7M-04

GPUを用いた分散粒子群最適化法の一提案

高畠　和輝†　　佐藤　裕二‡
法政大学情報科学部†‡

1. まえがき

粒子群最適化法(Particle Swarm Optimization，以下PSO)は鳥や魚などの群れの振る舞いに着想を得て，1995年にKennedy，Eberhartらによって提案された最適化手法である[1]．PSOは遺伝的アルゴリズムや他の集団ベースのアルゴリズムに比べ実装の容易さ，収束・大域探索能力の維持という点で優れており，近年最適化問題を解くための効果的な手法として広く用いられている．その反面，群れに多数の個体数が必要である場合や，問題の探索空間が高次元である場合等では，膨大な数の評価を逐次的に行うために実行に非常に時間がかかる問題が存在する．

一方，近年ではGPUの演算の柔軟性の向上から，GPUの高い演算能力・並列処理能力を画像処理以外の汎用計算に利用する試みが活発に行われてきた．PSOアルゴリズムは独立な操作が多く，並列処理に向いていることから，GPUを用いたPSOアルゴリズムの実装に関する研究は存在するものの[2]，[3]，十分な実行速度の報告はされていない．

本稿では，特にNVIDIA社のCUDAアーキテクチャを活用して[4]，最適解探索の精度を保ちつつ速度を更に向上させるための実装法を提案し，代表的なテスト問題を用いてその有効性を示す．

2. GPUを用いたPSOの実装

2.1. 並列化構造

NVIDIA社が開発するGPUには，演算器の最小単位であるStreaming Processor (SP)の集合からなるStreaming Multiprocessor (SM)が複数存在する．既存の研究ではPSOアルゴリズムにおける互いに独立な各処理を複数に分割し，各SM内のSPに割り当てることでアルゴリズムを並列に実行する．またアルゴリズムの実行をCPUで管理することが多いため，特に並列度の低い処理に関してはCPU上で実行する場合もある．

2.2. データの格納

図1に従来手法におけるデータの格納方法を示す．NVIDIA社のGPUは独自のメモリ階層によって構成され，各メモリはアクセス速度，アクセス権，容量などの特性がそれぞれ異なる．GPUを利用した並列処理では，演算に用いるデータをどのメモリに格納するかが実行時間に大きく影響する．

既存の研究ではGPU内部に存在する高速なメモリの容量が不十分であるという問題もあり，PSOアルゴリズムに用いる各データはGPU外部の低速なGlobal Memoryに格納されている．

GPU-based Distributed Particle Swarm Optimization
†KAZUKI TAKABATAKE ‡YUJI SATO
†‡Faculty of Computer and Information Science, Hosei University

また基本的にはデータをGlobal Memoryに格納し，演算時にのみSM内部の高速に通信が可能であるShared Memoryに格納する手法も広く用いられている．

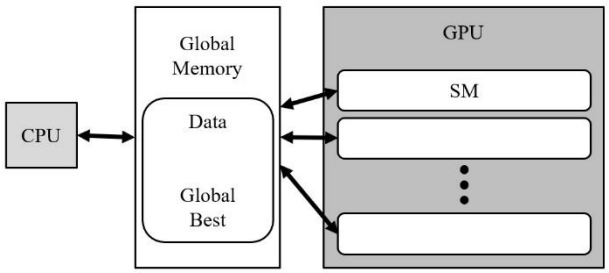

図1. 従来手法におけるデータの格納方法

3. 低速なメモリの使用を避ける実装方法

3.1. 群れの分割

提案する手法では，PSOアルゴリズムにおける個体群を複数の新たなPSOアルゴリズムの個体群に分割し，それぞれをGPU内の各SMに対応付ける．

図2に提案手法におけるデータの格納方法を示す．既存の研究ではデータをGlobal Memoryに格納するため，PSOの各操作において用いるデータをGlobal Memoryと通信する際に実行に非常に時間がかかる問題がある．

一方，近年の急速なGPUの発展に伴い，GPU内部の高速なメモリの容量は大幅に増加している．そこで提案手法では，分割された群れに関する各データを対応するSM内部に存在するShared Memoryやregisterに格納する．これにより分割された群れに対するアルゴリズムの操作を各SM内で完結させることで，Global Memoryとの低速な通信の大部分を回避することが可能になる．

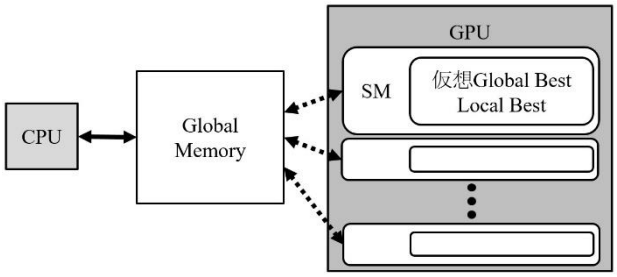

図2. 提案手法におけるデータの格納方法

3.2. 仮想Global Bestを用いた更新頻度の削減

従来の手法ではPSOアルゴリズムとしての群れは単一であり，「群れ全体における最適解」であるGlobal Bestを各SMが共有する必要がある．図1に示すように，異なるSM間においてGlobal Bestを共有するためにはGlobal Memoryを介しなければならないため，Global Bestの更新に時間がかかるという問題が存在する．

図2に示すように，提案手法では初期世代の Global Best をアルゴリズムの開始時に各 SM 内の Shared Memory に仮想 Global Best として格納するが，それとは別に新たに「分割された群れにおける最適解」を Local Best として各 SM に定義し，Shared Memory へ格納する．粒子の更新の際には各 SM 内で仮想 Global Best と Local Best を比較し，より評価の高い方を利用する．更に，各 SM 内で Local Best が仮想 Global Best を上回った際に，その Local Best を新たな全体の Global Best として他の SM が持つ仮想 Global Best を更新する．

これにより複数の SM を跨いだ仮想 Global Best の同期・更新を最小限にし，Global Memory との通信量を更に削減する．

3.3. 粒子速度更新式の変更

従来の PSO[1]の粒子速度更新式は次式で与えられる．

$$v(t+1) = v(t) + c_1 * rand_1 * \left(P_{bestx} - x(t)\right) + c_2 * rand_2 * \left(G_{bestx} - x(t)\right)$$

ここで $v(t+1), v(t)$ はそれぞれ世代 $t+1, t$ における各粒子の速度ベクトルであり，$x(t)$ は世代 t における各粒子の位置ベクトル，P_{bestx} は各粒子が探索した内で最も評価の高い位置ベクトル，G_{bestx} は Global Best を表す．また c_1, c_2 は正の定数であり，$rand_1, rand_2$ はそれぞれ異なる [0, 1] の乱数である．

提案手法では分割された各群れの持つ粒子数が少なくなり，従来の手法に比べて局所解に陥りやすくなる．そこで各群れの収束速度を抑え，全体として効率よく最適解を探索するために，更新式を既存研究でも広く用いられている Standard PSO[5]のものに変更する．3.2. において述べた Local Best を L_{bestx} と表記すると，変更後の更新式は次式で与えられる．

$$v(t+1) = \chi\left(v(t) + c_1 * rand_1 * \left(P_{bestx} - x(t)\right) + c_2 * rand_2 * \left(L_{bestx} - x(t)\right)\right)$$

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \varphi = c_1 + c_2$$

4. 評価実験

4.1. 実験方法

評価実験として，5 つのベンチマーク関数に対して CPU 上で Standard PSO と提案する手法を実行し，結果を比較する．また GPU を用いて提案する手法を実装した場合に，用いない場合と比較してどの程度速度が向上するかを検証する．ただし，粒子数 6000，Sphere 関数における次元数 100，その他の関数における次元数 50，世代数 2000 として実験を行い，50 回の試行平均をとる．

4.2. 実験結果

表 1 は，Standard PSO と提案手法を CPU 上で実行した際の実行時間と Global Best の値である．表 1 が示すようにほぼ全ての関数において提案手法は従来の PSO よりも早く，良い Global Best の値を示した．

表 1. Standard PSO と提案手法の比較

Function	time (ms)		Gbest Value	
	SPSO	提案法	SPSO	提案法
Sphere	14136.06	13446.14	3.78E-02	1.20E-20
Rosenbrock	7505.12	7024.32	2.10E+01	4.16E+01
Rastrigin	16174.52	13641.76	1.08E+02	6.17E+01
Griewank	14122.60	16979.12	3.62E-10	1.20E-32
Ackley	11991.04	9480.18	5.07E-05	3.89E-06

表 2. GPU を用いた場合の速度向上比

Function	time (ms)		CPU time / GPU time
	CPU	GPU	
Sphere	13446.14	691.75	19.44
Rosenbrock	7024.32	271.92	25.83
Rastrigin	13641.76	340.80	40.03
Griewank	16979.12	379.74	44.71
Ackley	9480.18	334.03	27.35

表 2 は，提案手法において GPU を用いた場合と用いない場合の速度比較結果である．表 2 より，全ての関数において GPU を用いた提案手法は約 20 倍以上の実行速度，特に Griewank 関数においては 44.71 倍の実行速度を示していることがわかる．

この実験結果より，提案する手法によって探索の精度が向上し，更にアルゴリズムの実行時間が大幅に短縮されたことが確認できた．

5. むすび

本論文では GPU を用いて PSO アルゴリズムを並列に実行する際に，群れを分割し，低速なメモリとの通信頻度を下げることで実行速度を大幅に短縮する実装法を提案した．また評価実験により，提案手法の適用が解探索の精度維持，実行速度向上に十分な効果を発揮することが確認できた．

多峰性問題に代表される解探索が難しい関数に対するアルゴリズムの探索精度の改善，GPU を用いたより高速な実装は今後の課題である．

文献

[1] J. Kennedy and R. C. Eberhart: *Particle Swarm Optimization*, IEEE International Conference on Neural Networks, Vol.4, pp.1942-1948(1995).

[2] Y. Zhou and Y. Tan: *GPU-based Parallel Particle Swarm Optimization*, 11th IEEE Congress on Evolutionary Computation, pp. 1493-1500 (2009)

[3] Md. M Hussain, Hiroshi Hattori and Noriyuki Fujimoto: *A CUDA Implementation of the Standard Particle Swarm Optimization*, 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing.

[4] NVIDIA: CUDA Toolkit Documentation https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[5] D. Bratton and J. Kennedy: Defining a Standard for Particle Swarm Optimization, IEEE Swarm Intelligence Symposium, pp.120-127 (2007)