遺伝的アルゴリズムの並列計算フレームワーク GAROP

山中 亮典
Ryosuke YAMANAKA

1　はじめに

大規模な最適化問題を解くために遺伝的アルゴリズム (Genetic Algorithm: GA) [1] が用いられている．一般的に，GA を用いて良好な解を得るには膨大な演算量が必要であり，対象問題によっては現実的な時間内に解を求めることが難しい場合がある．そのため，演算量を削減，もしくは高速に処理することが課題となっている．GA は解候補集団による多点探索を用いて大域的な探索を実現している．多数の解候補に対する操作を行いながら探索を進めるため，並列処理との親和性が高い．

一方，一般のパーソナルコンピュータ（Personal Computer: PC）を用いた様々な規模の PC クラスタや，マルチコア CPU，および GPU など様々な構成のハードウェアが普及してきている．しかし，それらの計算資源を使用するには，各資源に独立のプログラミングが必要である．そのため，使用するハードウェアを変更するにはプログラムを変更しなければならない．また，計算資源の性能を引き出すためには，アーキテクチャのメモリ階層向け最適化やスケーラビリティを達成するための通信と計算をオーバーラップする技術が必要である．このような複雑かつ煩雑なプログラミングは生産性に欠ける．

我々は，これらの課題を解決するために，GA の並列計算フレームワーク GAROP（Genetic Algorithms framework for Running On Parallel environments）を提案している [2]．ユーザ*1は，並列処理に関する特別な知識を必要とせず，一般的なプログラミングと同等の記述方法で処理時間を短縮できる．GAROP における GA の並列手法は，個体間で独立の処理である評価計算を並列化するマスタ・スレーブモデル [3] である．GAROP では，ユーザと並列計算環境とのインターフェースとして個体プールという概念を導入している．並列計算環境の構築はユーザが行う必要があるが，プログラミングの際に計算資源のアーキテクチャを意識する必要はなく，個体プールの概念を使用するのみで並列処理を実現する．

本研究では，マルチコア CPU，GPU，および Windows クラスタに対する GAROP を実装している．本稿では，CUDA(Compute Unified Device Architecture) 対応 GPU を例に GAROP によるプログラミング量削減について説明し，速度向上率に対して GAROP を評価する．

2　遺伝的アルゴリズム

GA は生物が環境に適応していく仮定を工学的に模倣した最適化アルゴリズムである [1]．自然界における生物の進化過程においては，ある世代を形成している個体集団の中で，環境に適応した個体がより高い確率で生き残り，次世代に子を残す．この生物進化のメカニズムをモデル化し，環境に対して最もよく適応した個体，すなわち目的関数に対して最適値を与えるような解を計算機上で求めることが GA の概念である．

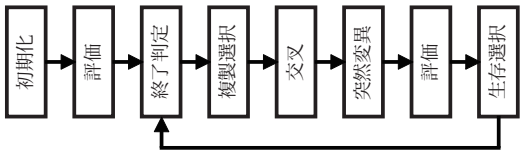

Fig.1　GA の流れ

GA は Fig. 1 に示す流れに沿って行われる．GA では母集団の各個体に対して交叉，突然変異といった遺伝的操作を施し，新しい個体を生成する．その後，新しい個体に対する評価を行い，優れた個体を選択し，次世代に残す．これら一連の操作を定められた終了条件まで繰り返すことで，解を探索する．

GA の各個体は染色体によって特徴を持ち，染色体は遺伝子の集まりから構成される．GA では 1 つの染色体で 1 つの個体を表す．最適化問題の設計変数値が染色体へとコーディングされ，GA の各操作は染色体あるいは遺伝子に対して行われる．

大規模問題を GA で解く場合，評価に用いる目的関数が非常に複雑になる．そのため，個体の適応度を評価する計算に膨大な時間がかかる．

3　GAROP

GAROP は，GA を並列計算環境下で実行する際のモデルを定義し，そのモデルを実現するためのアプリケーションレベルフレームワークである．GAROP の目的は，ユーザが特別な並列化プログラミング技術を有する必要なく，マスタ・スレーブ型の並列処理を実現することである．並列処理に関する API（Application Programming Interface）を提供することで，逐次プログラムと同程度の記述を保ちながら並列化による恩恵を受けられる．

GAROP では，ユーザは任意の GA を構築し，評価部以外の部分を実装する．各並列計算環境に応じた評価部のテンプレートを用い，対象問題のコードと組み合わせる事により，評価部の実装を行う．このテンプレートを利用することで，特殊な通信と評価タスクのスケジューリング実装をユーザから隠蔽できる．すなわち，ユーザは通信や計算資源に関する知識がなくとも，実行する並列計算環境に適したアルゴリズムを構築できる．

*1 GA の開発者

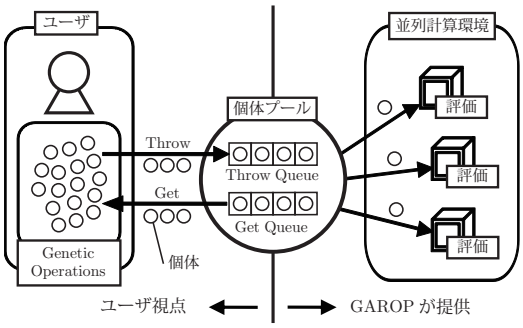

Fig.2　GAROP の概要

Table1　GAROP の API

関数名	動作
initialize	個体プールの作成
	並列計算環境の初期化
throw	データを個体プールに登録
get	個体プールからデータを取得
finalize	確保したメモリの解放
	並列計算環境の接続破棄

Fig. 2 に GAROP の概要を示す．GAROP では，ユーザと並列計算環境を結ぶインターフェースとして個体プールの概念を導入し，上記事項を実現する．

3.1　個体プール

個体プールは評価すべき個体の溜まり場であり，内在する個体を自動的に並列評価する．ユーザは評価したい個体を 1 個体ずつ個体プールに登録する．そして，必要な時に個体プールから個体を取得することで評価済みの個体を得ることができる．

個体プールは 2 つのキューから構成されている．登録された個体を格納する Throw キュー，および評価済みの個体を格納する Get キューである．GAROP では，Throw キューを監視するバックグラウンドスレッドが存在し，個体の格納と同時に個体データを計算資源へ送信する．計算資源は受信した個体データを評価し，評価値もしくは個体データを Get キューに格納する．

3.2　GAROP の API

GAROP はライブラリレベルのアプリケーションであり，Table 1 に示す 4 つの関数を提供する．initialize 関数では，バックグラウンドスレッドの生成や並列計算環境の設定および初期化を行う．throw 関数は個体データを個体プールに登録し，get 関数は個体プールからデータを取得する．その際，個体がどのようなデータ構造で記述されていても対応できるよう，BYTE 単位のデータ列に変換する必要がある．また，throw/get するデータのサイズを BYTE 単位で指定する．finalize 関数は GAROP の確保したメモリの解放や，スレッドの破棄，および並列計算環境との接続を切断する．これら 4 つの関数は，どのような計算環境を用いる場合でも不変である．

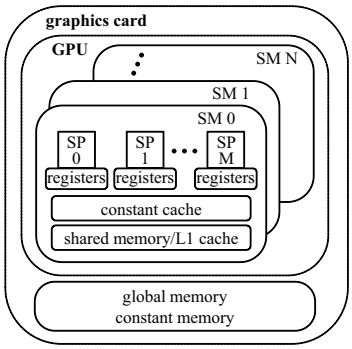

Fig.3　CUDA 対応 GPU のアーキテクチャ

3.3　GAROP の使い方

GAROP に基づいて GA を実行する場合に，ユーザの行う作業を以下に示す．

1. 並列計算環境の構築
2. テンプレートを用いた評価関数の実装
3. GAROP の API を用いた GA の実装
4. コンパイル
5. 実行ファイルを計算資源に配置
6. 実行

評価計算を実行するのは並列計算のための計算資源である．そのため，評価関数は使用する計算資源上で実行可能な記述を行う必要がある．対象問題に依存する評価関数の実装を GAROP 提供者が担うのは現実的ではないため，評価計算の実装はユーザが行う．この時，GAROP の提供するテンプレートを用いることで，並列計算の恩恵を受けられる．そのテンプレートは用いる環境，具体的にはプログラミング言語によって異なる．

4　GPU における GAROP

CUDA は NVIDIA の GPU 向け並列計算アーキテクチャである．CUDA C/C++ という C/C++ 言語の拡張言語を使用した CUDA は，GPU を用いた汎用計算を容易にした．しかし，逐次プログラムと比較すれば考慮すべき点は多い．CUDA における GPU アーキテクチャを Fig. 3 に示す．GPU チップ内部には，ストリーミングマルチプロセッサ (Streaming Multi Processor: SM) が複数ある．さらに SM 内部には，ストーミングプロセッサ (Streaming Processor: SP) と呼ばれる最小単位の演算コアがある．また，容量およびアクセス速度の異なる複数種類のメモリを搭載している．

本章では，CUDA 対応 GPU におけるプログラミングに要する知識を説明し，GAROP によって労力を削減できることを示す．

4.1　スレッドの階層構造

CUDA では，膨大な数のスレッドを起動し SP によって演算を行う．しかし，膨大な数のスレッドを 1 系列の整理番号で管理するのは困難である．そこでグリッドおよびブロックという概念を導入し，その中で階層的にス

レッドを管理する．概念的には，グリッドの中に複数のブロックがあり，ブロックの中に複数のスレッドがある．ハードウェア的には，スレッドは SP によって処理され，ブロックは SM によって処理される．グリッド，ブロックおよびスレッドの数は，ホストからカーネル関数を呼び出す際に指定する必要がある．その際，用いる GPU の SM 数や SP 数を考慮し，適切に値を設定しなければ速度向上を実現することは難しい．

4.2 メモリの階層構造

ビデオカードには大きく分けて 2 種類のメモリが搭載されている．GPU 内に搭載されているオンチップメモリ，およびビデオカード上に搭載されているオフチップメモリである．オンチップメモリは，容量は少ないが高速にアクセスできる．オフチップメモリは，容量は大きいがアクセスが低速である．CUDA では，レジスタメモリ，シェアードメモリ，グローバルメモリ，テクスチャメモリおよびコンスタントメモリを使用可能である．GPU を用いてパフォーマンスを向上させるには各メモリの特徴を理解し，アクセス速度を考慮するプログラミングが重要である．特に，CPU 上のメインメモリとデータをやり取りするグローバルメモリ，および SM 内の SP で共通に使用できるシェアードメモリを有効に利用する必要がある．

4.3 GPU のための GAROP 実装

前述のように，GPU を用いたプログラミングにおいてスレッド数および使用メモリなどのパラメータは非常に重要である．しかし，CUDA 対応 GPU はバージョンごとにアーキテクチャが異なり，最適なパラメータは変化する．また，今後も新しいアーキテクチャが登場すると予想される．GAROP では，用いる GPU の構成を取得し，使用スレッド数を静的に決定する．また，各メモリの容量と個体プールに throw される個体サイズから，最も処理を高速化できるメモリ領域へ個体を配置する．GAROP は，各 GPU に適したパラメータを決定する労力を 0 にし，処理速度の向上を実現する有用なフレームワークである．

5 GAROP の評価

本章では，実装しているライブラリを使用して GA を実行し，その速度向上率および記述プログラムに関して評価する．具体的には，GAROP に基いて実装した SGA (Simple GA) [1] を各環境で実行し，対象環境でのシングルコア実行時と比較する．対象とする環境は Table 2 に示す 3 つである．Windows クラスタを構成するマシンのスペックを Table 3 に，マルチコア CPU を搭載するマシンのスペックを Table 4 に，GPU のスペックを Table 5 に示す．

- 実験 1 : Windows クラスタ
  対象問題として JAXA 宇宙科学研究所宇宙輸送工学研究系 [4] より公開されているハイブリッドロケットエンジン（HRE）概念設計最適化問題 [5] を使用する．実行する SGA のパラメータを Table 6 に示す．
- 実験 2 : マルチコア CPU
  対象問題として 1-max 問題を使用する．ただし，大規模問題を模擬するため，100,000 回の繰り返しを行う．実行する SGA のパラメータを Table 7 に示す．
- 実験 3 : GPU
  対象問題および SGA のパラメータは，マルチコア CPU での実験と同様である．また，GPU 搭載マシンとして Table 4 を使用する．

Table2　実装済み環境

並列計算環境	言語
Windows クラスタ	C#
マルチコア CPU	C/C++
CUDA 対応 GPU	CUDA

Table3　Windows クラスタを構成するマシン

OS	Windows Server 2008 HPC Edition
メモリ	8 GB
プロセッサ	AMD Opteron 2356 × 2
周波数	2.30 GHz
コア数	4

Table4　マルチコア CPU 搭載マシン

OS	Debian 4.1.2
メモリ	6 GB
プロセッサ	Intel Xeon W3530
周波数	2.80 GHz
コア数	8

Table5　GPU のスペック

グローバルメモリ	2.68 GB
SM 当たりのシェアードメモリ	65536 Bytes
アーキテクチャ	Tesla C2050
周波数	1.15 GHz
SM 数	14
SP 数	448

Table6　SGA のパラメータ（HRE 設計最適化問題）

母集団サイズ	64
染色体長	41
世代数	32

Table7　SGA のパラメータ（1-max 問題）

母集団サイズ	64
染色体長	64
世代数	100

List. 1　Windows クラスタでの GAROP 使用コード (抜粋)

```
Individual[] population = InitPopulation();
// initialization of GAROP
GAROP g = new GAROP();
for( j = 0; j < generation_limit; j++ ) {
  for( i = 0; i < population_size; i++ )
    // throw individuals to Individual Pool
    g.Throw( population[i] );
  for( i = 0; i < population_size; i++ )
    // get individuals from Individual Pool
    g.Get( population[i] );
  selection( population );
  crossover( population );
  mutation( population );
}
g.Finalize(); // finalization of GAROP
```

List. 2　マルチコア CPU および GPU での GAROP 使用コード（抜粋）

```
Individual[] population = InitPopulation();
// initialization of GAROP
Initialize( sizeof(Individual) );
for( j = 0; j < generation_limit; j++ ) {
  for( i = 0; i < population_size; i++ )
    // throw individuals to Individual Pool
    Throw( (BYTE*)&population[i] );
  for( i = 0; i < population_size; i++ )
    // get individuals from Individual Pool
    Get( (BYTE*)&population[i] );
  selection( population );
  crossover( population );
  mutation( population );
}
Finalize(); // finalization of GAROP
```

Table8　使用スレーブに対する速度向上率

Windows クラスタ	0.82
マルチコア CPU	0.75
GPU	0.05

5.1　実験結果

List. 1 に GAROP を用いて実装した Windows クラスタ用 SGA の一部を，List. 2 にマルチコア CPU および GPU 用 SGA の一部を示す．マルチコア CPU と GPU は，使用言語が共通して C 言語であるため，同様のコードである．両コードともに，一般的な関数を使用する記述のみしか使用していないことが確認できる．

Fig. 4 に，シングルコア実行時を 1 とした場合の，GAROP を用いた並列実行時の速度向上率を示す．スレーブプロセッサとして，Windows クラスタは 16 ノード，マルチコア CPU は 7 コア，GPU は 64 コアを使用している．また，使用スレーブプロセッサ数に対する速度向上率を Table 8 に示す．

Fig. 4 より，Windows クラスタでは 13.07 倍，マルチコア CPU では 5.25 倍，GPU では 2.94 倍の速度向上が確認できた．Table 8 より，最も並列性能が高いのは Windows クラスタであり，GPU は使用コア数に対する速度向上率が極めて低い結果となった．

6　まとめと今後の展望

本研究では，GA を並列計算環境で実行するためのフレームワーク GAROP を提案している．また，GAROP

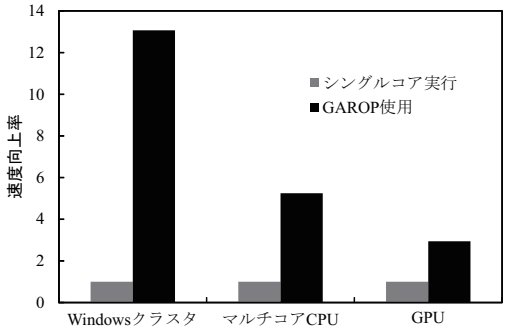

Fig.4　シングルコア実行時に対する速度向上率

を実現するためのライブラリをマルチコア CPU，GPU，および Windows クラスタ環境で実装している．マルチコア CPU および GPU 環境において，1-max 問題を用いた性能評価の結果，それぞれ 5.25 倍および 2.94 倍の速度向上を実現した．Windows クラスタ環境では，実問題のひとつであるハイブリッドロケットエンジンの概念設計最適化問題を対象に，13.07 倍の速度向上を実現した．その際，各環境においてプログラムは一般的な記述のみであった．GAROP は逐次プログラムと同等の記述で並列処理の恩恵を受けられる有用なフレームワークであると考えられる．

今後の展望として，地球シミュレータを対象としたライブラリを実装する予定である．大規模なクラスタにおけるスケーリングを確認するとともに，膨大な演算資源を有効活用できるフレームワークを目指す．GAROP は個体並列が考えの根本にあるため，並列度数をそれほど上げることができない．そこで，余った演算資源を最大限に使用するため，個体の遺伝子情報をランダムに変更し，未知個体を評価させる仕組みを提案する予定である．

参考文献

1) D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.
2) T. Hiroyasu, R. Yamanaka, M. Yoshimi, and M. Miki. GAROP: Genetic Algorithm framework for Running On Parallel environments. 数理モデル化と問題解決研究報告, Vol. 2012, No. 5, pp. 1–6, 2012.
3) E. Alba and J. M. Troya. A Survey of Parallel Distributed Genetic Algorithms. *Complexity*, Vol. 4, No. 4, pp. 10–11, 1999.
4) JAXA 宇宙科学研究所宇宙輸送工学研究系. http://flab.eng.isas.jaxa.jp/.
5) 幸寛小杉, 聖大山, 孝藏藤井, 雅博金崎. ハイブリッドロケットエンジンの概念設計最適化. 宇宙輸送シンポジウム講演論文集 [CDROM], Vol. 2009, No. 75, 2009.