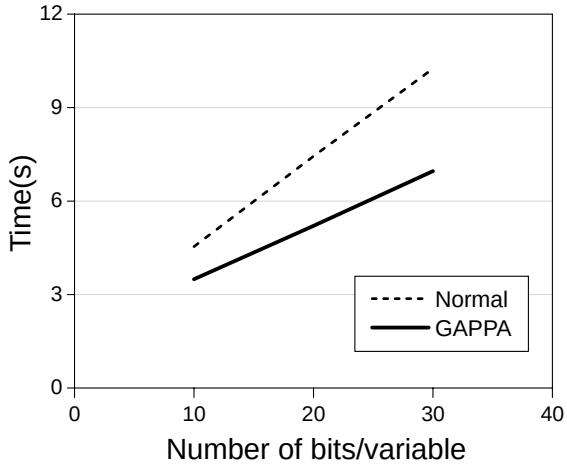

図 1: 突然変異による速度向上

ンポーネントベースプログラミングができるよう考慮して設計されている．

List 1-1

```
#include <iostream>
#include "gappa.h"

using namespace std;

main()
{
 BaseGAEnvironment ga(some_func);
 ga.loadConfiguration("sample.ini");

 ga.run();
 cout << ga(0) << ":"
  << ga(0).fitness() << endl;
 return 0;
}
```

図 2: 最も簡単なプログラム例

2.6 他の GA ライブラリとの比較

GAPPA と既存の GA ライブラリとの速度比較を行う．比較対象として，GENESIS[3]，GAlib[4] を使用する．また，対象問題として式 (6) で示される 10 次元の Rastrigin 関数を用いた．GA のパラメータは染色体長 100bit，母集団サイズ 100，エリート数 10，突然変異率 0.01，交叉率 0.6 とした．解精度と経過時間の推移を図 3 に示した．

$$f_{\mathrm{Rastrigin}} = 10n + \sum_{i=1}^{n}\left(x_i^2 - 10\cos(2\pi x_i)\right) \quad (6)$$

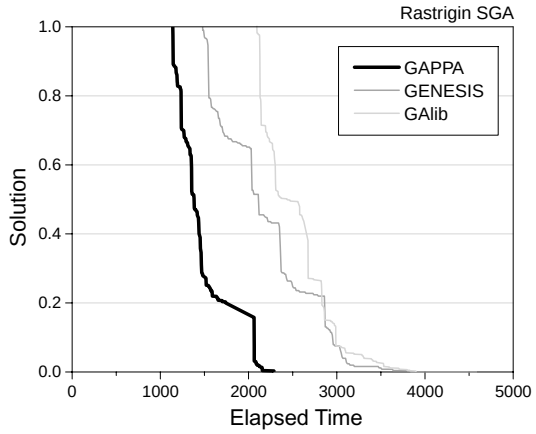

図 3: 他の GA ライブラリとの速度比較（SGA）

3 分散遺伝的アルゴリズム

分散遺伝的アルゴリズム (Distributed Genetic Algorithm： DGA) は，母集団を複数のサブ母集団に分割し，定期的に移住と呼ばれる個体情報の交換を行う粗粒度の GA 並列モデルである．DGA では並列化による速度向上の以外にも，SGA よりも高品質な解が得られると報告されている．また，DGA を拡張したモデルとしてサブ母集団ごとに個別の環境設定 (突然変異率，交叉率) を行う環境分散 GA[5] なども提案されている．

3.1 サブ母集団を基準としたクラス構築

DGA において，その GA の実行上中心的な概念となるのはサブ母集団である．GAPPA では，環境分散 GA も視野に入れた上で，環境クラス `GeneEnvironment` を構築した．このクラスは，サブ母集団に必要なパラメータをすべて保持し，個体の住む環境としても機能している．GAPPA の実装では，染色体の個体 `Gene` クラスのインスタンスが存在するためには，`GeneEnvironment` の存在が不可欠である．

4 並列環境への実装

並列分散 GA（Parallel Distributed GA : PDGA）は，DGA の並列環境への実装である．GAPPA では，スレッドモデルと MPI によるクラスタ，そして，TCP/IP ベースのクラスタをサポートする．

4.1 分散メモリと位置透過性

GAPPA では，共有メモリ型の DGA と MPI クラスタでの PDGA のプログラミングにおけるプログラミングスタイルのギャップを避けるために，コ

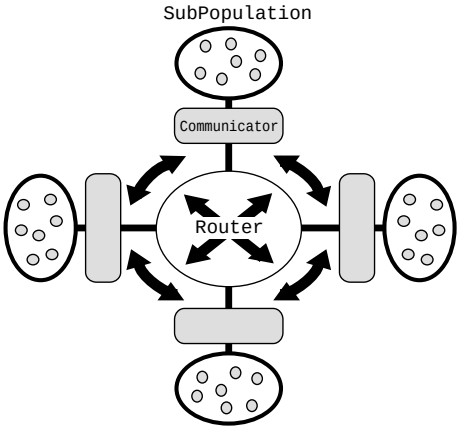

図 4: 位置透過性を実現するモデル

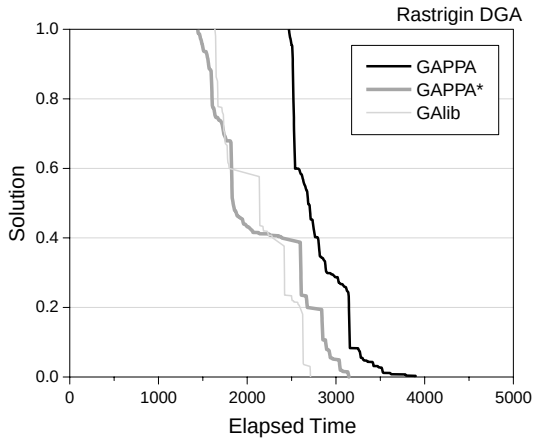

図 5: GAlib との速度比較 (DGA)

ミュニケータという概念を導入し，サブ母集団間の通信は，このコミュニケータを介してのみ行えるようにしている (図 4)．コミュニケータは，移住トポロジを決定するオブジェクトであるルータを使って移住個体の移住先を決定する．コミュニケータには，共有メモリ用の `SharedMemoryComm`，MPI クラスタ用の `MPICommunicator`，そして TCPI/IP 要の `TCPCommunicator` がデフォルトで用意されているが，`Communicator` を継承して，ユーザ定義のコミュニケータを作成することや，新たな通信プロトコルに対応することも可能である．コミュニケータは，通信経路を完全に抽象化するため，結果的に位置透過性が実現する．そして，ルータの実装により新しい移住トポロジを設計することが可能になっている．

4.2 他の DGA ライブラリとの比較

他の DGA ライブラリと GAPPA の性能比較を行う．比較対象として GAlib を用いた．対象問題は 10 次元の Rastrigin 関数であり，GA のパラメータは染色体長 100 ビット，母集団サイズ 100，エリート数 10，突然変異率 0.01，交叉率 0.6, サブ母集団数 8，移住率は 0.1 とした．解精度と経過時間の推移を図 5 に示した．

図 5 において GAPPA*としたものは，いくつかのパラメータにチューニングを施したものである．GAlib とノーマルの GAPPA を比較すると GAPPA の方が速度の面において劣っているが，高速化のためのオプションを使用することによって，GAlib に近い性能を示した．他のテスト関数についても同様の実験を行ったが，全ての関数について同様の結果が得られた．このことから，最適なスキームの選択などのチューニングによってはさらなる高速化が期待できると考えられる．

5 結論

本研究では共有メモリ型および分散メモリ型並列計算機に対応した並列分散 GA ライブラリ，”GAPPA”の開発を行った．GAPPA と他の代表的な GA ライブラリと比較では，SGA においては速度の面で高い性能を示したが，DGA においては GAlib の方が高速であった．しかしながら，最適なスキームの選択など簡単なチューニングによって，かなりの性能改善が図れることもわかった．高速化のための最適化が今後の課題である．なお，GAPPA は http://www.is.doshisha.ac.jp/gappa/ において公開されている．

参考文献

1) D.E.Goldberg, ”Genetic Algorithms in Search Optimization and Machine Learnig”, Addison-Wesley, Reading, Mass.(1989)

2) Reiko Tanese, ”Distributed Genetic Algorithms”, Proc. 3rd International Conference on Genetic Algorithms, P.434-439. (1989)

3) **GENESIS Version 5.0** Copyright (c) 1990 by John J. Grefenstette.

4) **GAlib 2.2.4** Copyright (c) 1994-1996 MIT, 1998-1999 Matthew Wall. `http://lancet.mit.edu/ga/`

5) M.Miki, T.Hiroyasu, M.Kaneko, K.Hatanaka, ”A Parallel Genetic Algorithm with Distributed Environment Scheme”, EEEProceedings of Systems, Man and Cybernetics Conference SMC’99.(1999)