

# Cell B.E. による SIMD-oriented Fast Mersenne Twister を用いた Particle Swarm Optimization

五十嵐 潤 園尾 聡 古賀 崇了

九州工業大学大学院 生命体工学研究科 脳情報専攻

最適化問題において、群知能に着想を得た Particle Swarm Optimization (PSO) と呼ばれる手法の有効性が多数報告されている。PSO では、差分方程式による繰り返し計算を行い、パラメータ探索に膨大な計算が必要となる。PSO の一連の計算の中で、特に計算コストが高いのは、擬似乱数生成である。我々はこの点に着目し、擬似乱数生成に、Cell の高い SIMD 演算性能を用いることで、高速に計算できないかと考えた。そこで、高速かつ質の良い擬似乱数を生成することで知られる、SIMD-oriented Fast Mersenne Twister (SFMT) を Cell で実行し、PSO の計算を行った。その結果、汎用 CPU で通常の擬似乱数生成法を用いるのに比べて、14-42 倍の高速な計算を実行することができた。

## 1 はじめに

最適化問題とは、制約条件を満たした上で目的関数を最小（もしくは最大）にするパラメータを探索する問題である。工学的な実応用における最適化問題は、その解空間が非線形・非凸・多峰性関数などの複雑な目的関数で表されることが多い。近年、そのような最適化問題を解決するものとして、Particle Swarm Optimization (PSO) と呼ばれる最適化手法が注目されている [1]。PSO は、昆虫や動物の群行動・人間の社会生活などの集団行動に着想を得た最適化手法である。そのアルゴリズムは簡素に記述できる上に、様々な最適問題に対して有効であるという特長を有する。

PSO は優れた最適化手法であるが、大規模で複雑な問題を PSO で探索するには、膨大な計算が必要になる。PSO の計算過程で、特に計算コストが高いものとして、多用される擬似乱数生成が挙げられる。もし、PSO における擬似乱数生成に、Cell B.E. の高い演算性能を活用できるならば、計算の高速化が期待できる。

近年、高速で質が良い擬似乱数を生成するアルゴリズムとして、SIMD-oriented Fast Mersenne Twister (SFMT)[3] が提案されている。SFMT は多段パイプライン処理や SIMD 演算などの並列演算処理を考慮した設計であり、さらに生成される擬似乱数は「長周期を有する」、「良好な均等分布特性が証明されている」などの非常に優れた性質を持っている。本稿では、SFMT を擬似乱数生成法として用いて Cell B.E. に PSO を実装することで、非常に高速な計算を行えることを示す。

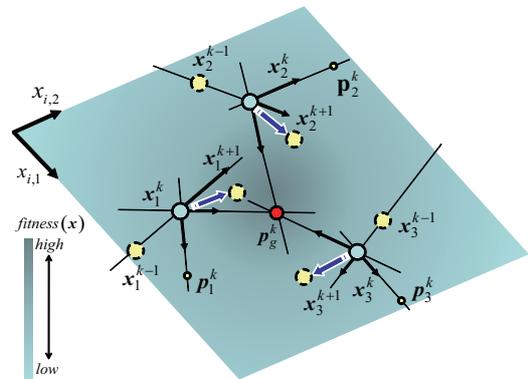


図1 PSO による探索過程の概略

## 2 方法

### 2.1 PSO のアルゴリズム

PSO では、群れ状の多数の探索点（パーティクル）が互いに情報を共有しながら解空間における探索を進める。図1に PSO の計算過程の概略を示す。

各探索点は、位置ベクトル  $x_i^k$ 、速度ベクトル  $v_i^k$ 、各探索点の過去の最良値  $E_i^k$  とその時の位置ベクトル  $p_i^k$  を保持している。パーティクル集合全体（群れ）では過去の最良値  $E_g^k$  とその時の位置ベクトル  $p_g^k$  の情報を共有する。ここで  $i, k$  はそれぞれ各パーティクルの番号と計算ステップを表す。

以下に PSO の計算の処理手順を簡潔に示す。

1. 探索空間における各パーティクルの初期位置と初期速度をそれぞれ擬似乱数によって決定する。

- 現在の  $\mathbf{x}_i^k$  から、目的関数の値を計算する。
- 各パーティクルについて計算した目的関数の値を各パーティクルの最良値  $E_i^k$  と比較し、 $\mathbf{p}_i^k$  と  $\mathbf{x}_i^k$  を更新する。最後に全パーティクルで比較を行い、全体の最良値  $E_g^k$  を得る。
- 次時刻における各パーティクルの速度と位置を数式に従って計算する。

$$\begin{aligned} \mathbf{v}_i^{k+1} &= \gamma \mathbf{v}_i^k + c_1 \mathbf{U}(0, \phi_1) \otimes (\mathbf{p}_i^k - \mathbf{x}_i^k) \\ &\quad + c_2 \mathbf{U}(0, \phi_2) \otimes (\mathbf{p}_g^k - \mathbf{x}_i^k) \\ \mathbf{x}_i^{k+1} &= \mathbf{x}_i^k + \mathbf{v}_i^k \end{aligned}$$

ここで、 $\gamma, c_1, c_2$  は探索方針を決定するパラメータで、それぞれ 0.99, 0.9, 0.9 である。 $\mathbf{U}(0, \phi_i)$  は区間  $[0, \phi_i]$  で一様分布をなす擬似乱数を要素とするベクトルを表し、各計算時刻、各パーティクルについて計算される。

- 上記 2.~4. の手順を、規定回数繰り返す。

## 2.2 ベンチマーク関数

PSO の最適化問題のベンチマークとして、図 2 に示す De Jong のテスト関数群 [2] の最小値を求めた。本研究では、パーティクル数は 32 とし、F1-F3 は 100, F4-F5 では 500 回、差分方程式の計算を繰り返し、1 試行とした。全てのベンチマーク関数において、2400 回の試行を行い、計算に要した時間を測定した。

## 2.3 使用機器と計算設定

使用機器と計算環境について表 1 に示す。Cell B.E. の計算時間の測定には、主に SPU Decrementer を使い、50 秒以上かかる計算にのみ Linux シェルコマンド time を用いた。比較対象機では、C 言語標準ライブラリの clock 関数を用いた。両計算環境において、数値演算精度には単精度を用いた。

## 2.4 疑似乱数生成法 -SFMT と rand-

疑似乱数生成には、松本眞、斎藤睦夫が WEB ページ [4] で公開している SFMT バージョン 1.3.3 と、C 言語標準ライブラリの rand 関数を用いた。PowerPC Altivec 用の SFMT コードに変更を加え、Cell B.E. で実行できるようにした。Core 2 Duo で SFMT を実行する際は、SIMD

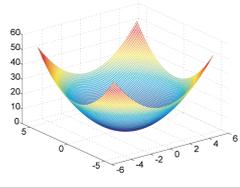
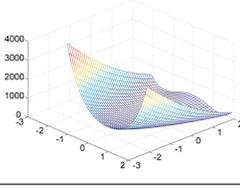
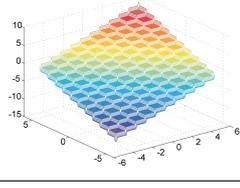
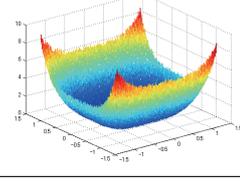
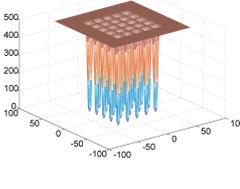
| 目的関数名と数式  | 目的関数の概形   |
|---|---|
| <b>F1: Sphere Model</b><br>$F_1(\mathbf{x}) = \sum_{j=1}^n x_j^2$ $-5.12 \leq x_j \leq 5.12$  |    |
| <b>F2: Generalized RosenBrock's Function</b><br>$F_2(\mathbf{x}) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$ $-5.12 \leq x_j \leq 5.12$                                 |    |
| <b>F3: Step Function</b><br>$F_3(\mathbf{x}) = \sum_{j=1}^5 [x_j]$ $-5.12 \leq x_j \leq 5.12$   |    |
| <b>F4: Quartic Function with Noise</b><br>$F_4(\mathbf{x}) = \sum_{j=1}^3 0x_j^4 + \text{Gauss}(0.1)$ $-1.28 \leq x_j \leq 1.28$                                |   |
| <b>F5: Shekel's Foxholes</b><br>$F_5(\mathbf{x}) = \frac{1}{500} + \sum_{k=1}^25 \frac{1}{k + \sum_{j=1}^{25} (x_j - a_{jk})^2}$ $-65.536 \leq x_j \leq 65.536$ |  |

図 2 DeJong のテスト関数群

演算を用いなかった。SFMT では、乱数を一括生成する `fill_array32` 関数を用いた。

図 3 で、PSO の探索の一例として、Dejong ベンチマーク関数 F1 で rand と SFMT を用いた場合の、400 試行の最良値の平均と標準偏差の推移を示す。rand と SFMT で、最良値の収束の速さと最終的な最良値に大きな違いはなかった。

## 3 結果

はじめに、疑似乱数生成に rand を用いて PSO の計算を行ったときの、Cell の性能を調べた。図 4 は、SPE 一基で、組み込み関数を用いて SIMD 演算を使用したとき、使用しないとき、および Core 2 の計算時間を示している。

表 1 使用機器と計算環境

|          | Cell B.E. 搭載機 : PlayStation3             | 比較対象機 : 汎用 PC                       |
|----------|--|-------------------------------------|
| CPU      | Cell B.E. 3.2GHz (PPE+6SPE)              | Core 2 Duo 2.66GHz (use 1-core)     |
| Memory   | XDR 256MB                                | DDR2 2GB                            |
| OS       | Fedora Core 6, (kernel2.6.18-53.1.4.el5) | CentOS 5, (kernel2.6.18-53.1.4.el5) |
| Compiler | Cell 用 GCC 4.1.1 (-O3)                   | Intel compiler ICC (-O3)            |
| Library  | Cell SDK 2.1, libspe2, SIMD Math Library |                                     |

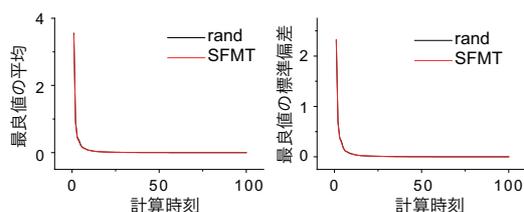


図 3 最良値の推移の例

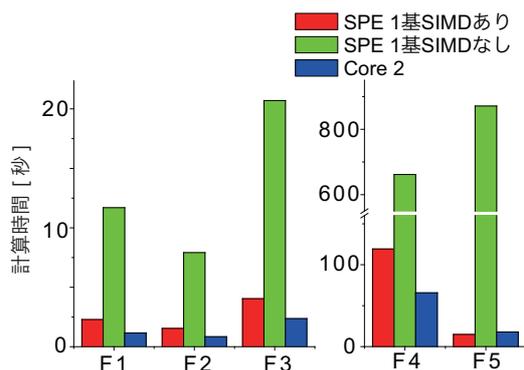


図 4 rand を用い SPE1 基を使用したときの計算時間

SIMD 演算を用いる場合は、4 試行をベクトルの 4 要素に割り当て並列に計算を行った。F1-F5 の全てのケースで、SIMD 演算を使用したとき、使用しないときに比べて、5 倍以上高速に計算された。ベクトル計算による高速化に加え、組み込み関数により、計算の効率が上昇したためであると考えられる。SPE1 基で SIMD 演算を使用したときは、Core 2 に比べ、約 2 倍の計算時間を要した。

次に SIMD 演算を使用し、SPE6 基で計算を行った (図 5)。SPE6 基を使用したとき、SPE1 基を使用したときに比べて、計算時間がほぼ 1/6 になった。これは、PSO の計算では SPE 間の通信の必要がないため、SPE の数が増えた分だけ、計算時間が短縮されるためである。F1-F5 の

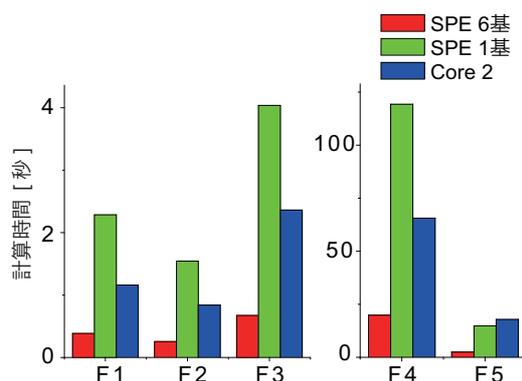


図 5 rand を用い SPE6 基を使用したときの計算時間

全てのケースで、SPE6 基を使用したときは、Core 2 に比べて、1/3-1/7 倍の計算時間であった。

SPE6 基で計算を行ったときの計算時間の内訳を図 5 に示す。PSO は、主要な計算部分として、位置の更新、評価値の算出、最良値の決定、速度の更新がある。図 5 の水色の速度の更新の部分には、主に rand により乱数生成を行う部分で、F1-F5 のテストにおいて、割合が非常に大きいことがわかる。この結果から、PSO における擬似乱数生成を rand から SFMT に変えることで、より高速な計算を行うことが期待される。

そこで、擬似乱数生成に SFMT を用いて、PSO の計算を行った。図 7 は Cell B.E. と汎用 Core 2 において、SFMT と rand を用いたときの計算結果を示している。Cell B.E. と汎用 CPU とともに、SFMT を用いることで、rand に比べ、計算時間は大幅に短縮された。F1-F5 の全てのケースにおいて、Cell B.E. で SFMT を用いて計算したときは、もっとも高速に計算が行われた。

表 2 は、図 7 で示される Cell B.E. と Core 2 の計算時間の比を求めたものである。Cell B.E. で SFMT を用いた

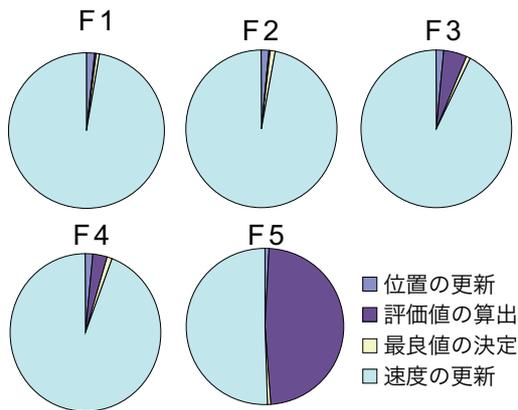


図6 rand を用いたときの計算時間の内訳

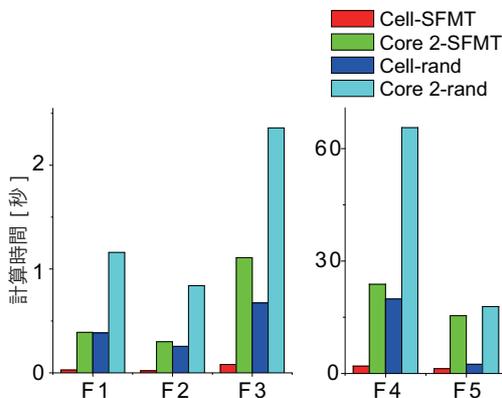


図7 Cell B.E. で SFMT を用いたときの計算時間

ときは、Core 2 で rand を用いたときに比べ、約 14-42 倍の速度になった (表 2, Core 2-rand/Cell-SFMT の行). 汎用 CPU で rand を用いた、通常の PSO の計算に比べ、Cell B.E. で SIMD 演算、マルチコア、SFMT を用いた計算は、非常に高速であることがわかる.

また、Cell B.E. と Core 2 において、擬似乱数を rand から SFMT へ変えた効果の違いを、表 2 からみることができる. Cell B.E. と Core 2 とともに rand を用いたとき、Cell B.E. の計算速度は、Core 2 に比べ約 3-7 倍であるのに対し (表 2, Core 2-rand / Cell-rand の行), Cell B.E., Core 2 とともに SFMT を用いたときは、約 12-15 倍となった (表 2, Core 2-SFMT / Cell SFMT の行). この結果から、Cell B.E. に SFMT を用いたときは、Core 2 に用いたときよりも、より高速化の効果が高いといえる. Cell B.E. の SIMD 演算の高い演算性能が、SFMT を用いることでより効果的に発揮されたといえる.

表 2 Cell B.E. と Core 2 における計算時間の比

|                       | F1   | F2   | F3   | F4   | F5   |
|-----------------------|------|------|------|------|------|
| Core 2-rand/Cell-SFMT | 41.4 | 42.0 | 29.1 | 33.5 | 13.9 |
| Core 2-SFMT/Cell-SFMT | 13.9 | 15.0 | 13.7 | 12.1 | 12.0 |
| Core-2-rand/Cell-SFMT | 3.0  | 3.3  | 3.5  | 3.3  | 7.2  |

## 4 議論

本研究では、Cell B.E. で SFMT を用いて PSO を実行することで、非常に高速に計算を実現できることを示した. Cell B.E. の SIMD 演算やマルチコアを用いた並列計算だけで、汎用 CPU よりも約 3-7 倍高速な計算を行うことが可能である. さらに、Cell B.E. の得意な SIMD 演算を活かせる SFMT を利用することで、汎用 CPU で rand を用いたときに比べ、約 14-42 倍の高速な計算を達成することができた.

今回行った Dejong のベンチマークテストでは、rand と SFMT を用いて PSO を実行したとき、最終的に得られる最良値の値に、違いはみられなかった. しかしながら、今回よりも複雑な高次元空間や、未知の空間を探索する場合、擬似乱数の性質の差が探索の効率を変える可能性がある. そのような場合、長周期かつ、高次元に均等分布の擬似乱数を生成する SFMT は、高い探索性能を発揮することが期待される.

## 参考文献

- [1] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *Proc. of IEEE the International Conference on Neural Networks*, pp.1942-1948, 1995.
- [2] KA. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," *PhD thesis*, University of Michigan, 1975.
- [3] M. Saito and M. Matsumoto, "SIMD-oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator," *Monte Carlo and Quasi-Monte Carlo Methods 2006*, Springer, pp.607-622, 2008.
- [4] <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index-jp.html>