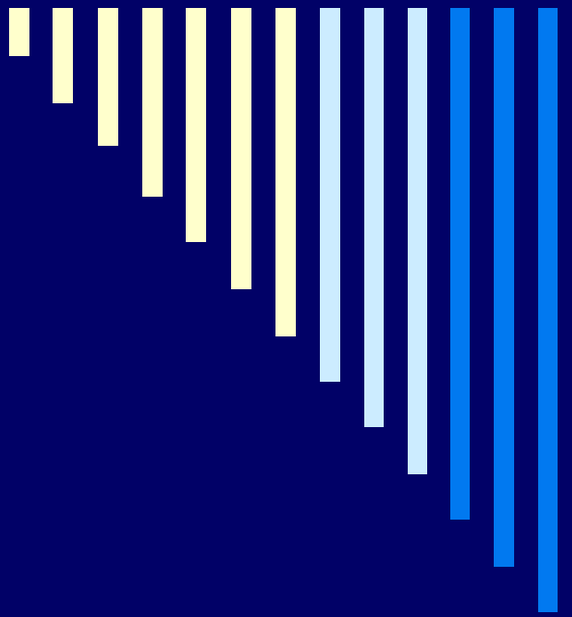


並列乱数の性能報告

核融合科学研究所
理論･シミュレーション研究センター
日本学術振興会　特別研究員
佐竹　真介

発表内容

- SX-7上でHPFによる大規模並列計算に用いる擬似乱数の並列生成サブルーチンを作成した。
- 擬似乱数のアルゴリズムとして、Mersenne Twister を採用した。
- 乱数発生にかかる時間測定および乱数の統計的な性質についてのチェックを行った。

モンテカルロシミュレーションにおける並列擬似乱数の必要性

- 並列計算機上での数値シミュレーションを行う上で、プログラムの高並列化は計算機の性能を引き出す上で非常に重要

アムダールの法則

並列化による速度向上率: $e = \dfrac{1}{s + \dfrac{1-s}{n}}$

s: 逐次処理部の割合、 n:プロセッサー数

s＝5%、n＝1000でも e=20程度にしか得られない。

- プラズマ中の輸送シミュレーションに用いられるモンテカルロ法は、独立事象を扱うために高い並列化を実現しやすい。
- その一方、大規模なモンテカルロ計算では膨大な大きさの乱数列が必要となる。
（粒子数×ステップ数のオーダー）
- 一般的な擬似乱数アルゴリズムは数列の漸化式を逐次的に解くことで得られるため、並列化できない。

乱数発生サブルーチンが並列化効率を上げる上でボトルネックになってしまう

並列化乱数に要求されるもの

- 並列に生成された乱数列が、互いに統計的な相関を持たない独立な乱数列であること（種を変えるだけの並列化は危険）
- 個々の擬似乱数列自体の統計的な性質が良好であること（長周期、多次元構造を持たない、etc…）
- SX-7上で動かす場合、並列性だけでなく高いベクトル化率も求められる
- 乱数列の再現性　[ハードウェア並列乱数にはない利点（欠点？）]

Mersenne Twister(MT)を用いた並列擬似乱数生成

- 並列乱数生成アルゴリズムとして、松本真氏(京都大学)のMersenne Twisterと、その並列種行列発生ルーチンDynamic Creatorを用いる。(この業績で1999年IBM科学賞を受賞されている)
- ソースコードが公開されており、無償で利用が可能
- ベクトル化効率も高い

Mersenne Twisterの特徴

- Generalized Feedback Shift Register (GFSR)乱数（M系列とも呼ばれる）の1種
- 長周期性（メルセンヌ素数 例：p=19937に対し2^p-1の周期長）
- 超多次元均等分布性
- ビット操作（shift,xor,and)しか使わないのでアルゴリズムの複雑さのわりに高速

しかし、擬似乱数として使用する上で要求される、多次元均等分布性以外の様々な統計的品質に対し良好であるか資料が少ない（評判だけは良い）。

MTのアルゴリズム（1）

$\{\mathbf{x}_i\}$：w（＝32）bit長のベクトル列（状態ベクトル）
に対し、

$$\mathbf{x}_{n+k} := \mathbf{x}_{n+m} \oplus \left(\mathbf{x}^u_k | \mathbf{x}^l_{k+1}\right) \mathrm{A}$$

ここで、 $\mathrm{A} = \begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ a_0 & a_1 & a_2 & \cdots & a_{w-1} \end{pmatrix}$ ：Companion Matrix

$(\mathbf{x}^u_k | \mathbf{x}^l_{k+1})$ ：x_kの上位rビットとx_{k+1}の下位w-rビットを連結したもの

$\oplus$ ：排他的論理和（XOR）

また、$nw - r = p$（メルセンヌ数）となるように取られる。

MTのアルゴリズム(2)

(1)で得られた$\{\mathrm{x}_i\}$に対し、次のようなtemperingと呼ぶ操作を行い、wビット長乱数列$\{\mathrm{y}_i\}$を生成する。

$$
\begin{aligned}
\mathrm{y} &= \mathrm{x} \oplus (\mathrm{x} >> u) \\
\mathrm{y} &= \mathrm{y} \oplus ((\mathrm{y} << s)\mathrm{AND}\ \mathrm{b}) \\
\mathrm{y} &= \mathrm{y} \oplus ((\mathrm{y} >> t)\mathrm{AND}\ \mathrm{c}) \\
\mathrm{y} &= \mathrm{y} \oplus (\mathrm{y} >> l)
\end{aligned}
$$

ここで、$(\mathrm{y} << s)$等はsビット左(右)シフト操作を表す。また、tempering parameter b, c は与えられた A とs、tに対し乱数列$\{\mathrm{y}_i\}$の最大周期長が $2^p - 1$ となるように選ばれる。

MTの並列化

- 適当に与えられたCompanion Matrixに対し、最大周期列を与えるtempering parameterを求める Dinamic Creatorが松本氏によって公開されている。
- 異なる種行列(A, b, c)から生成される乱数列はその特性方程式が互いに素であり、それらの乱数列は相関を持たないとされる。
- Dinamic Creator自体はTry&Errorで種行列を検索するので、SXでの実行には向かない ⟹ 種行列はデスクトップPCで生成。

MTのベクトル化効率

- MTのベクトル長は $nw - r = p$ を与えるnの大きさ程度になるので、より大きなメルセンヌ数pを用いた方がベクトル効率が良い。

p	521	4423	9941	19937
n	17	139	311	624
ベクトル長	19	110	215	220

しかし、独立な種行列を多数個作るのにかかる時間はpに対し指数関数的に増大する。例：160個生成ならp=521の場合1分、p=19937の場合約2週間（athlon XP2400+上）。

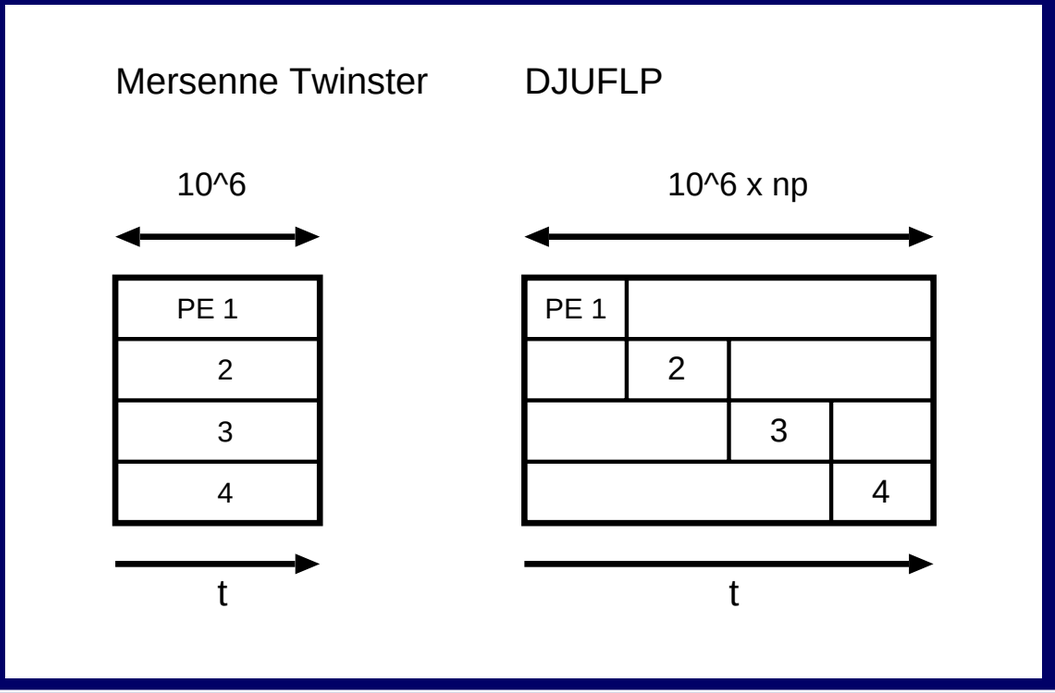
Parallel MT on HPF

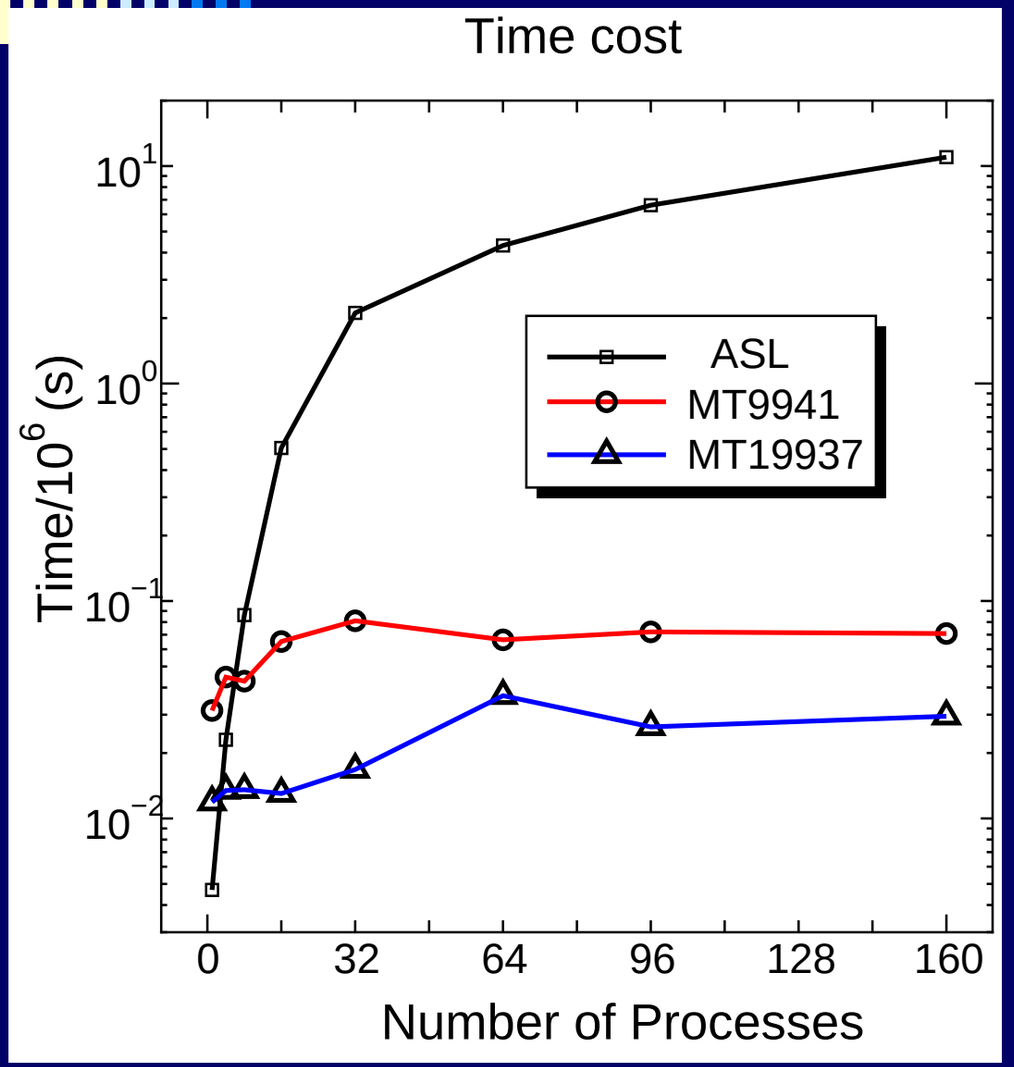
- SX-7での並列乱数生成はHPFを用いて記述される。
- 各プロセスはDinamic Creatorで作られた異なる種行列の値をファイルから読み込み、生成ルーチンが初期化される。
- 乱数列はr(nr,np)の様な2次元配列に格納される。ここで、nrは1プロセス当たりの乱数の数、npは並列プロセス数で、2次元目でdistributeされている。
- 並列乱数の生成には call grnd(r,nr) と呼ぶだけでよい。
- 各プロセス上の状態ベクトル $\{\mathbf{x}_i\}$ の値をmoduleに保持することで、次回callされた時に前回の続きの乱数列を生成できる。

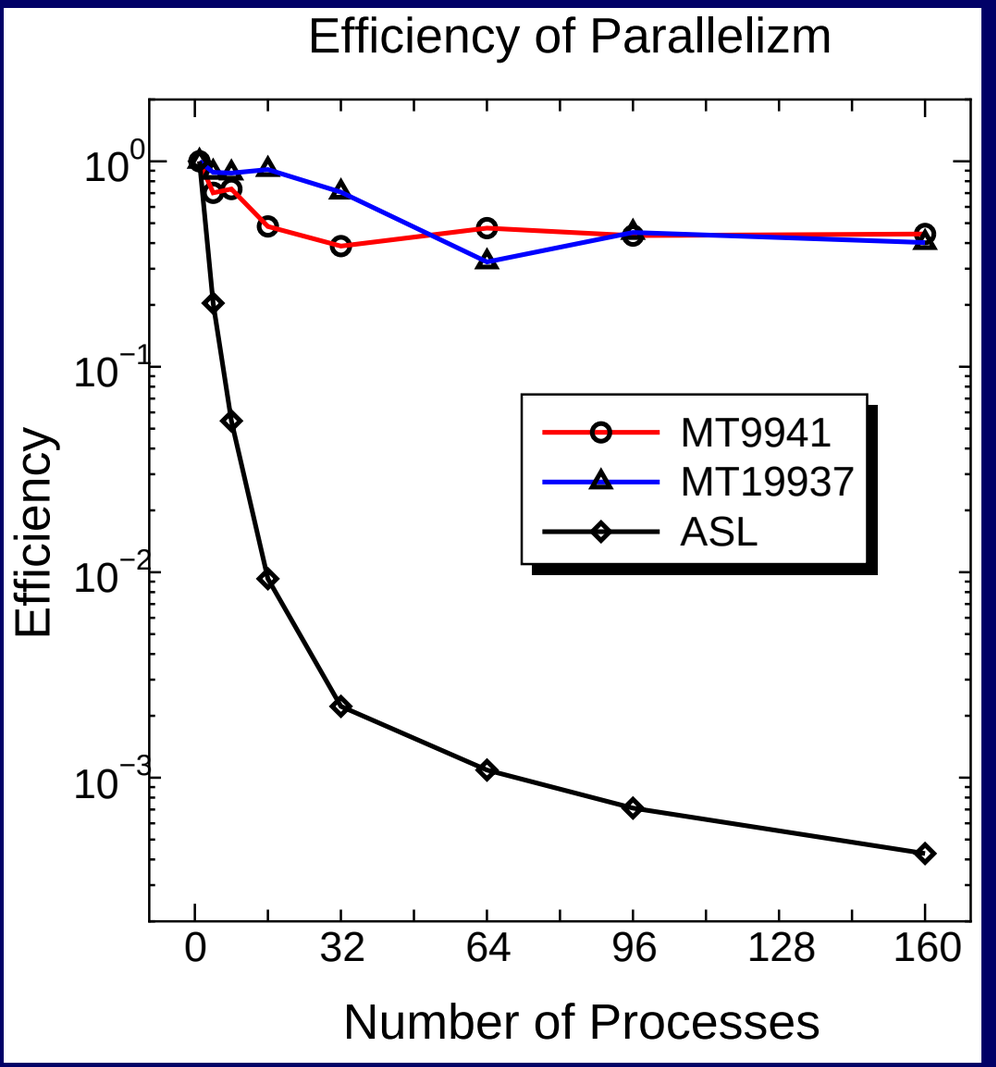
（サンプルコード）

乱数発生の時間測定

- P=9941, 19937の並列化MTサブルーチンの乱数発生にかかる時間を、ASLライブラリのDJUFLPルーチン（非並列、M系列）と比較した。
- 1プロセス当たり100万個で固定し、HPFプロセス数を増やすことで総発生数を変化させて計測。





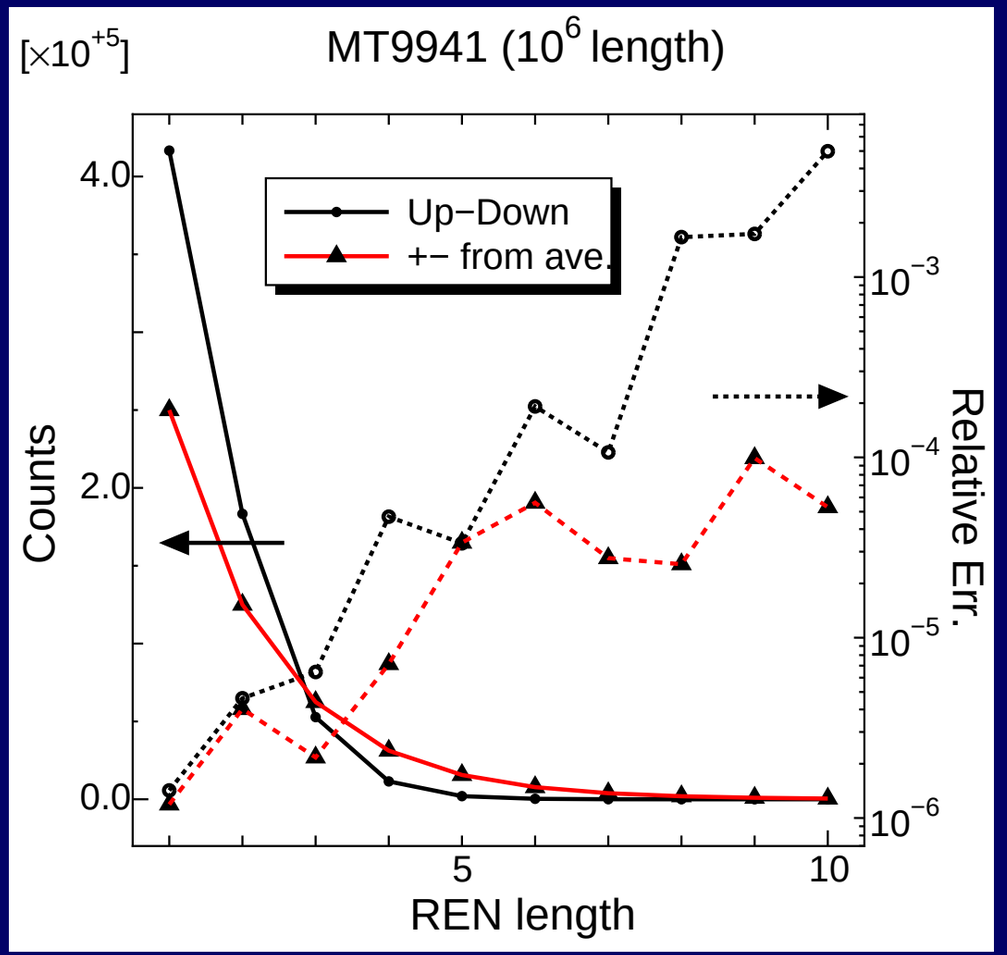


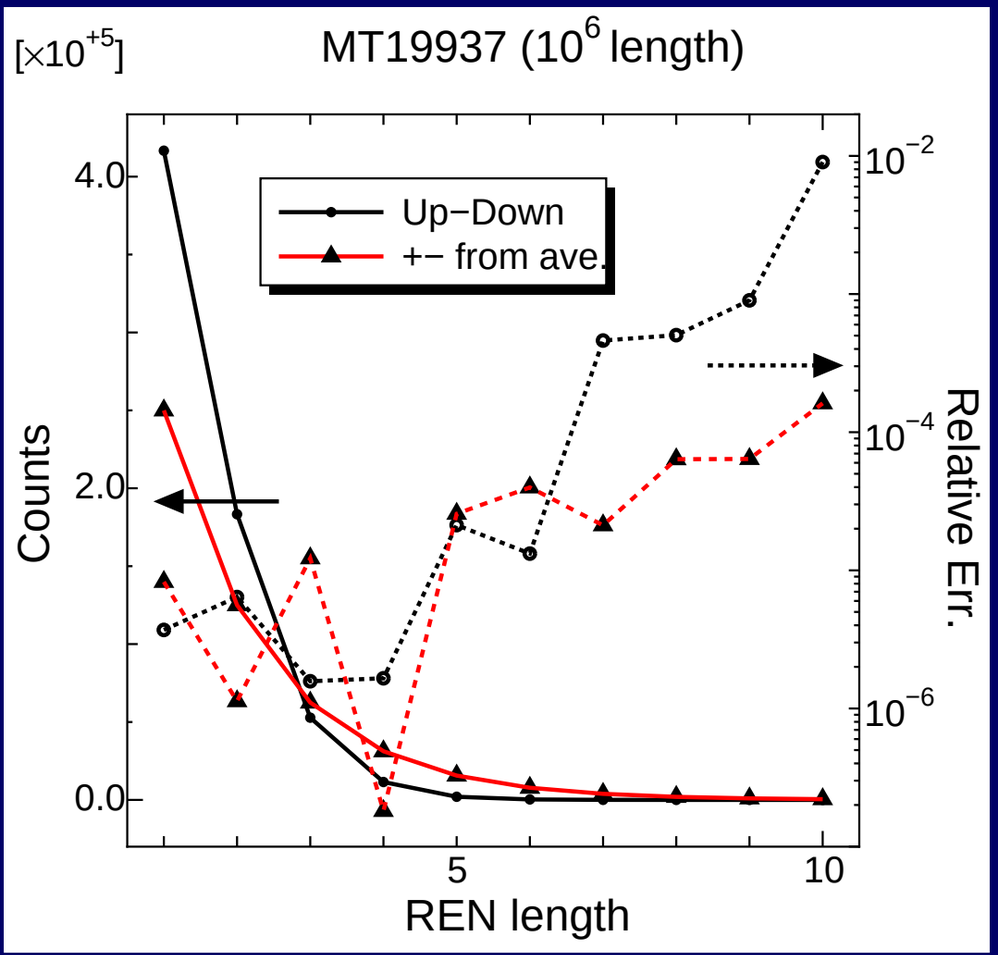
8並列以上でMTの方が生成時間が短い。HPFのオーバーヘッドもMTではあまり影響しない。

乱数の統計的品質の検定（1）

- まず、並列に生成された各々の乱数列に対して、乱数としての品質を検定した。
- 検定項目は以下のとおり。
  - 上昇、下降の連の出現確率
  - 平均値より大、または小の連の出現確率
  - 連続した2数がギャップΔの間に入る確率
- いずれも良い性質を示した。

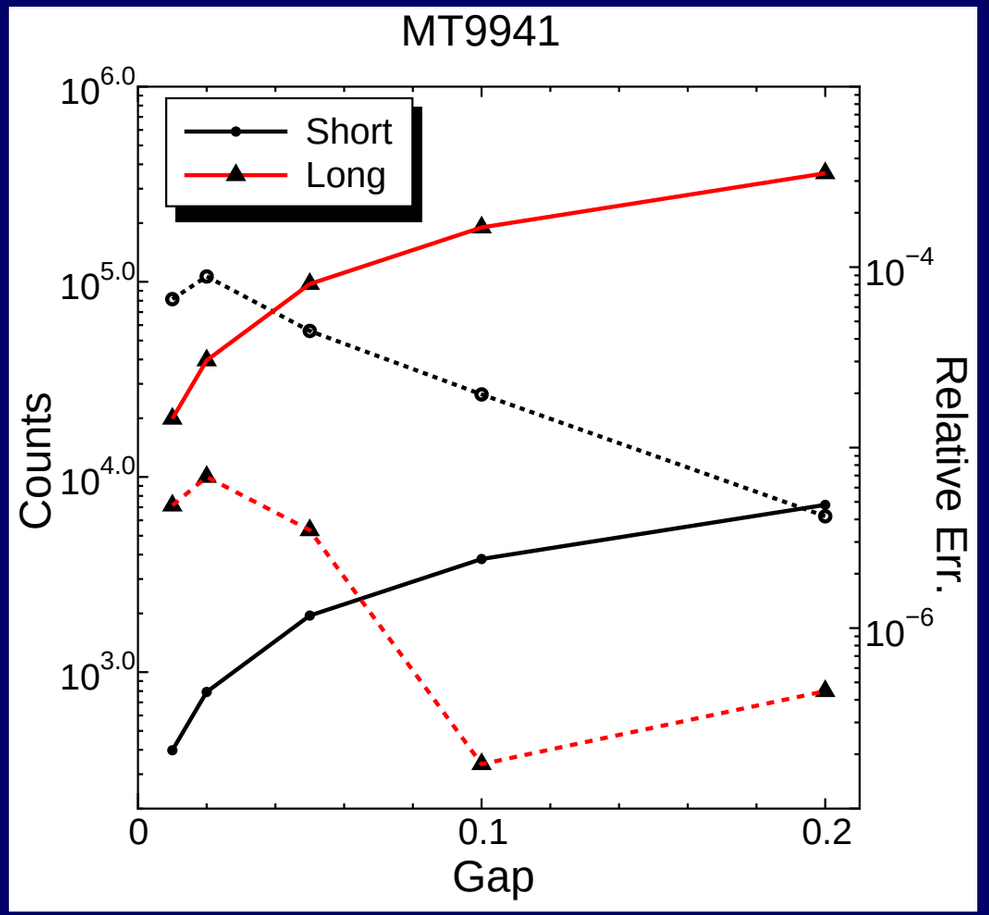
連検定の結果(上昇、下降、土 平均値)

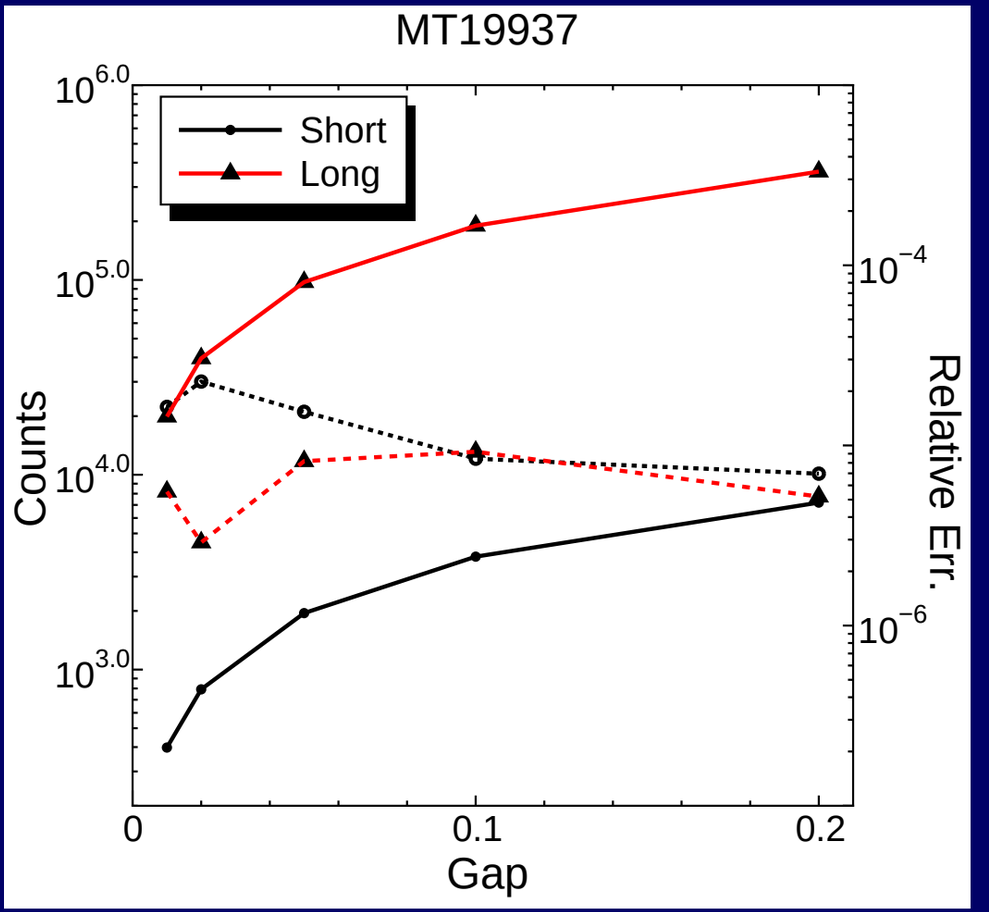




ギャップ検定の結果

（short=1万、long=100万個の区間中）

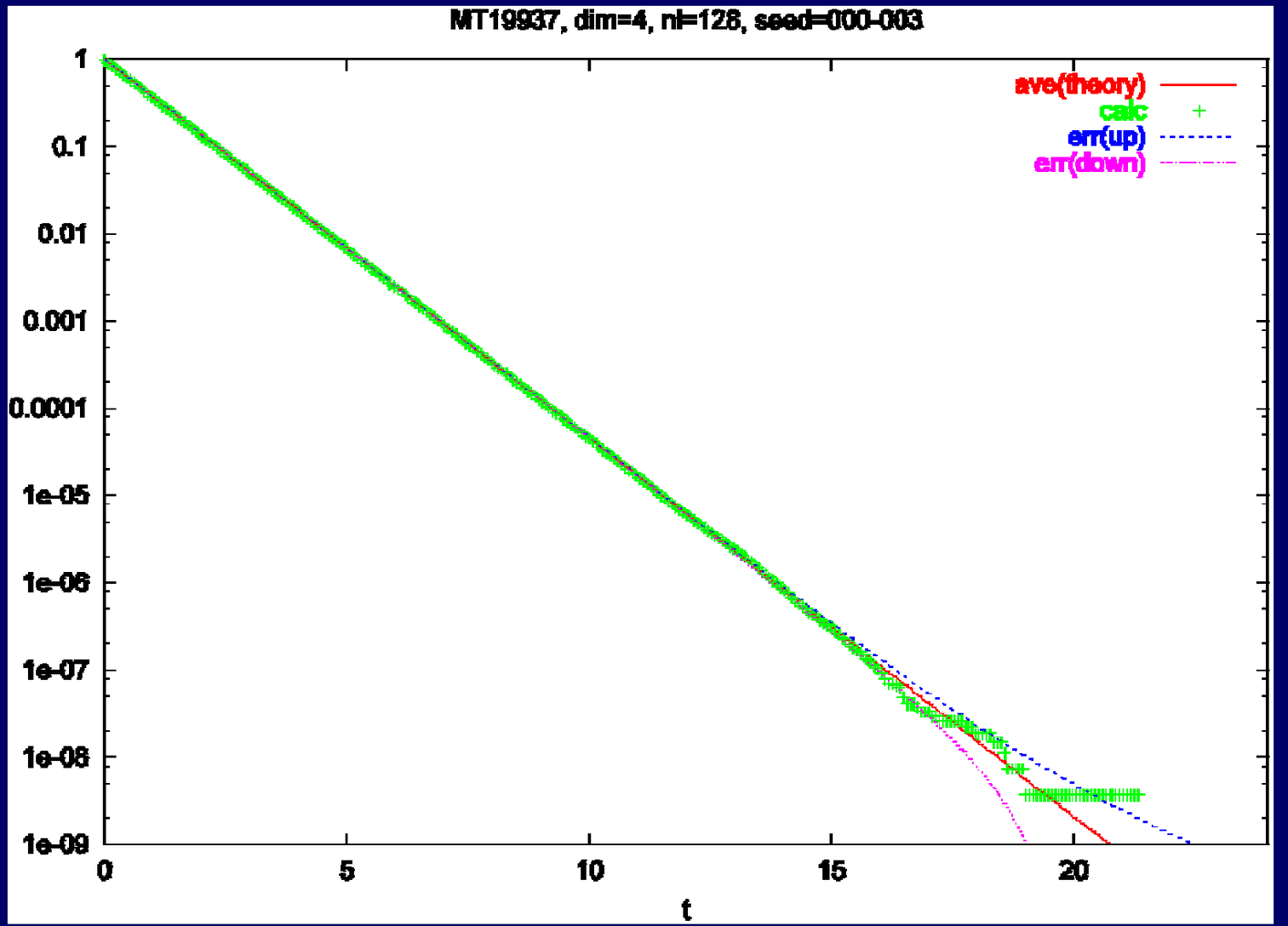




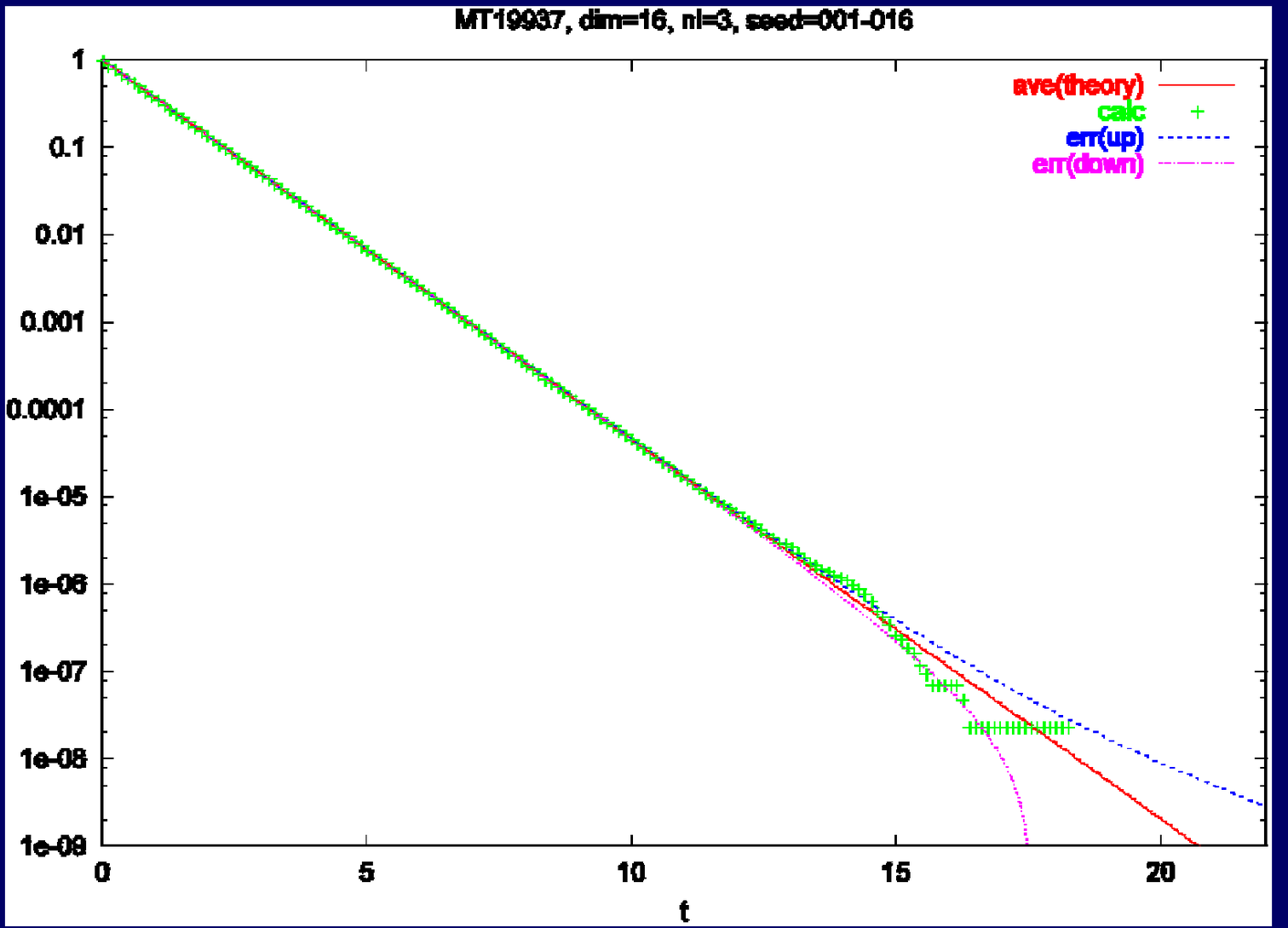
乱数の統計的品質の検定(2)

- 並列乱数間に相関が生じていないか、次のような多次元構造に関する検定を行った。
- n並列で生成された乱数$x_i^m \ (m = 1, 2, \cdot, n)$からn次元ベクトル列 $\mathbf{x}_i = (x_i^1, x_i^2, \cdots, x_i^n)$ を作る。
- n次元立方格子中の $\mathbf{x}_i$によって指定された格子点をONにする。
- この操作を繰り返すと、OFFのままの格子点数は、並列乱数に相関がなければ指数関数的に減少する。

4並列の相関チェックの例 （128^4格子）



１６並列の相関チェックの例 （3^{16}格子）



まとめ

- Mersenne Twisterを用いた並列擬似乱数発生ルーチンをSX-7上でHPFにより実装した。
- 高い並列化、ベクトル化効率を達成した。
- 乱数の各種検定から、統計的な品質の良好さが確認された。

参考：http://www.math.keio.ac.jp/~matumoto/mt.html

「モンテカルロ法とシミュレーション」 津田孝夫、培風館

e-mail : satake@nifs.ac.jp