GPU を用いた遺伝的アルゴリズムの並列計算フレームワークの提案

蔵野 裕己

1　はじめに

近年，コンピュータ処理の高速化を目的とした GPU（Graphics Processing Unit）による並列処理に関する研究が行われている [1) 2)]．しかし十分な高速性を得るには，並列性の高い処理を行う必要がある．また，並列処理には専門性の高いコーディング技術や性能のチューニングに要する開発コストなど，改善すべき課題が多い．

一方，進化計算の分野では最適化問題を解くための様々なアルゴリズムの研究が行われている．その中の一つに GA（Genetic Algorism：遺伝的アルゴリズム）というアルゴリズムがある．その適用例にはトラス構造物の重量最適化問題 [3)]，タンパク質の立体構造予測問題 [4)] などがあり，これらの問題において良好な解を得るためには，多く複雑な評価計算を行う必要がある．この評価計算という処理は，計算回数が多い一方でデータ並列性を持つ処理であるため，並列化することで高速化できる可能性が高い．

本研究では，並列処理に関する専門的な知識を持たない GA 開発者にも並列処理環境を利用しやすくする並列計算フレームワークを提案，実装し，実装方法や計算性能などを議論する．

2　GA 向け並列処理フレームワークの構築

ここでは，GPU で GA の評価計算を行うためのフレームワークについて述べる．

2.1　Genetic Algorithm

図 1 に本フレームワークの構造を示す．

GA では，まず初期母集団を生成し，図 1 の CPU 内の処理のようにその母集団中の各個体に対して交叉 (Cross Over)，突然変異 (Mutation)，評価 (Evaluation)，選択 (Selection) という処理を繰り返し行い，この一回のループを一世代と数える．これらの処理を繰り返すことでよりよい個体が残り，最適解に近づいていく．

2.2　フレームワーク

フレームワークとは，よく使う機能などをまとめた，アプリケーションの枠組みとなるものである．本稿では特に，GPU での処理に関する機能をまとめて持ち，GPU による評価計算を用いた GA 開発の枠組みとなるもののことである．

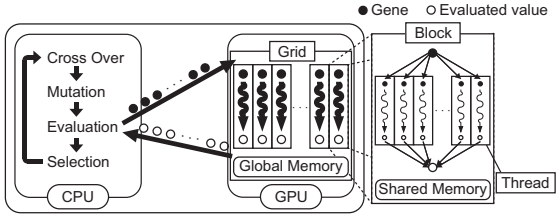

Fig.1　本フレームワークの構造

2.3　提案

本フレームワークは，利用者が特別な知識を持たなくても簡単に GPU による並列計算を利用できることを目的とする．そのため，GPU で行う評価計算以外の処理は利用者が自由に記述することができ，GPU による評価計算は簡単な関数の呼び出しで実現できるフレームワークを提案する．構造としては，図 1 に示すように評価計算の部分を GPU で並列処理し，高速化を図る．

3　GPU を用いたフレームワークの実装

本稿では CUDA(Compute Unified Device Architecture) という言語を用いて GPU の処理を記述する．CPU での処理は，通常の CUDA では C++ で記述するが，本フレームワークでは PyCUDA という言語バインディングを用いて Python で記述する．

3.1　CUDA における並列処理の概念

CUDA では，多数のスレッドと呼ばれる実行単位を同時に実行することで並列処理を実現する．さらに，大規模な計算において大量になってしまうスレッドを管理しやすくするためにグリッド，ブロックという概念が存在する．スレッドはブロックの中に定義され，同じようにブロックはグリッドの中に定義される．ただし，一つのブロック内，またグリッド内で定義できるスレッドやブロックの数には制限があり，この制限は GPU ごとに違う．本フレームワークにおいては，それらの制限を超える大規模な評価計算を要する GA は想定していない．また，これらのスレッドが使うメモリにはいくつかの種類がある．グローバルメモリと呼ばれるメモリは，単一グリッド内でのみ共有可能である．シェアードメモリと呼ばれるメモリは，単一ブロック内でのみ共有可能であり，グローバルメモリに比べて非常に高速である．他にも数種類のメモリが存在するが，本フレームワークでは特に基本的なこれら 2 種類のメモリを使用した．

3.2　評価計算の並列化

本フレームワークでは，二つの並列性を利用して評価計算を並列化する．一つは，各個体のそれぞれが持つ値を用いて計算するため，評価計算を個体毎に独立して行うことができるという並列性である．もう一つは，評価計算自体のもつ並列性である．図 1 に示すように，一つの個体の評価計算を一つのブロックで行い，これらのブロックを同時に実行することで並列に処理を行う．さらに各ブロック内で，評価計算を並列な処理ごとに切り分けたものを一つのスレッドに割当てて，これらのスレッドを同時に実行する．この際，各スレッドの計算結果をシェアードメモリを用いて共有することで，メモリアクセスの時間を抑えることができる．

3.3　GPU による処理の呼び出し

本フレームワークは PyCUDA を用いているため，GA 開発者は評価計算の処理以外を Python で記述する．擬

```
1: from Gpu_evaluation import Evaluation
2: for(number of generation):
3:     Crossover(genes)
4:     Mutation(genes)
5:     evaluated_values = Evaluation(genes)
6:     Selection(genes)
```

Fig.2　GPU での評価計算における PyCUDA の擬似コード

似コードを図 2 に示す．図 2 の 1 行目に示すように，まずフレームワークをインポートする．そして評価計算時に，5 行目のようにして本フレームワークを関数として呼び出す．その際，引数として評価計算させたい遺伝子を格納した一次元のリストを関数に渡し，返り値として評価値が格納された一次元のリストを得る．これら 2 つのリストのインデックスは共通で，それぞれが指す値は同じ個体のものである．

4　実装例

GA のプログラムを開発し，本フレームワークを用いて GPU で評価計算を行った．この実装では，評価計算に以下に示すような Rastrigin 関数の関数値最小化問題を用いた．

$$F_{\mathrm{Rastrigin}}(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i))$$

$$(-5.12 \leq x_i < 5.12)$$

$$\min(F_{Rastrigin}(x)) = F(0, 0, ..., 0) = 0$$

なお，式中の n は次元数を表す．本フレームワークに従う場合，各個体の Rastrigin 関数の計算がブロックに割り当てられる．Rastrigin 関数の総和計算の繰り返し処理は，各ループを独立に計算できるため並列性を持つため，ブロック内では一つのループの処理を一つのスレッドに割り当て，繰り返しの処理を並列に処理する．また，これらのスレッドの実行結果の総和はリダクションという方法で計算した．

4.1　実行結果

CPU と GPU の速度を比較するために，CPU で評価計算を行う関数も作成した．GPU で評価計算を行うときと，評価計算以外の部分は全て共通である．これらでの Rastrigin 関数の計算時間を計測し，比較した．ここで用いた CPU，GPU を含むマシンの構成を表 1 に示す．その結果を以下図 3 および図 4 に示す．図 3 では個体数を 400，x_iを 10bit の数として次元数を変化させながら，100 世代分計算を繰り返した．左から次元数 10，50，100 の場合の計算時間である．その結果，CPU では世代数の増加にほぼ比例して計算時間が増加することが確認できた．それに対して，GPU は世代数の増加と比例せず，計算時間は微量に増加していることが確認できた．

Table1　マシンの構成

OS	Ubuntu11.04
Memory	8 GB
CPU	Intel Core i5-2400(3.10GHz)
GPU	NVIDIA GeForce GTX460

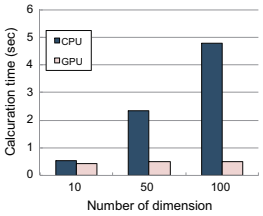

Fig.3　GPU と CPU の Rastrigin 関数の速度比較 1

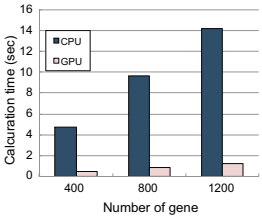

Fig.4　GPU と CPU の Rastrigin 関数の速度比較 2

次に，図 4 では次元数を 100，x_iを 10bit の数として個体数を変化させながら，100 世代分計算を繰り返した．左から個体数 400，800，1200 の場合の計算時間である．その結果 CPU は図 3 での結果と同様に，個体数の増加に比例して計算時間が増加していることが確認できた．ところが，GPU も個体数の増加に比例して計算時間が増加した．これは図 3 とは異なる傾向である．これらの結果より，CPU に比べて GPU の方が計算速度が高いことが確認できた．また，GPU の計算時間はスレッドを増加させた場合よりブロック数を増加させた場合の方が計算時間が大きくなっており，ブロックの処理の方がスレッドの処理に比べて計算時間が大きいことがわかる．これは，スレッドの処理はシェアードメモリを用いることで高速に処理できることが原因と考えられる．

5　まとめと今後の展望

本稿では，特別な知識を持たない GA 開発者にも並列処理を利用して評価計算の処理を高速化することができるようなフレームワークを提案した．提案したフレームワークは関数として呼び出され，その際，引数に遺伝子を保持するリストを渡し，戻り値に評価値が格納されたリストを得る．このフレームワークを自作の GA プログラムで利用した．その結果評価計算部の処理速度が向上したことが確認できた．今後の展望としては，より大規模な GA に向けたスレッド数の制限を超えても利用可能なフレームワークの構築が必要と考えられる．その際には，計算の切り分け方や切り分けた計算のスレッドへの割り当て方等を工夫する必要があると考えられる．

参考文献

1) 湯川英宜, 平野敏行, 西村康幸, 佐藤文俊．GPU によるタンパク質高精度静電ポテンシャルの計算の高速化．生産研究, Vol. 2009, pp. 27–34, 2009.
2) 大磯正嗣, 松村嘉之, 保田俊行, 大倉和博．CUDA 環境におけるデータ並列化を用いた遺伝的アルゴリズムの実装手法．知能と情報（日本知能情報ファジイ学会誌）, Vol. 2011, pp. 18–28, 2011.
3) 薫田匡史, 大森博司, 河村拓昌．大型望遠鏡を支持するトラス構造物の多目的最適設計．日本建築学会大会学術講演梗概集, Vol. 2008, pp. 917–918, 2008.
4) 中田秀基, 中島直敏, 小野功, 松岡聡, 関口智嗣, 小野典彦, 楯真一．情報処理学会研究報告．[ハイパフォーマンスコンピューティング], No. 29, pp. 155–160.