A-002

GPGPU を用いた遺伝的アルゴリズムにおける疑似エリート保存戦略
Pseudo Elitism in Genetic Algorithms Using GPGPU

懸樋 亮一† 井口 寧†
Ryoichi Kakehi Yasushi Inoguchi

1. はじめに

遺伝的アルゴリズム (Genetic Algorithm, GA)は、個体群を用いた多点探索型の最適化アルゴリズムであり、進化論に基づいた強力な探索手法である。しかし、GA は計算量が多く、大規模な問題では実行時間が大きな課題となる。そこで GPU の並列計算能力を利用した GA の高速化が注目されている。本研究では、GPU を用いて GA の実行を高速化するアプローチとして「疑似エリート保存戦略」を提案する。この提案手法はソートをせずに、リダクションを用いて効率的にエリート個体を選択する手法であり、ONEMAX 問題に適用して、大幅な実行速度の改善を確認した。

2. 研究の背景と目的

2.1 研究の背景

GA は設計最適化問題などで広く利用されているが、遺伝的オペレーションの各ステージで膨大な計算を必要とし、大規模な問題に対しては実行時間が課題となる。一方、GPU は、単一命令複数データ(Single Instruction, Multiple Data, SIMD)アーキテクチャに基づく並列処理能力を有しており、GA の各ステージを、GPU を用いて並列化することで計算時間を大幅に短縮することが可能である。しかし、従来のエリート保存戦略はデータのソートを必要とし、GPU における実行効率が低下するという問題があった。

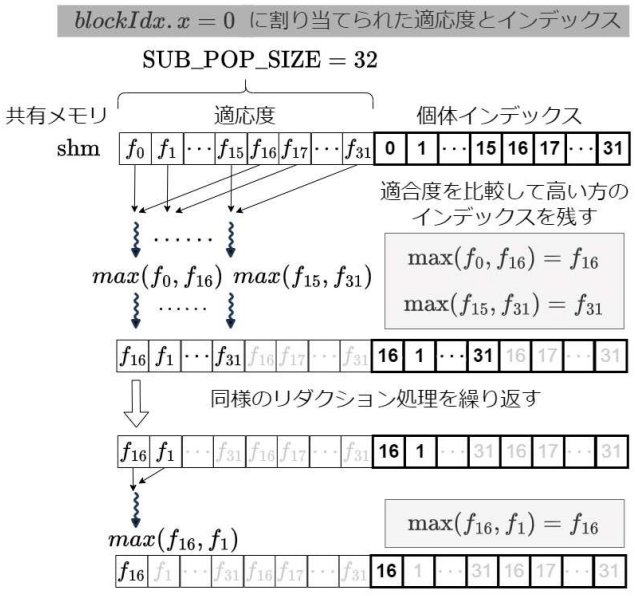

図 1 リダクションによる疑似エリート選出

† 北陸先端科学技術大学院大学 Japan Advanced Institute of Science and Technology

2.2 研究の目的

本研究の目的は、GPU を用いた GA の高速化において、エリート保存戦略を効率的に実装する新しい手法を提案することにある。具体的には、ソートを必要としない「疑似エリート保存戦略」を開発し、その性能を評価する。

3. 提案手法

3.1 疑似エリート保存戦略の概要

従来のエリート保存戦略では、個体群を適応度順にソートし、上位の個体を次世代に引き継ぐ方法が一般的である。しかし、このソート処理はデータ依存性が高く、GPU の並列処理能力を十分に活用できない。そこで本研究では、リダクション処理を用いることでソートを行わずにエリート個体を効率的に選択する「疑似エリート保存戦略」を提案する。これにより並列処理の効率を最大限に引き出し、GA の実行速度を大幅に向上させることが可能となる。

3.2 疑似エリート保存戦略の詳細

疑似エリート保存戦略の疑似コードを Algorithm 1 に、そのリダクション処理部を図 1 に示す。

Algorithm 1 疑似エリート保存戦略

```
      Input: popData                          // 個体群データ
      Output: popData.eIdx      // 疑似エリート個体の Index
1:    function PseudoElitismKernel(popData)
2:        SUB_POP_SIZE← blockDim.x
3:                                  // 部分個体群のサイズ
4:        eIdx←blockIdx.x   // 疑似エリートインデックス
5:        tIdx←threadIdx.x              // Local thread Index
6:        gIdx←tIdx+eIdx*SUB_POP_SIZE   // Global index
7:        shm[2 * SUB_POP_SIZE]          // 共有メモリ
8:        Load fitness and index into shared memory
9:        syncthreads()
10:       for stride←SUB_POP_SIZE/2 to 0 step stride>>1 do
11:           if tIdx < stride then
12:               bestIdx ← (shm[tIdx]
13:                   ≥ shm[tIdx + stride])
14:                   ? tIdx : tIdx + stride
15:               shm[tIdx] ← shm[bestIdx]
16:               shm[tIdx + SUB_POS_SIZE]
17:                   ←shm[bestIdx+SUB_POP_SIZE]
18:           end if
19:           syncthreads()
20:       end for
21:       if tIdx = 0 then
22:           popData.eIdx[eIdx] ← shm[SUB_POS_SIZE]
23:       end if
24:   end function
```

最初にグローバルメモリ上に個体群の適応度を格納した配列を準備する。後ほどリダクション処理を行う為に、配列の要素数が 2 の冪乗となるように調整する。余分な領域は 0 パディングし、エリート個体の数(2 の冪乗に限定する)で等分割した部分個体群を GPU の各ブロックに割り当てる(Alg.1:行 2)。ブロックとはスレッドのグループであり、各ブロックは独立して並列処理を実行する。共有メモリは各ブロック単位で存在し、同じブロック内のスレッド間で高速なデータ共有を可能とする。そこで疑似エリート個体を選出するリダクション処理の高速化の為に、適応度(図 1:$f_0 \sim f_{31}$)を共有メモリにコピーする。その際、適応度に続けて部分個体群のインデックス(図 1:0 ~ 31)もコピーする(Alg.1:行 7-9)。この配列に対してマックスリダクションを適用する(Alg.1:10-20)。図 1 は部分個体群のサイズが 32 でブロックのインデックスが 0 の場合に行われるリダクション処理の具体例を示したものである。f_0とf_{16}を比較するとf_{16}の適応度が大きいので、f_{16}を共有メモリ shm の最初の要素に格納し、shm の後ろ半部の最初の要素に 16 を格納する。このリダクション処理を繰り返した結果、インデックス 16 の個体が部分個体群の中の最優良個体として選出された(図 1:最下部)。この個体を疑似エリートと呼び、本処理を分割した全ブロックで実行することで、所望の疑似エリートを全て求める事が出来る(Alg.1:行 22)。

4. 実験・評価

4.1 実験条件

本研究では、ONEMAX 問題をベンチマークとして採用し、提案手法の性能を評価した。ONEMAX 問題は、染色体を構成するビット(0 または 1)の総和を最大化する事を目的とする最適化問題である。エリート個体の数は 8、個体群と染色体のサイズは共に 32 から 1024 まで 32 刻みで設定し、実行世代は 512 とした。実験には NVIDIA A100 GPU を用い、提案手法の性能評価には、NVIDIA 提供の CUB ライブラリ内の **cub::DeviceRadixSort()**を用いたエリート保存戦略と比較した。また、次節以降で示す実行時間はデータ転送時間を含まない。

4.2 実行時間の比較

図 2 に実験結果をボックスプロットにて示す。

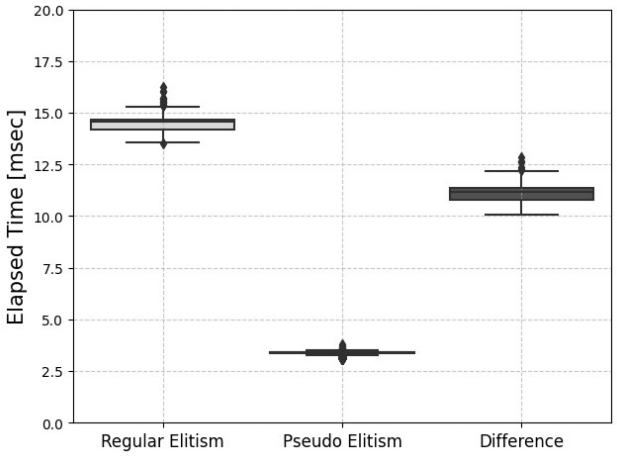

図 2　エリート保存戦略実行時間比較

Regular Elitism が Radix ソートを用いたエリート保存戦略の実行時間、Pseudo Elitism が提案手法である疑似エリート保存戦略による実行時間を示す。Difference は両者の差分であり、単位はいずれもミリ秒である。 平均実行時間はそれぞれ 14.41 [msec]、 3.36 [msec]であり、その差は 11.05 [msec]である。疑似エリート保存戦略を採用する事で 4.29 倍ほど高速化している事が確認できた。

4.3 適応度曲線の比較

図 3 は染色体のサイズが 1024 で、個体群のサイズが 128 の時と 1024 の時のエリート保存戦略(“regular elitism”)と疑似エリート保存戦略(“pseudo elitism”)の適応度曲線を示している。図からどちらの戦略においても適応度曲線には殆ど

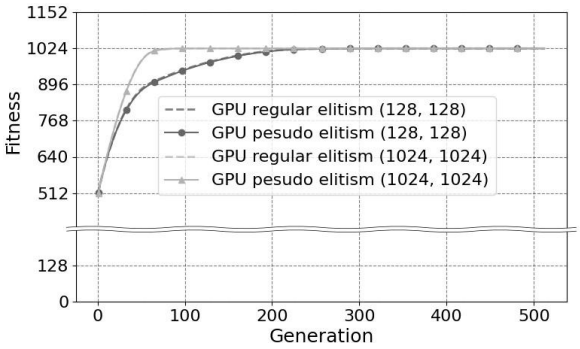

図 3　適応度曲線比較

差が無い。ONEMAX 問題においては、疑似エリート保存戦略を用いた GA は、Radix ソートを用いた従来の手法と殆ど同等の収束性能を示し、高い適応度の解を迅速に見つける事が確認出来ている。

5. おわりに

本研究では、GPU 上でのソートを必要としない新しいエリート保存戦略である「疑似エリート保存戦略」を提案した。提案手法は、従来のエリート保存戦略に比べて 4.29 倍の計算時間の短縮を実現し、GA の効率的な実行を可能にした。今後の課題は、より複雑な最適化問題への提案手法の適用である。

参考文献

[1] Stefano Debattisti, Nicola Marlat, Luca Mussi, and Stefano Cagnoni. Implementation of a simple genetic algorithm within the CUDA architecture. (2009)
[2] John Runwei Cheng and Mitsuo Gen. Accelerating genetic algorithms with GPU computing: A selective overview. Comput. Ind. Eng., 128:514–525, February 2019.
[3] Noriko Ohtani. 進化計算アルゴリズム入門. オーム社, 2018. ISBN 9784274222382.
[4] Uger Cekmez, Mustafa Ozsiginan, and Ozgur Koray Sahingoz. Adapting the GA approach to solve Traveling Salesman Problems on CUDA architecture. In 2013 IEEE International Symposium on Computational Intelligence and Informatics, 19-21 IEEE, November 2013
[5] Jiri Jaros. Multi-GPU Island-Based Genetic Algorithm for Solving the Knapsack Problem. In 2012 IEEE World Congress on Computational Intelligence. IEEE, Jun 2012.