2K-3

GPU 上でのリアルタイム並列乱数生成とその応用

井上 佑哉† 高丸 尚教‡

中部大学大学院工学研究科情報工学専攻† 中部大学情報工学科‡

1. はじめに

核融合プラズマ分野など大規模シミュレーション研究では,10^15 個以上の乱数を必要とすることは,珍しくない.並列乱数生成は 30 年以上もの歴史を持つ.ただし,並列に乱数を生成した場合の安定性を保障するために,生成速度を増加させることが困難であった.GFSR 系の Mersenne Twister は,周期長とスレッド数を勘案して乱数種の設定を行っておけば,並列化による生成速度の向上と共に安定性を担保できる.本研究では,GPGPU を用いて並列化を促進すると共に,乱数スレッドとシミュレーション・スレッドとの間で遅滞なく乱数を共有できる手法を提案する.

そこで, 乱数発生時間を事実上 0 に近づけるために,GPGPU のパイプラインのうち休眠状態に入ったパイプライン上で乱数生成を行い,再度スレッドを動作させた際に,GPU メモリ上に新しい乱数が生成済みとなっているようにしたい.

そこで,プロセスが管理しているメモリ情報を別のプロセスから参照し利用することができるインターフェースの構築を目指す.

2. 提案手法

シミュレーションで用いる乱数はシミュレーションを行うスレッドとは異なるスレッドで生成する. 図 1 に示すように乱数生成スレッド(R_1, $R_{2,\dots}$)は GPU のメモリ上に乱数をリアルタイムに生成する.シミュレーションスレッド(S_1, $S_{2,\dots}$)は乱数生成スレッド対してに乱数を要求し,乱数生成スレッドはそれに応じてシミュレーション・スレッドに乱数を受け渡す. 乱数生成スレッドはシミュレーション・スレッドに乱数を受け渡した後,新たに乱数を生成する.

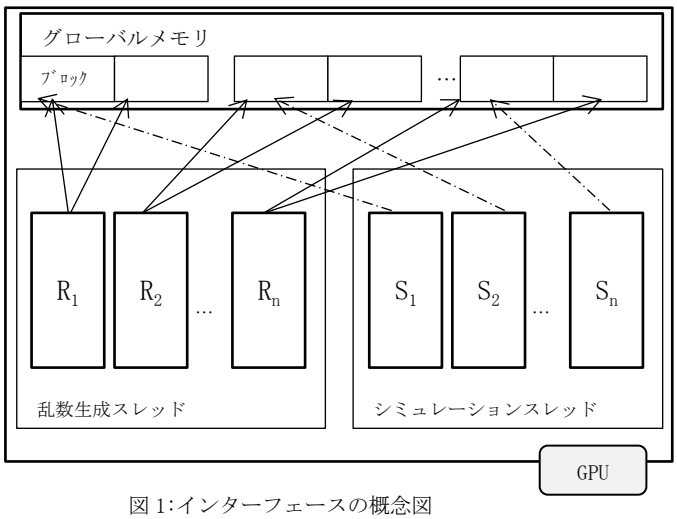

図 1:インターフェースの概念図

3. 具備すべきインターフェース機能は少なくとも以下の 3 つである.

1. 他のプロセスへアクセスする手段
2. GPU 上でスレッドを実行する手段
3. GPU 上で乱数を並列に生成する手段

3.1 他のプロセスへのアクセス

すべてのプロセスはそれぞれ独立した仮想アドレス空間にデータを格納している.そのためプロセス同士でデータを共有するにはプロセス間通信を行う必要がある.プロセス間で通信を行うために,ファイルマッピングを利用することとする.具体的には OpenCL を利用する.

計算環境として,表 1 に示す環境を利用する.

表 1:計算環境

環境	デバイス	コア数	メモリ
環境 1	AMD Phenom Ⅱ X4 980	4	8GB
	TeslaC2070	448	6GB
環境 2	AMD athlon64 x2 3800+	2	2GB
	TeslaC1060	240	4GB

「タイトル」英文による記述

†Yuya Inoue, Dept.Comp.Sci.,Grad.School Eng.,Chubu Univ.

‡Hisanori Takamaru, Dept.Comp.Sci.Col.Eng.,Chubu Univ.

3.2 GPU 上で乱数を並列に生成する手段

GPU 上で乱数を生成する際には,Mersenne Twister を利用する.

4. 乱数の消費,生成プロセス

メモリブロックを分割して,各メモリブロック毎に乱数を生成する. 図 2 に示すように,メモリブロックにはフラグが設けられており,このフラグによって乱数の生成が必要か否かを判断する.

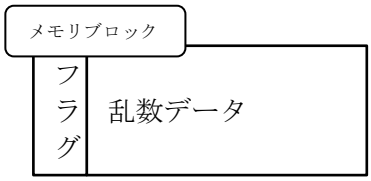

図 2:メモリブロックの概念図

図 3 に示すように,順次メモリブロック内の乱数が消費されていく.ブロック内の乱数がすべて消費されると,乱数生成プロセスはシミュレーションで利用していたメモリブロックに対して乱数の生成を開始する.同時にシミュレーションコードは次のメモリブロックを読みに行く.これにより乱数生成とシミュレーションを並行して行うことが可能となる.

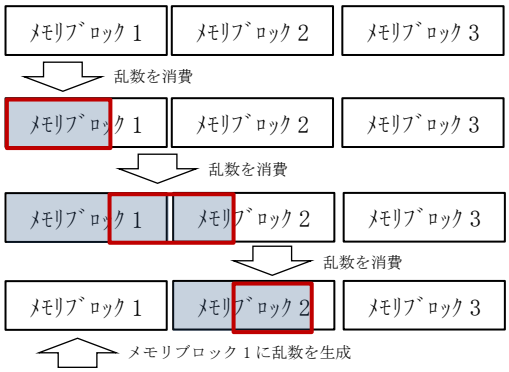

図 3:乱数の消費と生成のサイクル

4.1 実装例

ユーザーは初期化関数を呼ぶ際に,利用するメモリのサイズを指定する.初期化関数はユーザーにとって必要な情報を配列に格納し返却する.

実際に乱数を利用する際にはユーザーがメモリの生成,乱数が利用可能かの確認,ポインタの更新等の関数を呼ぶことで乱数生成プロセスの動きを制御する.

擬似コード

初期化

```
randSize : 利用する乱数配列の最大サイズ
info     : ユーザーに必要な情報を返す
initRand(randSize, *info){
  /* randSize をもとに環境をセットアップ */
  /* info に情報を格納 */
  /*   ・デバイスが持つメモリのサイズ */
  /*   ・1 ブロックのサイズ */
  /*   ・乱数生成スレッドの*/
```

実装例

```
NB: メモリ分割の際の 1 ブロックのサイズ
N : 必要な乱数の総数
Simulation(){
  for( k=0 ; k<N ; k+=NB ){
    /* メモリが利用可能かチェック */
    for( i=k ; i<k+min(N-k, NB) ; i++ ){
      /* 乱数を消費 */
    }
    /* メモリの生成命令,フラグを立てる */
    /* ポインタを更新 */
  }
}
```