B-102

島モデルに基づく差分進化の性能評価
Experimental Study of Island-based Differential Evolution

中島 健一†　田川 聖治‡
Nakajima Kenichi　Tagawa Kiyoharu

1. はじめに

差分進化（DE：Differential Evolution）[1]は、決定変数が実数値をとる単目的の関数最適化問題を対象とした進化計算の一種である。DE のアルゴリズムは単純であるが、微分不可能な多峰性の関数最適化問題に対しても良好な解が求められる。このため、多くの現実的な最適化問題において DE の適用例が報告されている。解（個体）の集団による確率的な多点探索法である進化計算は、そのアルゴリズム自体に並列性が内在する。このため、遺伝的アルゴリズム（GA：Genetic Algorithm）を含む多くの進化計算では、幾つかの並列化手法が提案されている[2]。

近年、複数のプログラムを同時に実行できるマルチスレッド・プロセッサや、1 つのチップ上に複数のコア（プロセッサ）を集積したマルチコア CPU が増えている。さらに、Graphics Processing Unit（GPU）では数百のコアによるプログラムの並列処理が可能である。マルチコア CPU や GPU の普及は、進化計算の並列化に新たなパラダイムをもたらした。DE でも、従来の PC クラスタやネットワーク PC のほか[3]、GPU による並列化手法が報告されている[4]。著者らも、マルチコア CPU を利用した CDE（Concurrent DE）を提案している[5]。マルチコア CPU を有効に活用して、進化計算のようなアプリケーションを高速化するためには、その並行プログラムを実装する必要がある。並行プログラムは複数のスレッドから構成され、それらのスレッドは複数のコアで並列に実行される。ただし、スレッドのコアへの割り当ては、並行プログラム自体ではなく、オペレーティング・システムが担当する。このため、並行プログラムにおいて CPU の構造を考える必要はない。GPU を対象とした DE の並列プログラムと比較し、マルチコア CPU を利用する DE の並行プログラムは、並列化の規模は小さいが、ハードウェアを考慮した専用の開発ツールを必要とせず、既存の DE の実行時間を着実に短縮できる。また、DE の並行プログラムは、汎用性や移植性の面でも優れている。

本稿では、島モデルと呼ばれるサブ集団（島）に基づく差分進化（IbDE：Island-based Differential Evolution）を提案する。IbDE では集団を複数の島に分け、島ごとに独立に DE を実行する。また、島の間で個体を移動させる従来の島モデルの移民と異なり、優れた個体の形質のみを交換するパンミクシー（任意交叉）と呼ぶ手法を採用している。さらに、IbDE は並行プログラムとして実装し、マルチコア CPU を搭載した PC で実行することができる。最後に、数値実験と統計的検定により、従来の CDE と比較して、IbDE は実行時間と解の精度で勝ることを示す。

2. Concurrent Differential Evolution（CDE）

DE は主集団 P と副集団 Q を持ち、主集団 P から副集団 Q を生成した後、主集団 P を副集団 Q で上書きする。著者らは、DE の世代交代モデルを改良した DER を提案している[6]。DER では、主集団 P から副集団 Q を生成した後、副集団 Q から主集団 P を生成する。このため、集団の上書きに要する計算時間を節約できる。本稿では、マルチコア CPU を対象とした DER の並行プログラムを CDE と呼ぶ。CDE は、1 つのメイン・スレッドと N_T 個（$N_T \geqq 1$）のワーカー・スレッドから構成される。主集団 P と副集団 Q は、それぞれ以下のように N_T 個のサブ集団 P_n と Q_n に分割されて共有メモリに保存される。

$$\begin{cases} \boldsymbol{P} = \boldsymbol{P}_1 \cup \cdots \cup \boldsymbol{P}_n \cup \cdots \cup \boldsymbol{P}_{N_T} \\ \boldsymbol{Q} = \boldsymbol{Q}_1 \cup \cdots \cup \boldsymbol{Q}_n \cup \cdots \cup \boldsymbol{Q}_{N_T} \end{cases} \quad (1)$$

CDE のメイン・スレッドは初期集団 P をランダムに生成した後、N_T 個のワーカー・スレッドを同時に起動し、それらの処理が終わるまで待つ。n 番目のワーカー・スレッドは他ワーカー・スレッドと同期して DER を実行する。すなわち、集団 P からサブ集団 Q_n を生成した後、集団 Q からサブ集団 P_n を生成する。以下に CDE メイン・スレッドとワーカー・スレッドの手順を示す。

【メイン・スレッドの手順】
手順 1：初期集団 P をランダムに生成する。
手順 2：N_T 個のワーカー・スレッドを起動する。
手順 3：全てのワーカー・スレッドの終了を待つ。
手順 4：最良の個体 $\vec{x}_b \in \boldsymbol{P}$ を出力して終了する。

【ワーカー・スレッドの手順】
手順 1：全個体 $\vec{x}_i \in \boldsymbol{P}$ の $f(\vec{x}_i)$ を計算し、g=0 とする。
手順 2：集団 P からサブ集団 Q_n を生成する。
手順 3：すべてのワーカー・スレッドの到着を待つ。
手順 4：集団 Q からサブ集団 P_n を生成する。
手順 5：すべてのワーカー・スレッドの到着を待つ。
手順 6：$g < G_M$ なら、g=g+2 として手順 2 に戻る。
そうでなければ終了する。

上記の n 番目のワーカー・スレッドは、サブ集団 $P_n \subseteq P$ と $Q_n \subseteq Q$ に含まれる個体の書き込みと読み出しを独占する。また、すべてのワーカー・スレッドが完全に揃って実行されるなら、複数のワーカー・スレッドによる集団 P と Q からの個体の読み出しと、n 番目のワーカー・スレッドによるサブ集団 P_n と Q_n への書き込みに排他制御は不要である。しかし、スレッドのコアへの割り当てはオペレーティング・システムが担当する。さらに、スレッド数 N_T が CPU のコア数より多い場合も考慮する必要がある。そこで CDE のワーカー・スレッドの手順 3 と手順 5 では全ワーカー・スレッドを待ち合わせる。このため、並行プログラムの実装では「Cyclic-Barrier」[7]によるスレッドの同期制御を採用する。ここで、P から Q_n を生成する過程を「往路」、Q から P_n を生成する工程を「復路」と呼ぶ。CDE では往路と復路が交互に実行される。集団 P と集団 Q はサブ集団に分割されているが、トライアル・ベクトルの生成には集団内の任意の個体を利用でき、CDE の集団は島モデルのように構造化されていない。

† 近畿大学総合理工学研究科, Graduate school of Science and Engineering Research , Kinki University

‡ 近畿大学理工学部, School of Science and Engineering, Kinki University

3. Island-based Differential Evolution（IbDE）

3.1. IbDE の概要

マルチコア CPU を対象とした IbDE の並行プログラムについて説明する。IbDE は、1 つのメイン・スレッドと N_T 個（N_T≧1)のワーカー・スレッドから構成される。主集団 P と副集団 Q は、それぞれ N_T 個のサブ集団に分割されて共有メモリに保存される。ここで、サブ集団 P_n と Q_n の組を「島」と呼ぶ。メイン・スレッドでは N_T 個のワーカー・スレッドを同時に起動し、それらの処理が終わるまで待つ。各ワーカー・スレッドは担当する島（P_n と Q_n）において独立に DER を実行する。以下に IbDE のメイン・スレッドの手順を示す。

【メイン・スレッドの手順】
手順1：N_T個のワーカー・スレッドを起動する。
手順2：全てのワーカー・スレッドの終了を待つ。
手順3：最良の個体 $\vec{x}_b \in P$ を出力して終了する。

3.2. IbDE のワーカー・スレッド

前述のとおり、IbDEのn番目のワーカー・スレッドは担当する島（P_nとQ_n）において独立にDERを実行する。ただし指定された世代間隔N_Cでパンミクシーが行われる。すなわち、n番目のワーカー・スレッドは、Q_nからP_nを生成する。一方、パンミクシーでは全てのワーカー・スレッドが任意の個体 $\vec{x}_i \in Q$ を読み出すことができ、n番目のワーカー・スレッドは、QからP_nを生成する。以下にn番目のワーカー・スレッドの手順を示す。

【ワーカー・スレッドの手順】
手順1： 初期集団P_nをランダムに生成する。
手順2：全個体 $\vec{x}_i \in P_n$ の $f(\vec{x}_i)$ を計算しg=0とする。
手順3：カウンタをt=N_Cとする。
手順4：サブ集団からサブ集団を生成する。
手順5：g<tなら、手順10に進む。
手順6：すべてのワーカー・スレッドの到着を待つ。
手順7：集団Qからサブ集団P_nを生成する。
手順8：すべてのワーカー・スレッドの到着を待つ。
手順9：カウンタをt=t+N_Cとし、手順11に進む。
手順10：サブ集団Q_nからサブ集団P_nを生成する。
手順11：g<G_Mなら、*g*=g+2として手順4に戻る。
そうでなければ終了する。

3.3. パンミクシー頻度

本稿では、IbDEのパンミクシー頻度N_Cは試行錯誤により決めるのではなく、サブ集団のサイズN_Sに基づき算出する。まず、DEの戦略では、各ターゲット・ベクトルに対して、それとは異なる 3 つの個体をサブ集団から非復元抽出する。このため、あるターゲット・ベクトルに対して、同じ変異ベクトルが作られる確率は式(2)となる。

$$\delta = \frac{1}{(N_s-1)(N_s-2)(N_s-3)} \tag{2}$$

世代数N_Cの期間に、各ターゲット・ベクトルに対して作られる同じ変異ベクトル数の期待値kは式(3)となる。

$$k = N_s N_c \delta \tag{3}$$

ここで、同じ変異ベクトル数の期待値kが与えられると、パンミクシー頻度N_Cは式(4)のように決まる。

$$Nc = \frac{k(Ns-1)(Ns-2)(Ns-3)}{Ns} \tag{4}$$

IbDEの実装では、パンミクシーの前後でワーカー・スレッドの同期が必要であるが、各ワーカー・スレッドは大半の世代では非同期に実行される。このためスレッドの同期制御に要するオーバーヘッドは非常に小さい。また、パンミクシーは島の間で個体を移動させるのではなく、優れた個体の形質のみをトライアル・ベクトルを介して伝播させる。このため、ある個体のコピーが複数の島に重複して存在することがなく、集団内の個体の多様性が保持される。さらに、IbDEの島モデルのトポロジーは完全結合であり、形質の伝播方向に制約がない。

4. 数値実験

4.1. テスト問題

以下の 10 種類のテスト問題 f_p を使用した。

- Sphere Function : f_1
- S_Sphere Function : f_2
- Schwefel Function : f_3
- S_Schwefel Function : f_4
- Rosenbrock Function : f_5
- Rastrigin Function : f_6
- S_Rastrigin Function : f_7
- SR_Rastrigin Function : f_8
- Ackley Function : f_9
- Griewank Function : f_{10}

ただし、テスト問題の次元はすべて D=30 とする

4.2. 実験方法

先に報告した CDE、SIbDE と本稿で提案した IbDE の性能を比較する。ここで、SIbDE はパンミクシーを行わない IbDE である。まず全てのこれらの DE の並行プログラムは、Java 言語で実装した。また、オペレーティング・システムには Microsoft 社の Windows XP を使用した。さらに、マルチコア CPU には同時に 2 つのスレッドを実行できるマルチスレッド・プロセッサを 4 個集積し、最大 8 個のスレッドを並列処理できる Intel®Core™i7(@3.33[GHz])を用いた。すべての DE において、個体数 N_P=96、スケール係数 F=0.5、交叉率 C_R=0.9、世代数 G_M=2000 として、ワーカー・スレッド数は N_T=4、8、12、16 とした。このとき N_P=$N_S$$N_T$ よりの関係より SIbDE と IbDE の島のサイズは N_S=48、24、16、12 となる。ここで、すべての DE を各テスト問題 f_pに 30 回ずつ適用した。

4.3. 実験結果

各テスト問題 f_pにおける DE の並行プログラムの実行時間の実行時間の平均を図 1 に示す。縦軸が実行時間[ms]であり、横軸がワーカー・スレッド数 N_Tである。図 1 よりスレッド数が 8 以上になると速度の減少が緩やかになり、テスト問題によっては速度の増加が始まる。また、その場合、スレッド数 16 になると再び緩やかな速度減少が始まっている。ワーカー・スレッドを非同期に実行する SIbDE と IbDE は CDE より速く、ワーカー・スレッドが完全に独立して動く SIbDE は IbDE よりも実行時間は短い。

各 DE の並行プログラムより得られた解の質を目的関数値で比較した結果を表 1 に示す。表 1 において○は 50 回平均で最小の目的関数値の値、および Wilcoxon 検定によ

る比較の結果、最小値と統計学的に有意な差がない目的関数値が得られたことを示す。表 1 から、5 種類のテスト問題で IbDE は CDE に勝り、4 種類のテスト問題で CDE と IbDE に有意な差はない。すなわち、f_5 を除いて IbDE が他の DE に劣るということはなかった。

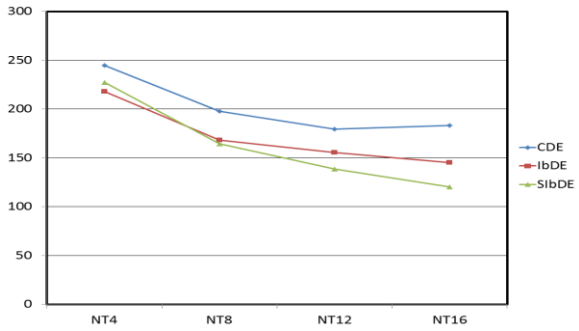

図 1 (a)　Sphere Function : f_1

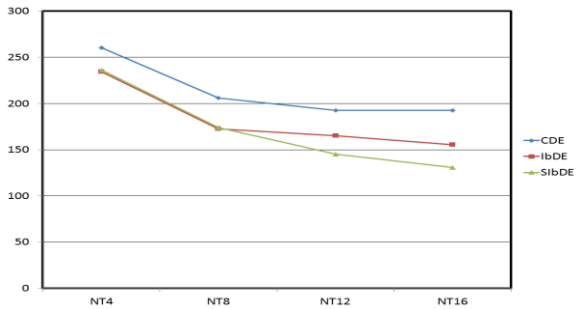

図 1 (b)　S_Sphere Function : f_2

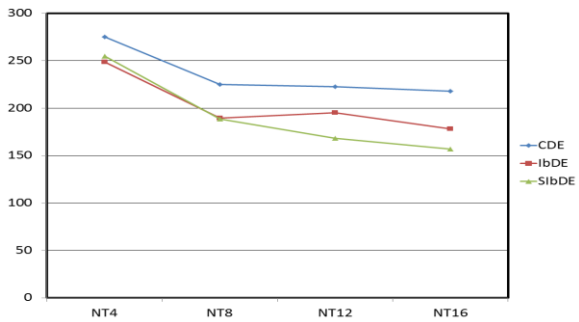

図 1 (c)　Schwefel Function : f_3

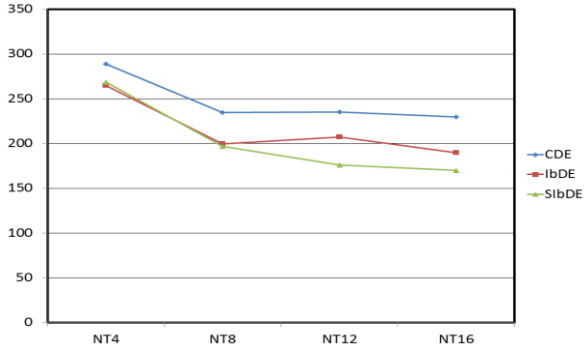

図 1 (d)　S_Schwefel Function : f_4

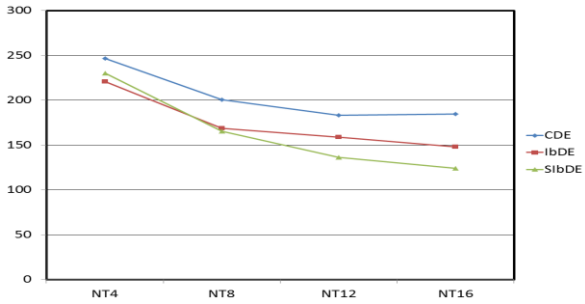

図 1 (e)　Rosenbrock Function : f_5

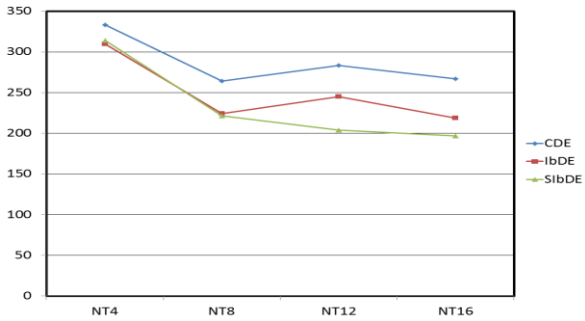

図 1 (f)　Rastrigin Function : f_6

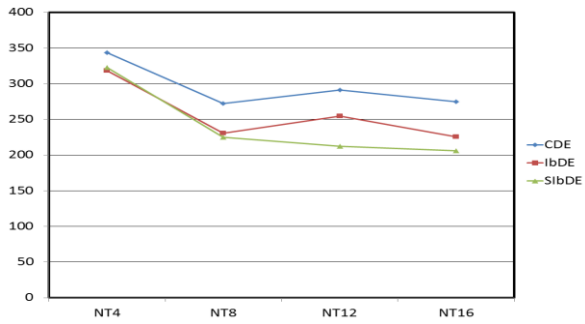

図 1 (g)　S_Rastrigin Function : f_7

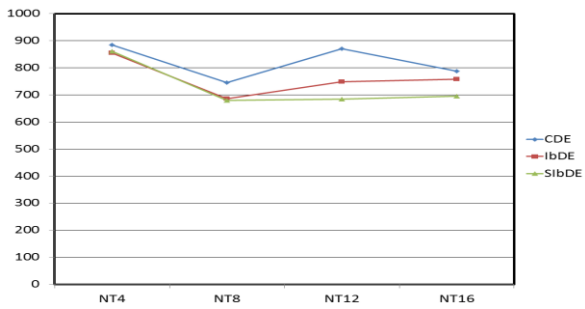

図 1 (h)　SR_Rastrigin Function : f_8

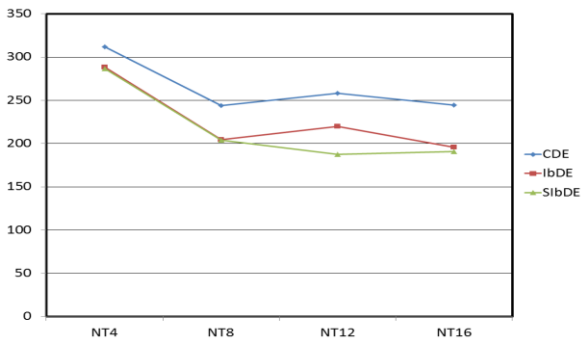

図 1 (i)　Ackley Function : f_9

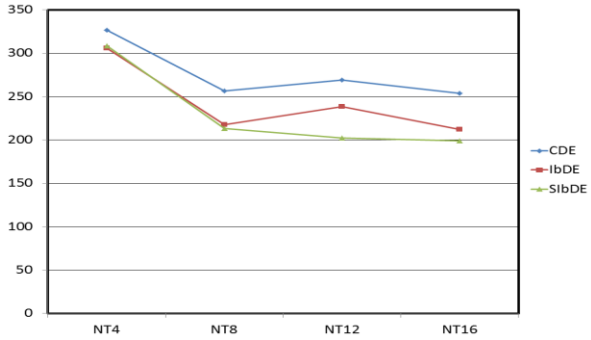

図 1 (j)　Griewank Function : f_{10}

表 1 (a)　Sphere Function : f_1

	NT4	NT8	NT12	NT16
CDEBR	○	○	○	○
IbDE	○	○	○	○
SIbDE	○	○	○	

表 1 (b)　S_Sphere Function : f_2

	NT4	NT8	NT12	NT16
CDEBR	○	○	○	○
IbDE	○	○	○	○
SIbDE	○	○		

表 1 (c)　Schwefel Function : f_3

	NT4	NT8	NT12	NT16
CDEBR				
IbDE				○
SIbDE				

表 1 (d)　S_Schwefel Function : f_4

	NT4	NT8	NT12	NT16
CDEBR				
IbDE				○
SIbDE				

表 1 (e)　Rosenbrock Function : f_5

	NT4	NT8	NT12	NT16
CDEBR	○	○	○	○
IbDE				
SIbDE				

表 1 (f)　Rastrigin Function : f_6

	NT4	NT8	NT12	NT16
CDEBR				
IbDE				○
SIbDE				

表 1 (g)　S_Rastrigin Function : f_7

	NT4	NT8	NT12	NT16
CDEBR				
IbDE				○
SIbDE				

表 1 (h)　SR_Rastrigin Function : f_8

	NT4	NT8	NT12	NT16
CDEBR	○	○	○	○
IbDE	○	○	○	○
SIbDE	○			

表 1 (i)　Ackley Function : f_9

	NT4	NT8	NT12	NT16
CDEBR				
IbDE		○	○	○
SIbDE		○		

表 1 (j)　Griewank Function : f_{10}

	NT4	NT8	NT12	NT16
CDEBR	○	○	○	○
IbDE	○	○	○	○
SIbDE	○	○		

5. おわりに

本稿では、パンミクシーの頻度を一意に決定する方法を提案し、島モデルに基づく DE の並行プログラムである IbDE の性能を評価した。これにより、従来の CDE と比較して IbDE は実行時間と得られる解の質で勝ることを示した。今後の課題は、より質の高い解を求めるため、パンミクシー頻度を決定する方法を改良することである。

参考文献

[1] R. Storn and K. Price: “Differential evolution – a simple and efficient heuristic for global optimization over continuous space”, Journal of Global Optimization, Vol. 11, No. 4, pp.341-359（1997）

[2] E. Alba and M. Tomassini: “Parallelism and evolutionary algorithms”, IEEE Trans. on Evolutionary Computation, Vol. 6, No. 4, pp. 443-462（2002）

[3] D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis: “Parallel differential evolution”, Proc. of IEEE Congress on Evolutionary Computaion, pp. 2023-2029（2004）

[4] L. de Veronses and R. Krohling: “Differential evolution algorithm on the GPU with C-CUDA”, Proc. of IEEE Congress on Evolutionary Computaion, pp. 1-7（2010）

[5] K. Tagawa: “Concurrent differential evolution base on generational model for multi-core CPUs”, Proc. of 9th International Conference, Simulated Evolution and Learning, LNCS7673, Springer, pp. 12-21（2012）

[6] K. Tagawa and K. Nakajima: “Island-based differential evolution with panmictic migration for multi-core CPUs”, Proc. of IEEE Congress on Evolutionary Computation, pp. 852-859 (2013)

[7] B. Goetz, T. Peierls, J. Bloch, J. Bowbeer, D. Holmes, and D. Lea: Java Concurrency in Practice, Addison-Wesley（2006）