Generation of Random Numbers on Graphics Processors: Forced Indentation In Silico of the Bacteriophage HK97

ARTICLE *in* THE JOURNAL OF PHYSICAL CHEMISTRY B · DECEMBER 2010

Impact Factor: 3.3 · DOI: 10.1021/jp109079t · Source: PubMed

4 AUTHORS, INCLUDING:

A. A. Zhmurov
Moscow Institute of Physics and Technology
16 PUBLICATIONS **123** CITATIONS
SEE PROFILE

Yroslav A. Kholodov
Moscow Institute of Physics and Technology
16 PUBLICATIONS **72** CITATIONS
SEE PROFILE

THE JOURNAL OF PHYSICAL CHEMISTRY B

ARTICLE

pubs.acs.org/JPCB

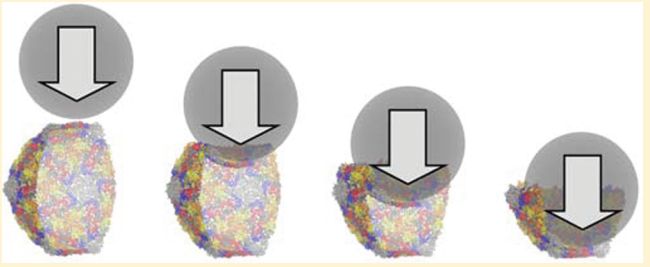
Generation of Random Numbers on Graphics Processors: Forced Indentation *In Silico* of the Bacteriophage HK97

A. Zhmurov,[†,§] K. Rybnikov,[‡] Y. Kholodov,[§] and V. Barsegov*[,†,§]

[†]Department of Chemistry and [‡]Department of Mathematics, University of Massachusetts, Lowell, Massachusetts 01854, United States
[§]Moscow Institute of Physics and Technology, Moscow Region, Russia, 141700

Supporting Information

ABSTRACT: The use of graphics processing units (GPUs) in simulation applications offers a significant speed gain as compared to computations on central processing units (CPUs). Many simulation methods require a large number of independent random variables generated at each step. We present two approaches for implementation of random number generators (RNGs) on a GPU. In the one-RNG-per-thread approach, one RNG produces a stream of random numbers in each thread of execution, whereas the one-RNG-for-all-threads method builds on the ability of different threads to communicate, thus, sharing random seeds across an entire GPU device. We used these approaches to implement Ran2, Hybrid Taus, and Lagged Fibonacci algorithms on a GPU. We profiled the performance of these generators in terms of the computational time, memory usage, and the speedup factor (CPU time/GPU time). These generators have been incorporated into the program for Langevin simulations of biomolecules fully implemented on the GPU. The ∼250-fold computational speedup on the GPU allowed us to carry out single-molecule dynamic force measurements *in silico* to explore the mechanical properties of the bacteriophage HK97 in the experimental subsecond time scale. We found that the nanomechanical response of HK97 depends on the conditions of force application, including the rate of change and geometry of the mechanical perturbation. Hence, using the GPU-based implementation of RNGs, presented here, in conjunction with Langevin simulations, makes it possible to directly compare the results of dynamic force measurements *in vitro* and *in silico*.



I. INTRODUCTION

Graphics processing units (GPUs) are emerging as an alternative programming platform that provides high raw computational power for scientific applications.[1−7] The computational efficiency of contemporary GPUs reaching ∼1 TFlops for a single chip[8] enables one to utilize GPUs as performance accelerators in compute-intensive molecular simulations.[1,2,6,7] The GPU-based calculations can be performed concurrently on many computational cores (Arithmetic Logic Units, ALUs) grouped into multiprocessors, each with its own flow control and cache units. For example, the number of multiprocessors per GPU can reach 15 on the most up-to-date graphics cards (e.g., GeForce GTX 480 from NVIDIA) bringing the total number of ALUs to 480 per chip. Although a GPU device has its own global memory with ∼10 times larger bandwidth compared to DRAM on a CPU, the number of memory invocations (per ALU) should be minimized to optimize the GPU performance. Hence, the computational task should be compute-intensive so that, most of the time, the GPU performs computations rather than reads/writes data.[8] This makes an N body problem a prime candidate for the numerical implementation on the GPU.

Langevin dynamics (LD) simulations, Monte Carlo (MC) simulations, and molecular dynamics (MD) simulations in implicit solvent, widely used to access the microscopic transitions in biomolecules, are among the many applications that can be implemented on a GPU. Since in MD simulations in implicit water and in LD simulations the effect of solvent molecules are described implicitly, these methods require a reliable source of $3N$ normally distributed random numbers, $g_{i,\alpha}$ ($i = 1, 2, ...,N$, and $\alpha = x, y, z$), generated at each step of a simulation run, in order to compute the Gaussian random force $G_{i,\alpha}$. For example, in MD simulations in implicit water,[9,10] the dynamics of the ith particle are governed by the equations of motion for the particle position, $d\mathbf{R}_i/dt = \mathbf{V}_i$, and velocity, $m_i\, d\mathbf{V}_i/dt = \xi\mathbf{V}_i + f(\mathbf{R}_i) + \mathbf{G}_i(t)$, where m_i is the particle mass, ξ is the friction coefficient, and $f(\mathbf{R}_i) = -\partial U/\partial \mathbf{R}_i$ is the molecular force exerted on the ith particle due to the potential energy U. In LD simulations, the dynamics of the ith particle are obtained by following the Langevin equation in the

Special Issue: Shaul Mukamel Festschrift

Received: September 22, 2010
Revised: December 1, 2010
Published: December 31, 2010

overdamped limit, $\xi\, d\mathbf{R}_i/dt = f(\mathbf{R}_i) + \mathbf{G}_i(t)$.[11] In MC simulations, the results of multiple trials, each driven by some random process, are combined to extract the average answer.

An algorithmic (pseudo)random number generator (RNG) must have a long period and must meet the conflicting goals of being fast while providing a large amount of random numbers of proven statistical quality.[12] There is an extensive body of literature devoted to random number generation on a central processing unit (CPU).[13] Yet, because of the fundamental differences in processor and memory architecture of a GPU, the CPU-based methods cannot be easily translated from a CPU to a GPU. One option is to have random numbers pregenerated on the CPU, and use these numbers in the simulations on the GPU. However, this requires a large amount of memory for an RNG. For a system of 10^4 particles in three dimensions, 3×10^4 random numbers are needed at each simulation step. To generate these numbers, say, every 10^2-10^3 steps, requires $3 \times 10^6-3 \times 10^7$ random numbers, which takes 12−120 MB of the GPU memory. This might be significant even for the most up-to-date GPUs, which have limited ∼1 GB memory.

We explored this option in Langevin simulations of N Brownian oscillators[11] using the Hybrid Taus and additive Lagged Fibonacci algorithms described below. We compared the computational time as a function of the system size N for three different implementations of Langevin simulations: (1) random numbers and dynamics are generated on the CPU, (2) random numbers, obtained on the CPU, are transferred to the GPU and used to generate dynamics on the GPU, and (3) random numbers and dynamics are generated on the GPU. The results for a 2.83 GHz Intel Core i7 930 CPU and a 1.4 GHz GeForce GTX 480 GPU show that, starting from $\sim 10^2$ particles, it becomes computationally expensive to generate random numbers on the CPU, transfer them to the GPU, and generate stochastic trajectories on the GPU (Figure 1a). We found a substantial speedup for LD simulations fully implemented on the GPU, compared to the CPU-based LD simulations, which also depends on the RNG choice and system size N (Figure 1b). We observed a ∼10−250-fold speedup for Langevin simulations of $N = 10^3-10^6$ Brownian particles on the GPU (Figure 1b). Hence, for efficient molecular simulations in a stochastic thermostat, random numbers must be generated on the GPU device.

While there exist stand-alone implementations of RNGs on a GPU, to fully utilize computational resources of a GPU, an RNG should be incorporated into the main simulation program. This allows one to minimize read/write calls associated with invocation of the GPU global memory, and to generate streams of random numbers "on-the-fly", i.e., at each step of a simulation run, using fast GPU shared memory. Here, we describe the methodology for generating pseudorandom numbers on a GPU, which can be used in GPU-based implementations of MD simulations in implicit solvent, LD simulations, and MC simulations. In the next section, we focus on the linear congruential generator (LCG), and the Ran2, Hybrid Taus, and Lagged Fibonacci algorithms. These are used in section III to describe the methodology for generation of (pseudo)random numbers on a GPU. Pseudocodes are given in the Supporting Information (SI). We test the GPU-based implementations of the LCG, and Ran2, Hybrid Taus, and Lagged Fibonacci algorithms in section IV, where we present the application-based assessment of their statistical properties using the Ornstein−Uhlenbeck process. We also profile these generators in terms of the computational time and memory usage. We use these algorithms in conjunction with the C_α-based

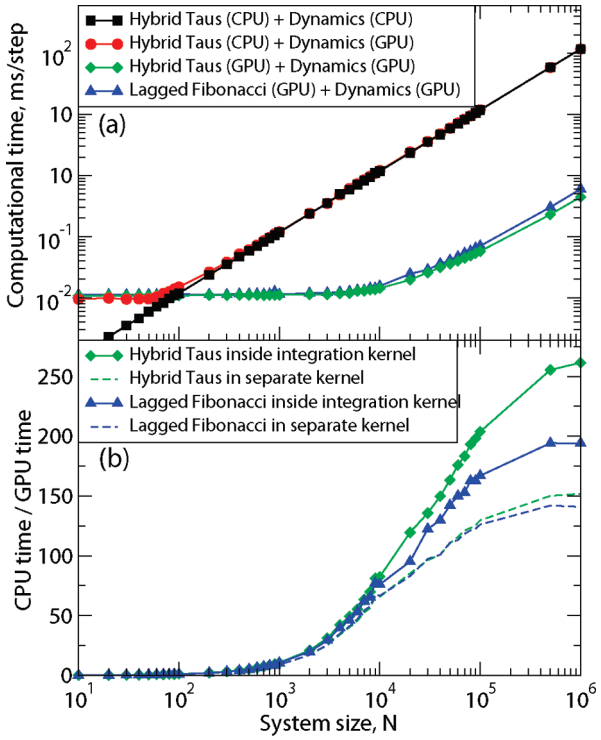


Figure 1. (a) The computational time for LD of N Brownian oscillators with the Hybrid Taus and additive Lagged Fibonacci generators of (pseudo)random numbers. We considered the three implementations, where (1) random numbers and LD are generated on the CPU (Hybrid Taus (CPU)+Dynamics (CPU)), (2) random numbers are obtained on the CPU, transferred to the GPU and used to propagate LD on the GPU (Hybrid Taus (CPU)+Dynamics (GPU)), and (3) random numbers and LD are generated on the GPU (Hybrid Taus (GPU)+Dynamics (GPU) and Lagged Fibonacci (GPU)+Dynamics (GPU)). (b) The computational speedup (CPU time versus GPU time) for LD simulations fully implemented on the GPU and on the single CPU core. We compared the two options when an RNG (Hybrid Taus or Lagged Fibonacci) is organized in a separate kernel or is inside the main (integration) kernel. We ran long trajectories (10^6 steps) to converge the speedup factor.

coarse-grained self-organized polymer (SOP) model[14−16] in section V to perform single-molecule dynamic force measurements *in silico* to characterize the physical properties of the viral capsid HK97, a λ-like dsDNA bacteriophage.[17] This is a model system for numerous studies of the kinetics of virus maturation, pressure-induced expansion, and the mechanism(s) of infection of cells.[18,19] All the simulations were carried on the GPU GeForce GTX 480 (NVIDIA). The main results are discussed in section VI. We conclude in section VII.

II. PSEUDORANDOM NUMBER GENERATORS: THE GPU PERSPECTIVE

We focus on algorithmic RNGs, the most common type of deterministic random number generators.[20] An RNG produces a sequence of random numbers, u_i, which is supposed to imitate independent uniform random variates from the unit interval (0,1). In implicit water models and in LD simulations of biomolecules, normally distributed random forces are used to emulate stochastic kicks from the solvent molecules. To generate the distribution of random forces, a common approach is to convert

the uniformly distributed random variates (u_i) into the Gaussian distributed random variates (g_i) using a variety of methods.[21−23] Here, we adopt the most commonly used Box−Mueller transformation.[23]

There are three main requirements for a numerical implementation of an RNG: (1) good statistical properties, (2) high computational speed, and (3) low memory usage. Because a deterministic sequence of random numbers comes eventually to a starting point, $u_{n+p} = u_n$ (Poincaré recurrence), an RNG should have a long period p. An RNG must be tested empirically for randomness, i.e., for the uniformity of distribution and for the independence.[12] The statistical tests of randomness are accumulated in the DIEHARD test suite and in the TestU01 library.[12,24−26] In addition, an RNG must pass application-based tests of randomness that offer exact solutions to the test applications. Using random numbers of poor statistical quality might result in insufficient sampling, unphysical correlations,[27,28] and unrealistic results, which leads to errors in practical applications.[29] A good quality RNG should also be computationally efficient so that random number generation does not become a major bottleneck. In Langevin simulations of a biomolecule on a GPU, one can follow a long trajectory over 10^{10} iterations, which requires $\sim 10^{15}$ random numbers for a system of $N = 10^5$ particles. The condition of low memory usage is also important since contemporary graphics processors have low on-chip memory, ∼64 KB per multiprocessor (graphics cards with Fermi architecture) compared to ∼2 MB memory on the CPU. Hence, an efficient RNG algorithm must use a limited working area without invoking the relatively slow GPU global memory.

A fast RNG employs simple logic and a few state variables to store its current state, but this may harm its statistical properties. On the other hand, using a more sophisticated algorithm with many arithmetic operations or combining several generators into a hybrid generator allows one to improve statistics, but such generators are slower and use more memory. Here, we focus on some of the most widely used LCG,[13] Ran2,[13] Hybrid Taus,[13,20,30,31] and Lagged Fibonacci algorithms (Appendix A).[13,32] LCG can be used in performance benchmarks since it employs a very fast algorithm. Ran2 is a standard choice for many applications due to its long period $p > 10^{18}$, good statistical quality, and high computational performance on a CPU. However, Ran2 requires a large amount of on-chip GPU local and global memory to store its current state. In Hybrid Taus, several simple algorithms are combined to improve the statistical characteristics of the random numbers produced. It scores better in terms of the computational speed on a GPU than KISS, the best known combined generator,[33] and its long period $p > 10^{36}$ makes it a good choice for GPU-based computations. Lagged Fibonacci employs simple logic while producing random numbers of high statistical quality.[12] It is used in distributed MC simulations, and it can also be utilized in GPU-based computations. We employed the additive Lagged Fibonacci RNG, which generates floating point variates without the usual floating of random integers.

III. LCG, RAN2, HYBRID TAUS, AND LAGGED FIBONACCI ON A GPU

A. Basic Ideas. To solve an N body problem on a GPU, an RNG should produce random numbers simultaneously for all particles. One possibility is to build an RNG into the main simulation kernel to maximize the amount of computations on a GPU while minimizing the number of calls of the GPU global memory (read/write operations). To fully utilize the GPU resources, the total number of threads should be ∼10-times larger than the number of computational cores, so that none of the cores waits for the others to complete their tasks. We employ the cycle division paradigm,[32] in which a single sequence of random numbers is partitioned among many computational threads running concurrently across an entire GPU, each producing a stream of random numbers. Since most RNG algorithms are based on sequential transformations of the current state (LCG, Hybrid Taus, and Ran2), the most common way of partitioning the sequence is to provide each thread with different seeds while also separating the threads along the sequence to avoid interstream correlations. This is the basis of the one-RNG-per-thread approach (Figure 1a in the SI). On the other hand, Mersenne Twister and Lagged Fibonacci algorithms, which employ recursive transformations, allow one to leap ahead in a sequence and to produce the $(n+1)$th random number without knowing the nth number.[32−34] The leap size, which, in general, depends on the parameters of an RNG, can be adjusted to the number of threads (number of particles N), or multiples of N. Then, all N random numbers can be obtained simultaneously, so that the jth thread produces numbers j, $j+N$, $j+2N$, and so forth. At the end of each simulation step, threads must be synchronized to update the current RNG state. Hence, the same RNG state can be used in all threads, each updating just one element of the state. We refer to this as the one-RNG-for-all-threads approach (Figure 1b in the SI).

B. One-RNG-per-Thread Approach. The idea is to run the same RNG algorithm in each thread to generate different subsequences of the same sequence of random numbers, but starting from different initial seeds. The CPU initiates N sets of random seeds (one for each RNG) and passes them to the GPU global memory (Figure 2 in the SI). To exclude correlations, these sets should come from an independent sequence of random numbers. Each thread on the GPU reads its random seeds from the GPU global memory and copies them to the GPU local (per thread) memory or shared (per thread block) memory. Then, each RNG generates random numbers without using the slow GPU global memory. At the end of a simulation step, each RNG saves its current state to the global memory and frees shared memory. Since each thread has its own RNG, there is no need for thread synchronization. However, when particles interact, threads must be synchronized. In the simulations, arrays of the initial seeds and the current state should be arranged for coalescent memory read to speedup the global memory access.

In the one-RNG-per-thread setting, an RNG should be very light in terms of the memory usage. Small size of on-chip memory can be insufficient to store the current state of an RNG with complex logic. The amount of memory required to store the current state is proportional to the number of threads (number of particles N). Hence, a significant amount of memory has to be allocated for all RNGs to describe a large system. For example, LCG uses one integer seed to store its current state, which takes 4 bytes per thread (per generator) or ∼4 MB of memory for 10^6 threads (particles), whereas Hybrid Taus uses 4 integers, i.e. 16 MB of memory. These are acceptable numbers, given the hundreds of megabytes of the GPU memory. By contrast, Ran2 uses 35 long integers and a total of 280 bytes per thread, or ∼280 MB of memory for 10^6 threads (particles). As a result, not all seeds can be stored in on-chip (local or shared) memory,

and the GPU global memory has to be accessed to read/update the current state. In addition, less memory becomes accessible to other computational routines. This might prevent Ran2 from being used in the simulations of large systems on some graphics cards, including GeForce GTX 280 and GTX 295 (NVIDIA), with 768 MB of global memory (per GPU). Yet, this is not an issue when using high-end graphics cards, such as Tesla C2070 with 6 GB of global memory. In this paper, we utilized the one-RNG-per-thread approach to develop the GPU-based implementations of the LCG, Hybrid Taus, and Ran2 algorithms (Figure 2 in the SI). Pseudocodes are presented in Section I in the SI. Numerical values of the constant parameters for LCG, Ran2, and Hybrid Taus can be found, respectively, in Appendix A,[12] in ref 13, and in Section I in the SI.

C. One-RNG-for-All-Threads Approach. In the one-RNG-for-all-threads approach, one can utilize a single RNG by allowing all computational threads to share the state of a generator. This can be used in algorithms that are based on the recursive transformations, i.e., $x_n = f(y_{n-r}, y_{n-r+1}, ..., y_{n-k})$, where r is a recurrence degree and $k > r$ is a constant parameter, to obtain a random number in the nth step from the state variables generated in the previous steps $n - r, n - r + 1, ..., n - k$. If a sequence of random numbers is obtained simultaneously in N threads, each generating just one random number, then N random numbers are produced at each step. Given $k > N$, all the elements of the transformation have been obtained at the previous steps, in which case they can be accessed without thread synchronization. One of the algorithms that can be implemented on the GPU using the one-RNG-for-all-threads approach is additive Lagged Fibonacci (Figure 3 in the SI).[34] A pseudocode is presented in section II in the SI. When one random number is computed in each thread and when $sl > N$ and $ll - sl > N$, where ll and sl are the long and short lags, N random numbers can be obtained simultaneously on the GPU device; sl and ll could be taken to be sufficiently large to guarantee good statistical properties of the random numbers produced.

To initialize the Lagged Fibonacci RNG on the GPU, ll integers are allocated on the CPU. On the GPU, each thread reads two integers of the sequence (one for ll, and the other for sl), generates the resulting integer, and saves it to the location in the GPU global memory, which corresponds to ll. Setting $sl > N$ and $ll - sl > N$ guarantees that the same position in the array of integers (current state variables) will not be accessed by different threads at the same time. The window of N random numbers, updated in N threads, is moving along the array of state variables, leaping forward by N positions at each step. Importantly, the period of the Lagged Fibonacci generator, $p \sim 2^{ll+31}$, can be adjusted to the system size N by assigning large values to sl and ll, so that $p \gg N \times S$, where S is the number of simulation steps. Varying ll and sl does not affect the execution time, but changes the size of the array of state variables, which scales linearly with ll, the amount of integers stored in the GPU global memory. A large ll value is not an issue even when $ll \sim 10^6$, which corresponds to ~4 MB of the GPU global memory. Numerical values of the constant parameters for Lagged Fibonacci are given in Table 1 in the SI.

IV. BENCHMARK TESTING

A. Test of Randomness: Ornstein−Uhlenbeck Process. To assess the statistical performance of the GPU-based realizations of LCG, Ran2, Hybrid Taus, and Lagged Fibonacci, we carried out Langevin simulations of N independent Brownian oscillators[11] on a GPU. Each particle evolves on a harmonic potential, $U(R_i) = k_{sp}R_i^2/2$, where R_i is the ith particle position and k_{sp} is the spring constant, and is subject to random force. We employed this analytically tractable model to directly compare the simulation output with the exact results that would be obtained with truly random numbers. The LDs, $\xi\, dR_i/dt = -\partial U(R_1, R_2, ..., R_N)/\partial R_i + G_i(t)$,[14−16,35,36] were obtained numerically using the first-order integration scheme,[37]

$$R_i(t+\Delta t) = R_i(t) + f(R_i(t))\Delta t/\xi + g_i(t)\sqrt{2k_BT\Delta t/\xi} \quad (1)$$

where $f(R_i) = -\partial U(R_1, R_2, ..., R_N)/\partial R_i$ is the deterministic force, and g_i are the normally distributed random variates (with zero mean and unit variance) used to obtain the Gaussian random forces $G_i(t) = g_i(t)(2k_BT\Delta t/\xi)^{1/2}$. Numerical algorithms for the GPU-based implementation of Langevin simulations, used here, are presented in ref 38.

Numerical calculations for $N = 10^4$ particles were carried out with the time step $\Delta t = 1$ ps at room temperature, starting from the initial position $R_0 = 10^3$ nm, and using the diffusion constant $D = 0.25$ nm^2/ns. A soft harmonic spring ($k_{sp} = 0.01$ pN/nm) allowed us to follow long 1 ms trajectories over 10^9 steps. The average position $\langle R(t)\rangle$ and the two-point correlation function $C(t) = \langle R(t)R(0)\rangle$, obtained from the simulations, are compared in Figure 2 with their exact counterparts,[11,39] $\langle R(t)\rangle = R_i(0) \exp[-t/\tau]$ and $C(t) = (k_BT/k_{sp}) \exp[-t/\tau]$, where $\tau = \xi/k_{sp}$ is the characteristic time. We see that all RNGs describe well the exact Brownian dynamics except for the LCG. Indeed, $\langle R(t)\rangle$ and $C(t)$, obtained using Ran2, Hybrid Taus, and Lagged Fibonacci algorithms, practically collapse on the theoretical curve of these quantities. By contrast, using LCG results in a repeated pattern for $\langle R(t)\rangle$ and in the unphysically short-lived correlations in $C(t)$. At longer times, $\langle R(t)\rangle$ and $C(t)$, obtained from simulations, deviate from their theoretical reference curves due to a soft harmonic spring and insufficient sampling.

B. Computational Performance. We benchmarked the computational efficiency of the obtained GPU-based realizations of the Ran2, Hybrid Taus, and Lagged Fibonacci algorithms using Langevin simulations of N Brownian oscillators in three dimensions. For each system size N, we ran one trajectory for 10^6 simulation steps. All N threads were synchronized at the end of each step to emulate an LD simulation run of a biomolecule on a GPU. The execution time and memory usage are profiled in Figure 3. We find that Ran2 is the most demanding generator. Using Ran2 for a system of $N = 10^4$ particles requires an additional ~264 h of wall-clock time to obtain a single trajectory over 10^9 steps. The memory demand for Ran2 is quite high, i.e., >250 MB for $N = 10^6$ (Figure 3b). Because in biomolecular simulations a large memory area is needed to store parameters of the force field, Verlet lists, interparticle distances, and so forth, the high memory demand might prevent one from using Ran2 in simulations of a large system. Also, implementing Ran2 in Langevin simulations on the GPU does not lead to a substantial speedup (Figure 3a). By contrast, Hybrid Taus and Lagged Fibonacci RNGs are light and fast in terms of the memory usage and the execution time (Figure 3). These generators require a small amount of memory, i.e., <15−20 MB, even for a large system of $N = 10^6$ particles (Figure 3b).

In general, the number of memory calls scales linearly with N. Because on a GPU the computational speed even of a fast RNG is determined by the number of memory calls, multiple

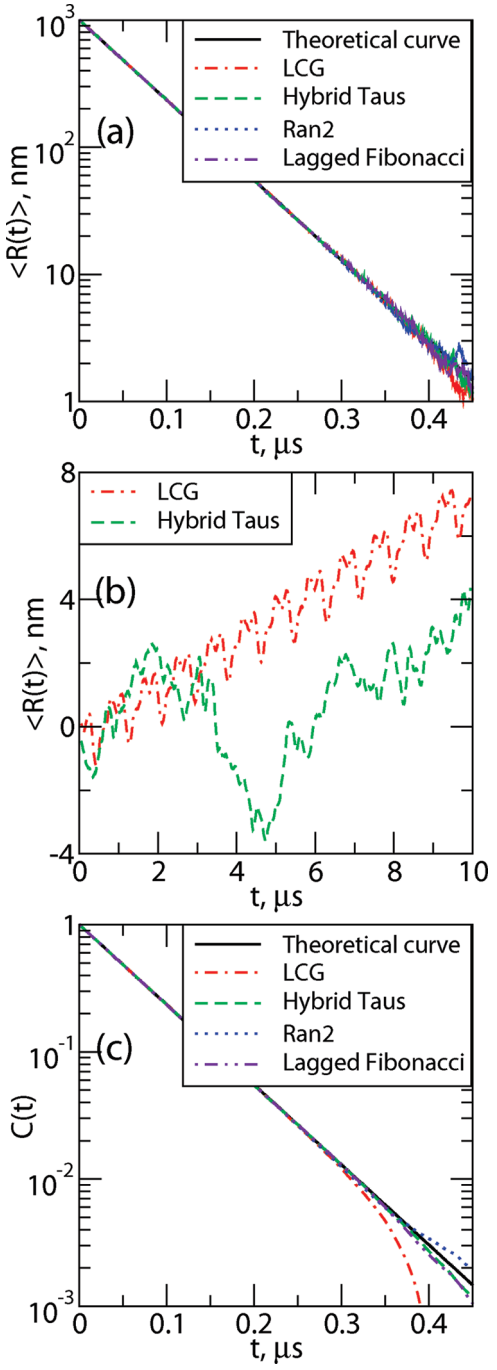


Figure 2. The average particle position $\langle R(t)\rangle$ (a,b) and two-point correlation function $C(t)$ (c) for a system of $N = 10^4$ Brownian oscillators. Theoretical curves of $\langle R(t)\rangle$ and $C(t)$ are compared with the simulation results obtained using the LCG, Hybrid Taus, Ran2, and Lagged Fibonacci algorithms. Equilibrium fluctuations in $\langle R(t)\rangle$ in a longer time scale, obtained using LCG, are magnified in panel b, where one can observe a repeated pattern due to correlations among N streams of random numbers.

reads/writes from/to the GPU global memory can prolong the computational time. We profiled the LCG, Ran2, Hybrid Taus, and Lagged Fibonacci RNGs, which use, respectively, 1, 40, 4, and ~3 state variables per thread, in terms of the number of memory calls per simulation step. The state size for Lagged Fibonacci depends on the choice of ll and sl (Appendix A). LCG,

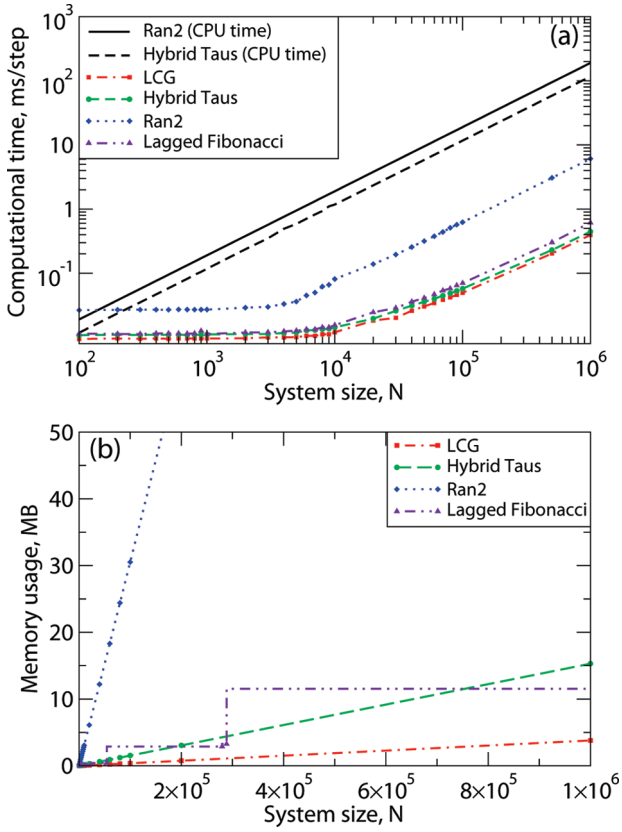


Figure 3. The computational performance of LCG, and Ran2, Hybrid Taus, and Lagged Fibonacci algorithms in Langevin simulations of N Brownian oscillators in three dimensions on the GPU. (a) The execution time threads have been synchronized on the GPU at the end of each step to imitate an LD simulation run of a biomolecule. As a reference, we display the CPU time for Langevin simulations with Ran2 and Hybrid Taus generators. (b) The memory demand, i.e., the amount of memory needed for an RNG to store its current state. Step-wise increases in the memory usage for Lagged Fibonacci are due to the change of constant parameters (Table 1 in the SI).

Hybrid Taus, and Lagged Fibonacci use 4−16 bytes/thread, which is quite reasonable even for a large system of $N = 10^6$ particles. However, Ran2 requires 280 bytes/thread, which is significant (Table 1). Since Ran2 has large size of the state, saving/updating its current state using the GPU local or shared memory is not efficient computationally. Also, Ran2 employs long 64-bit variables, which doubles the amount of data (memory), and requires 4/4 read/write memory calls (7/7 read/write calls are needed to generate four random numbers). Hybrid Taus uses the GPU global memory only when it is initialized, and when it updates its current state. Since it uses four state variables, 4/4 read/write calls per thread are required regardless of the amount of random numbers (Table 1). Lagged Fibonacci uses two random seeds, which results in 2/1 read/write calls per random number (8/4 read/write calls for four random numbers). The execution time for the Hybrid Taus and Lagged Fibonacci RNGs scales sublinearly with N for $N < 10^4$ particles due to insufficient parallelization of the GPU device, but grows linearly for larger systems when all ALUs on the GPU become fully subscribed (Figure 4). It takes about the same time to generate random numbers using these generators and to propagate LD to the next step (Figure 4). This is a high performance level given the fact that the potential function does not involve long-range interactions.

Table 1. Memory Usage (in bytes/thread) and the Number of GPU Global Memory Calls, i.e., the Numbers of Read/Write Operations Per One Random Number (M_1) and for Four Random Numbers (M_2), for the LCG, Hybrid Taus, Ran2, and Lagged Fibonacci Algorithms[a]

parameter	LCG	Hybrid Taus	Ran2	Lagged Fibonacci
bytes/thread	4	16	280	12
M_1	1/1	4/4	4/4	3/1
M_2	1/1	4/4	7/7	12/4

[a] Four random numbers are needed per particle to generate three components of the Gaussian random force.

V. DYNAMIC FORCE MEASUREMENTS *IN SILICO*

A. The SOP Model. We employed the Hybrid Taus and additive Lagged Fibonacci generators to develop the GPU-based implementation of Langevin simulations using a C_α-based coarse-grained self-organized polymer (SOP) model of a protein chain.[14] All the steps of the algorithm have been converted into a CUDA code (SOP-GPU package).[38] The SOP model, briefly reviewed in Appendix B, was developed to describe the mechanical properties of proteins. Steered molecular dynamics (SMD) simulations are currently limited to a 10−50 nm length scale and 0.1−2.5 μs duration.[1,40,41] Hence, it is virtually impossible to resolve the micromechanics of biological assemblies under the experimentally relevant force-loads ($v_f \approx 0.1-10$ μm/s) in the experimental subsecond time scale using all-atom MD methods.[38,42] Computational approaches based on elastic network models allow mostly for the theoretical exploration of equilibrium properties of biomolecules.[43,44]

The physical basis for using the SOP model is the following. First, atomic force microscopy (AFM) experiments cannot resolve biomolecular structures on length scales shorter than ~1 nm, and the typical radius of the cantilever tips used in AFM is $R \gg 1$ nm. Second, a force-driven mechanical reaction of a large-size biomolecule to an external perturbation occurs through sequential unraveling of large fragments and even entire blocks of the secondary structure elements. This allows us to use simplified coarse-grained descriptions of capsomers forming HK97, which involve averaging over irrelevant degrees of freedom that cannot be fully resolved experimentally. Finally, topology and arrangement of the secondary structure elements into the overall capsomer and the whole shell structure, not atomic details, govern the large-scale conformational transitions in viral capsids.[38] The structure-based SOP model, used here, preserves topology and keeps only one interaction site (C_α-bead) per each amino acid residue. The energy function used in the SOP model includes chain connectivity and noncovalent interactions that stabilize the native state. The SOP model has been shown to describe well the mechanical properties of proteins including the green fluorescent protein,[45] the tubulin dimer,[46] and kinesin.[47] This model has also been used to explore the kinetics and to resolve the free energy landscape of tetrahymena ribozyme,[14] riboswitch aptamers,[48] GroEL,[49] protein kinase A,[50] and myosin V.[51] In this paper, we employed the SOP model and Langevin simulations, fully implemented on a GPU, to carry out single-molecule dynamic force measurements *in silico* of the mechanical indentation of the bacteriophage HK97.

B. Forced Indentation of the Bacteriophage HK97. HK97 ($N = 115\,140$ residues)[18] is made of 420 copies of the gp5 protein[17] and is formed by 60 icosahedral units, each composed of 7 domains A−G. Domains A−F form 60 hexamers, and domain G binds to 5 G chains to form 12 pentamers. Each subunit is joined to two of its neighbors by ligation of Lys169 to Asp356, which results in the formation of the topologically linked protein rings (catenanes). The capsid outer radius is $X \approx 32$ nm, and the average wall thickness is $\Delta X \approx 2.1$ nm (in the head II state). The HK97 maturation involves pressure-induced capsid expansion due to the dsDNA packaging.[18] We probed the mechanical reaction of the bacteriophage HK97 in the head II state (Protein Data Bank (PDB) code: 2FT1) by indenting it with the cantilever tip. The tip exerts the time-dependent mechanical force $f(t) = r_f t$, where $r_f = \kappa v_f$ is the force-loading rate, and κ and v_f are, respectively, the cantilever spring constant and the tip velocity. We analyzed the dependence of the physical properties of HK97 on the rate of change r_f and geometry of mechanical perturbation. The effect of geometry was studied using the spherical tip of different radius R.

To obtain LD, we numerically integrated eq 1 for each residue position R_i. To mimic the force−indentation measurements, in each simulation run, the bottom portion of HK97 was constrained on the surface, and a time-dependent force $\mathbf{f}(t) = f(t)\mathbf{n}$ with the magnitude $f(t) = r_f t$ was applied to the spherical tip in the

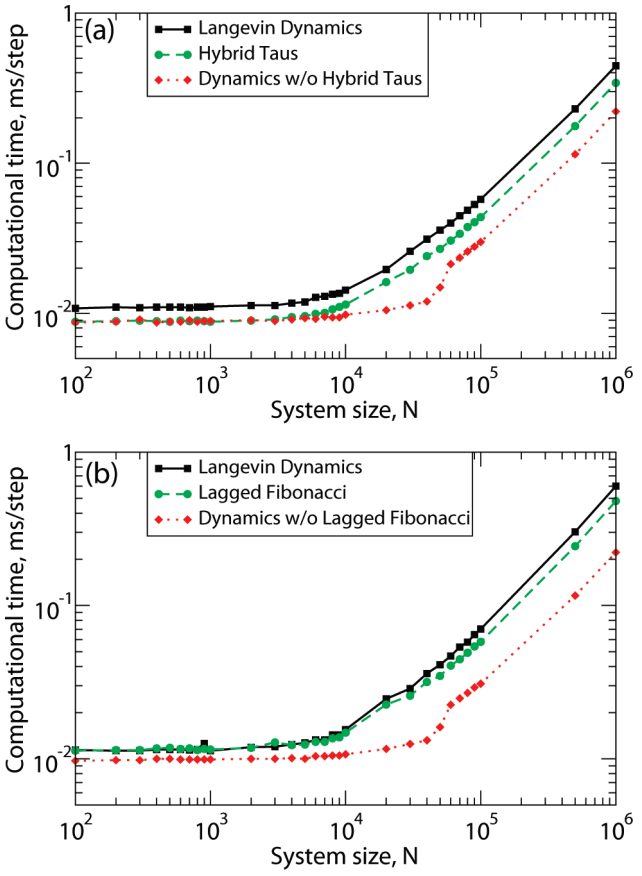


Figure 4. The computational time for the GPU-based implementation of Langevin simulations of N Brownian oscillators in three dimensions using Hybrid Taus RNG (a) and Lagged Fibonacci RNG (b). The simulation time for LD is compared with the time for generating random numbers using Hybrid Taus RNG or Lagged Fibonacci RNG, and with the time required to obtain deterministic (Newtonian) dynamics without random numbers. The associated computational speedup is displayed in Figure 1b.

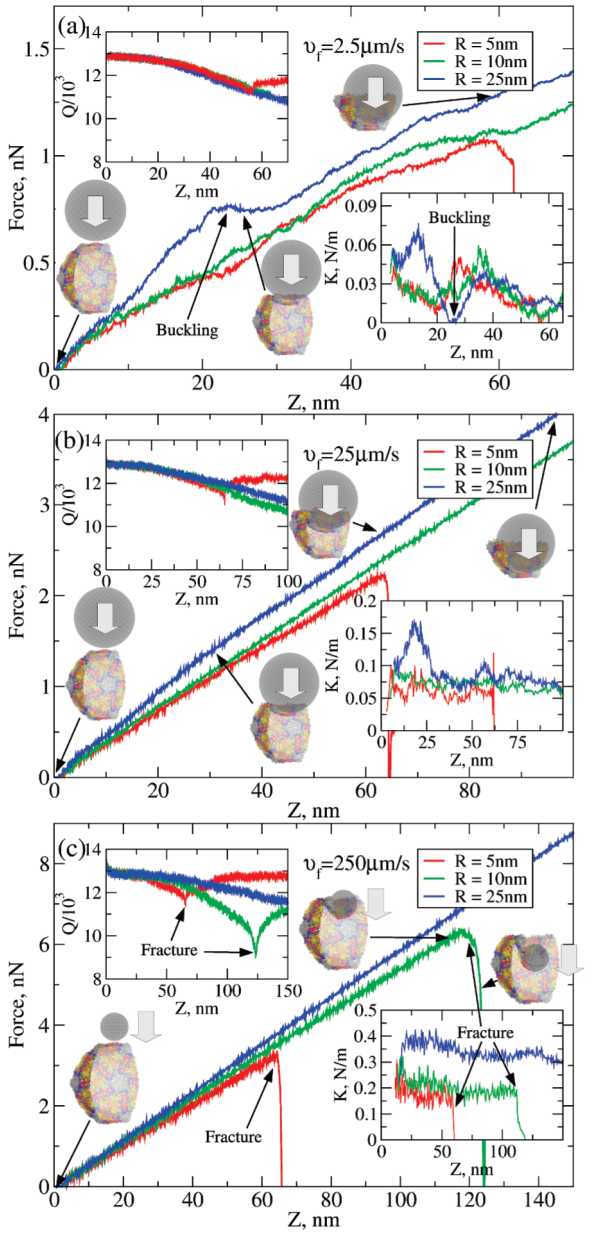


Figure 5. The force−indentation profiles showing the dependence of force on the cantilever displacement Z (FZ curves) for the bacteriophage HK97, obtained using Langevin simulations and Lagged Fibonacci RNG fully implemented on the GPU. To mimic the dynamic force measurements *in vitro*, the capsid was indented by using a spherical tip (gray balls) of radius R = 5, 10, and 25 nm. The cantilever with the spring constant κ = 50 N/m is moving downward in the direction shown by the gray arrows, approaching the viral shell with the constant velocity v_f = 2.5 μm/s (a), 25 μm/s (b), and 250 μm/s (c). Also presented are the transient structures formed in the course of a single indentation trajectory. These show the geometric changes to the HK97 conformation due to the continuous indentation (b), buckling (a), and fracture (c). The insets show the number of native contacts Q and the spring constant of the viral shell K as a function of Z.

direction **n** perpendicular to the outer surface of HK97 (see Figure 5). The Langevin equations of motion were propagated with the time step $\Delta t = 0.08\tau_H = 20$ ps, where $\tau_H = \zeta\epsilon_n\tau_L/k_BT$. Here, $\tau_L = (ma^2/\epsilon_n)^{1/2} = 3$ ps, $a = 3.8$ Å is the $C_\alpha - C_\alpha$ covalent bond distance, $\zeta = 50$ is the dimensionless friction constant for a residue in water ($\xi = \zeta m/\tau_L$), and $m \approx 3 \times 10^{-22}$ g is the residue mass. Because of the presence of catenanes in the HK97 structure, we assumed that the strengths of the intracapsomer contacts and the intercapsomer contacts are roughly equal (Appendix B). Langevin simulations were carried out at room temperature using the bulk water viscosity, which corresponds to the friction coefficient $\eta = 7.0 \times 10^5$ pN ps/nm.

We generated the force−indentation curves (FZ curves) to quantify the mechanical response of HK97 as a function of the distance traveled by the cantilever Z.[52,53] We varied the speed and radius of the cantilever tip by setting v_f = 2.5 μm/s (in the experimental range[54]), 25 μm/s, and 250 μm/s, and using R = 5, 10, and 25 nm. For each set of values of v_f and R, we generated three force−indentation curves. It took ∼34 days (10^9 steps) and ∼3.4 days (10^8 steps) of wall-clock time to generate a singe indentation trajectory of length 20 ms and 2 ms, using v_f = 2.5 and 25 μm/s, respectively, on the GPU GeForce GTX 480. For comparison, it would take ∼120 and ∼12 months, respectively, to complete the same jobs on the CPU Intel Core i7 930. The typical FZ curves, the number of native contacts Q, and the capsid spring constant K are displayed in Figure 5. We estimated the values of K, which quantifies the elastic component of the mechanical response of HK97, using the formula $1/K_{FZ} = 1/\kappa + 1/K$[52,55] for the spring constant for the combined system (capsid plus tip), K_{FZ}, extracted from the FZ curves.

The mechanical response of HK97 shows stochastic variation at a slow force-load v_f = 2.5 μm/s (Figure 5a), but it becomes more "deterministic" when v_f is increased (Figure 5b,c). The slope of the FZ curves (proportional to K), while increasing with v_f and R, fluctuates for all values of v_f and R used, which implies that the capsid elasticity is a dynamic, rather than static, property. Interestingly, K increases from 0.01−0.025 pN/nm at v_f = 2.5 μm/s to 0.05−0.075 pN/nm at v_f = 25 μm/s, and to 0.2−0.35 pN/nm at v_f = 250 μm/s (Table 2). This demonstrates that the capsid wall becomes stiffer when indented faster, and implies that the capsid elasticity also depends on the geometry of a force-bearing load. The buckling transitions were observed only at the slowest force-load of v_f = 2.5 μm/s (Figure 5a) when a large 25 nm tip was used. When the capsid buckles, K first increases and then decreases with Z, while the number of native contacts Q, stabilizing the shell structure, decreases monotonically with Z. The buckling transition sets in at $Z \approx 25$ nm, at which point K drops to zero, signifying loss of mechanical resistance (Figure 5a). At v_f = 2.5 μm/s and for R = 10 nm, the indentation is monotonic (no buckling); however, at v_f = 2.5 μm/s and for R = 5 nm, the indentation continues up to $Z \approx 55$ nm, at which point the mechanical fracture occurs (Figure 5a). The fracture is localized to the area where the tip penetrates the viral shell, and is associated with partial unfolding and disruption of some of the native contacts, which is also reflected in the sudden drop in Q (Figure 5a). The native contacts form again as the tip passes through the capsid wall, resulting in the increase in Q.

The buckling transitions were not detected at the faster loads v_f = 25 μm/s and 250 μm/s (Figures 5b and c). At v_f = 25 μm/s, the dependence of the mechanical reaction on Z is monotonic only for a large 10 nm and 25 nm tip. For a small 5 nm tip, the gradual indentation is interrupted by the capsid fracture at $Z \approx$ 65 nm. This results in a loss of capsid elasticity (Figure 5b), which is reflected in a sudden drop in Q, and in the decrease of K to zero (Figure 5b). At v_f = 250 μm/s, the FZ curves were monotonic only when a large 25 nm tip was used. Indenting with smaller 5 nm and 10 nm tips resulted in the capsid fracture at $Z \approx$ 65 nm

Table 2. Average Parameters Characterizing the Micromechanical Properties of the Bacteriophage HK97: Spring Constant K, Young's Modulus Y, Critical Pressure p_c, and Energy Change Due to Buckling ΔE_b and Fracture ΔE_f[a]

v_f, μm/s	K, N/m	Y, MPa	p_c, MPa	$\Delta E_b/10^{-17}$, Nm	$\Delta E_f/10^{-17}$, Nm
2.5	0.01−0.025	64−160	6 (5 nm)	1.4 (25 nm)	2.7 (5 nm)
25	0.05−0.075	320−480	9 (5 nm)		3.0 (5 nm)
250	0.2−0.35	1280−2230	13 (5 nm), 18 (10 nm)		3.4 (5 nm), 5.7 (10 nm)

[a] Values of R are given in parentheses.

and $Z \approx 120$ nm, respectively (Figure 5c). A small structural damage was localized to the residue positions affected by the tip moving downward, and the recovery of the native contacts was only partial for $R = 10$ nm, but full for $R = 5$ nm (Figure 5c). The structural analysis of bacteriophage HK97 has revealed that the ratio of the wall thickness to the outer radius is $\Delta X/X \approx 0.065 \ll 1$. This allowed us to use the thin-shell approximation to connect the spring constant K with the Young's modulus Y, using the formula $K = \alpha Y \Delta X^2/X$, where α is the proportionality factor.[56] Assuming that $\alpha \approx 1$, we estimated the modulus Y, which characterizes the "in-plane" elasticity of the viral shell. We evaluated the energy costs for the structural damage, i.e., for the formation of a spherical cavity, ΔE_f, and for buckling, ΔE_b. We also calculated the critical pressure $p_c = f_c/A$, where A is the contact area on the capsid outer surface impacted by the cantilever tip. The numerical values of Y, p_c, ΔE_b, and ΔE_f are accumulated in Table 2.

VI. DISCUSSION

A. Choosing RNG for GPU-Based Computations. RNGs are used in many computer applications such as simulations of stochastic systems, probabilistic algorithms, and numerical analysis among many others. The highly parallel architecture of a GPU provides an alternative computational platform that allows one to utilize multiple ALUs on a single processor. This comes at a price of having smaller cache memory and reduced flow control. Hence, to harvest raw computational power offered by the GPU, one needs to redesign computational algorithms that have been used on the CPU for many decades. Here, we described two general approaches to generating pseudorandom numbers on the GPU. In the one-RNG-per-thread approach, the same RNG algorithm is executed in each computational thread (for each particle), a procedure used in the CPU-based methods. In the one-RNG-for-all-threads setting, one can utilize the ability of different threads to communicate across an entire GPU device. These methods were used to develop the GPU-based realizations of the Ran2, and Hybrid Taus generators (Figure 2 in the SI), and the additive Lagged Fibonacci RNG (Figure 3 in the SI). The Hybrid Taus and Lagged Fibonacci generators provide random numbers at a computational speed almost equal to that of the LCG, and the associated memory demand is low (Figure 3). Their long periods are sufficient to describe the dynamics of a large system ($N > 10^6$ particles) over $>10^9$ simulation steps. Ran2 is a well tested generator of proven statistical quality,[13] but it works only ∼10−15-times faster on the GPU and requires a large memory area (Figure 3). By contrast, employing the Hybrid Taus and Lagged Fibonacci algorithms results in an impressive 200−250-fold speedup (Figure 1).

As an application-based test of randomness, we carried out Langevin simulations of N Brownian oscillators (Ornstein−Uhlenbeck process). We found an excellent agreement between the stochastic trajectories, obtained analytically and computationally, by using the Hybrid Taus, Ran2, and Lagged Fibonacci algorithms (Figure 2). We also applied stringent statistical tests to access the statistical properties of the random numbers produced by using the GPU-based implementation of the Hybrid Taus and Lagged Fibonacci RNGs developed. We found that Hybrid Taus does not fail a single tests in the DIEHARD test suite[24] and passes BigCrush and SmallCrush tests in the TestUO1 package.[12] Lagged Fibonacci, even with a small short lag sl = 1252, does not fail any test in DIEHARD, and passes BigCrush in TestU01. We recommend these generators for Langevin simulations, MC simulations, and MD simulations in implicit solvent of large biomolecular systems. Given their high statistical quality, these RNGs is a reasonable choice for GPU-based simulations of biomolecules.

B. Dynamic Signatures of the Force−Indentation Spectra. We utilized the structure-based coarse-grained description of proteins to carry out single-molecule forced indentation experiments *in silico* of the bacteriophage HK97. The 180-fold computational acceleration achieved on the GPU GeForce GTX 480, compared to the heavily tuned CPU version of the same program (Figure 1), allowed us to explore the physical properties of this biological assembly (10^5 particles) in the subsecond time scale.[38] We used experimental force-loading conditions (force-ramp), employed in the AFM-based dynamic force measurements, including the cantilever spring constant κ, and the spherical tip size R and velocity v_f. We found that the microscopic mechanical response of the virion HK97 depends rather sensitively on the rate and geometry of force application.

We observed a whole spectrum of biomechanical reactions in the far-from-equilibrium regime from gradual indentation at low and moderately high forces to buckling at intermediate forces, and to mechanical fracture at high forces (Figure 5). These dynamic signatures in the theoretical force spectra might reflect the general physical properties shared by many virus shells. We found that virus shell elasticity is a fluctuating dynamic property, which varies with the rate of change of the mechanical perturbation. The spring constant of ∼0.01−0.02 N/m for the bacteriophage HK97, observed at the experimental pulling speed $v_f = 2.5$ μm/s used in AFM, agrees with the experimental estimates of this parameter for empty viral shells.[55] Our finding that the spring constant of a virus shell (K) might change with size of a load-bearing tip implies that K is a local mechanical characteristic. Indeed, the larger the tip, the more structural units must cooperate to withstand the external mechanical stress. Hence, larger tips comparable with the dimensions of the viral shell in question should be used to average over local variations of the mechanical properties. In addition, K might vary depending on where on the shell surface the tip presses against the virus shell,[42] but we leave this aspect for future studies.

We found that temporary loss of virus shell elasticity, when K rapidly decreases to zero, might occur due to buckling or mechanical fracture. In the event of buckling, the capsid shell rapidly regains its elasticity, which results in the subsequent increase in

K. A sudden drop in K indicates, rather, the onset of the mechanical fracture due to the structural damage, associated with partial disruption of the network of native contacts, and unfolding transitions on a short length scale (3−5 nm). This process is reversible, as the native contacts tend to reform after the cantilever tip has passed through the capsid wall (Figure 5). These results agree well with the experimental observations on other virions.[52,55] We observed the expected crossover from the elastic behavior at low forces to the plastic behavior at higher forces,[42] which also follows from the decrease of K at longer Z values, but this effect is not so well-pronounced, which might be due to the presence of the topological links.[57] This rare feature of the molecular architecture of bacteriophage HK97 seems to enhance its elastic properties. In fact, the observed sudden drops in Q were mostly due to the disruption of the intracapsomer contacts, which stabilize the capsid structural units, rather than the intercapsomer contacts.

The onset of buckling is controlled by a universal physical characteristic: the Foppl−von Kármán (FvK) number γ.[58] For a thin spherical shell, it is defined as $\gamma = YX^2/k$, where k is the "out-of-plane" bending modulus. For a buckling transition to occur in a shell of radius X, the ratio of the "in-plane" stretching (YX^2) to the "out-of-plane" bending (k) must be large so that γ exceeds some critical value $\sim 10^3$.[58] The results obtained for $v_f = 2.5\ \mu$m/s show that the buckling regime sets in when the capsid is indented with a large tip, i.e., when $R \sim X$ (Figure 5a). In this case, the tip excites mostly the in-plane stretching modes, and $YX^2 \gg k$. On the other hand, the fracture occurs when a smaller tip is used, i.e., when $R < X$. Here, the tip motion excites the out-of-plane bending modes, and $YX^2 \ll k$. Hence, both dynamic regimes can be accessed by controlling the geometry of the force application. The results obtained for faster force-loads (v_f = 25 and 250 μm/s) indicate strongly that whether mechanical failure (buckling or fracture) occurs also depends on the rate of change of an applied force $f(t)$ (Figure 5b,c). Hence, theoretical models of viral shell mechanics should be extended to account for dynamic coupling of the in-plane modes and the out-of-plane modes of motion, and for the far-from-equilibrium conditions of propagation and distribution of the mechanical stress on the spherical surface.

The Young's modulus Y was found to depend on the rate of change and geometry of an applied force (Table 2). At the experimental value of v_f = 2.5 μm/s, the modulus Y = 60−160 MPa for HK97 is comparable with Y = 140 MPa for the empty shell CCMV,[55] but is less than Y = 1.8 GPa for the bacteriophage ϕ29[52] (Table 2). The empty shell HK97 is capable of withstanding the mechanical pressures on the order of 60−140 atm, which is comparable with the effective pressure inside the bacteriophage ϕ29 due to DNA packaging. The results obtained show that dynamic force assays *in silico* can be used to explore the limits of elasticity of virus shells and to predict the maximum internal pressure. We found that with the tip-sphere moving at $v_f = 2.5\ \mu$m/s, the energy cost associated with the local fracture (spherical cavity of radius 5 nm) is $\Delta E_f = 2.7 \times 10^{-17}$ Nm, which is roughly twice the energy required to buckle the capsid ($\Delta E_b = 1.4 \times 10^{-17}$ Nm). We also found that ΔE_f grows with v_f (Table 2).

VII. CONCLUSION

The development of new Fermi architecture (NVIDIA) and Larrabee architecture (Intel) is an important step for general purpose GPU computing. The high-speed interconnection network will provide a fast interface for threads communication. These advances will enable the programmer to distribute a computational workload among many cores on a GPU more efficiently, and to reach an even higher performance level. In this regard, the developed GPU-based implementation of additive Lagged Fibonacci RNG can be ported to new graphics processors with minor modifications. In the context of biomolecular simulations, this will make it possible to compute random forces using thread synchronization over an entire GPU device. This makes the one-RNG-for-all-threads method of generation of pseudo-random numbers on the GPU, where thread synchronization is utilized, all the more important. This approach can also be used to develop GPU-based implementations of Mersenne Twister algorithm[59] and several other algorithms, including multiple recursive generator (MRG) and linear/generalized shift feedback register (LSFR/GSFR) generators, such as the 4-lag Lagged Fibonacci algorithm.[12,33]

Continuum mechanics approaches, such as finite element analysis, provides valuable information about the average macroscopic parameters characterizing the mechanical properties of the protein shells of many viruses. The developed GPU-based realizations of several RNG algorithms open a new dimension in the theoretical exploration of viral capsids, as they enable one to follow stochastic dynamics of viral shells in the experimental centisecond time scale. Moreover, the presented formalism allows one to go beyond the ensemble-average picture and to resolve the entire distributions of the relevant molecular characteristics. The structure-based coarse-grained SOP model can also be used to explore the kinetics of nanomechanical transitions in supramolecular assemblies of the biological interest. Understanding the micromechanical properties of nanometer-scale protein shells is important for virus biology, materials engineering, and for nanotechnological applications. Dynamic force measurements *in silico*, described here, which mimic the AFM-based measurements *in vitro*, can be performed in reasonable "GPU-time" using realistic conditions of force application. This permits a direct comparison of experimental and simulation results. Hence, dynamic signatures observed for continuous transitions (indentation), phase transitions (buckling), and structural failure (fracture) can be used to provide meaningful interpretation for the force peaks and kinks in the experimental force spectra.

APPENDIX A: LCG, RAN2, HYBRID TAUS AND LAGGED FIBONACCI ALGORITHMS

LCG. The LCGs use a transitional formula

$$x_n = (ax_{n-1} + c) \bmod m \qquad \text{(A1)}$$

where m is the maximum period, and a = 1664525 and c = 1013904223 are constant parameters.[13] To produce a uniformly distributed random number, x_n is divided by 2^{32}. Assuming a 32-bit integer, the maximum period can be at most $p = 2^{32}$, which is far too low.[12] If $m = 2^{32}$, one can neglect mod m operation as the returned value is low-order 32 bits of the true 64-bit product. Then, the transitional formula reads $x_n = ax_{n-1} + c$, which is the so-called quick and dirty or ranqd2 generator (simplified LCG). Quick and dirty LCG is very fast, as it takes a single multiplication and a single addition to produce a random number, and it uses one integer to describe its current state.

Ran2. Ran2 combines two LCGs and employs randomization using some shuffling procedure.[13] Ran2 has a long period and provides random numbers of very good statistical properties.[12] It is one of a very few generators that does not fail a single statistical test. Ran2 is reasonably fast, but it involves long integer arithmetic (64-bit logic), a computational bottleneck for contemporary GPUs, and it requires a large amount of memory to store its current state.

Hybrid Taus. Hybrid Taus[20] is a combined generator that uses LCG and Tausworthe algorithms. Tausworthe taus88 is a fast equidistributed modulo 2 generator,[30,31] which produces random numbers by generating a sequence of bits from a linear recurrence modulo 2, and forming the resulting number by taking a block of successive bits. In the space of binary vectors, the nth element of a vector is constructed using the linear transformation

$$y_n = a_1 y_{n-1} + a_2 y_{n-2} + \ldots a_k y_{n-k} \qquad \text{(A2)}$$

where a_n are constant coefficients. Given initial values y_0, y_1, ..., y_{n-1}, the nth random integer is obtained as $x_n = \sum_{j=1}^{L} y_{ns+j-1} 2^{-j}$, where s is a positive integer and $L = 32$ is the integer size (machine word size). Computing x_n involves performing s steps of the recurrence, which might be costly computationally. Fast implementation can be obtained for a certain choice of parameters: when $a_k = a_q = a_0 = 1$, where $0 < 2q < k$ and $a_n = 0$ for $0 < s \leq k - q < k \leq L$, the algorithm can be simplified to a series of binary operations.[31] Statistical properties of a combined generator are better than those of its components. When periods of all components are coprime numbers, a period of a combined generator is the product of periods of all components. A similar approach is used in the KISS generator.[33] However, multiple 32-bit multiplications, used in KISS, might harm its performance on the GPU. The period of the Hybrid Taus is the lowest common multiplier of the periods of three Tausworthe steps and one LCG. We used parameters that result in the periods $p_1 \approx 2^{31}$, $p_2 \approx 2^{30}$, and $p_3 \approx 2^{28}$ for the Tausworthe generators and the period $p_4 = 2^{32}$ for the LCG, which makes the period of the combined generator equal $\sim 2^{121} > 10^{36}$. Hybrid Taus uses small memory area since only four integers are needed to store its current state.

Lagged Fibonacci. The Lagged Fibonacci algorithm is defined by the recursive relation,

$$x_n = f(x_{n-\mathrm{sl}}, x_{n-\mathrm{ll}}) \bmod m \qquad \text{(A3)}$$

where sl and ll are the short lag and the long lag, respectively (ll > sl), m defines the maximum period, and f is a function that takes two integers $x_{n-\mathrm{sl}}$ and $x_{n-\mathrm{ll}}$ to produce integer x_n. The most commonly used functions are multiplication, $f(x_{n-\mathrm{sl}}, x_{n-\mathrm{ll}}) = x_{n-\mathrm{sl}} \times x_{n-\mathrm{ll}}$ (multiplicative Lagged Fibonacci), and addition, $f(x_{n-\mathrm{sl}}, x_{n-\mathrm{ll}}) = x_{n-\mathrm{sl}} + x_{n-\mathrm{ll}}$ (additive Lagged Fibonacci). Random numbers are generated from the initial set of ll integer seeds. To achieve the maximum period $\sim 2^{\mathrm{ll}-1} \times m$, ll should be set equal the base of a Mersenne exponent, and sl should be taken so that the characteristic polynomial $x^{\mathrm{ll}} + x^{\mathrm{sl}} + 1$ is primitive. Also, sl should not be too small nor too close to ll. It is recommended that sl $\approx \rho \times$ ll, where $\rho \approx 0.618$.[12] When single precision arithmetic is used, the mod m operation can be omitted by setting $m = 2^{32}$.

APPENDIX B: SOP MODEL

In the SOP model,[14−16] each residue is described by a single interaction center (C_α-atom). The potential energy function of a protein conformation U_{SOP}, specified in terms of the coordinates $\{r\} = r_1, r_2, \ldots, r_N$, is given by

$$\begin{aligned} U_{\mathrm{SOP}} &= U_{\mathrm{FENE}} + U_{\mathrm{NB}}^{\mathrm{ATT}} + U_{\mathrm{NB}}^{\mathrm{REP}} \\ &= -\sum_{i=1}^{N-1} \frac{k}{2} {R_0}^2 \log\left(1 - \frac{(r_{i,i+1} - r^0_{i,i+1})^2}{{R_0}^2}\right) \\ &+ \sum_{i=1}^{N-3} \sum_{j=i+3}^{N} \varepsilon_n \left[\left(\frac{r^0_{ij}}{r_{ij}}\right)^{12} - 2\left(\frac{r^0_{ij}}{r_{ij}}\right)^{6}\right] \Delta_{ij} \\ &+ \sum_{i=1}^{N-2} \sum_{j=i+2}^{N} \varepsilon_r \left(\frac{\sigma_{ij}}{r_{ij}}\right)^6 + \sum_{i=1}^{N-3} \sum_{j=i+3}^{N} \varepsilon_r \left(\frac{\sigma}{r_{ij}}\right)^6 (1 - \Delta_{ij}) \end{aligned} \qquad \text{(B1)}$$

For a detailed account of the numerical values of the molecular parameters entering eq B1, the reader should consult ref 14. To summarize, in eq B1, the finite extensible nonlinear elastic (FENE) potential U_{FENE} with the spring constant $k = 14$ N/m describes the backbone chain potential. The distance between residues i and $i+1$ is $r_{i,i+1}$, $r^0_{i,i+1}$ is its value in the native (PDB) structure, and $R_0 = 2$ Å is the tolerance in the change of a covalent bond distance. Here, R_0 and k account for chain connectivity.[14−16] We used the Lennard-Jones potential $U_{\mathrm{NB}}^{\mathrm{ATT}}$ to account for the noncovalent interactions that stabilize the native state. We assumed that if the noncovalently linked residues i and j ($|i - j| > 2$) are within the cutoff $R_C = 8$ Å, then $\Delta_{ij} = 1$, and zero otherwise. The range of possible values for R_C is dictated by the protein structures in the PDB. Typically, the distance between two amino acids forming a native contact, which stabilizes the native folded state of a protein, is ~8 Å. The value of ε_n (= 1.5 kcal/mol) quantifies the strength of the nonbonded interactions. All the non-native interactions in the potential $U_{\mathrm{NB}}^{\mathrm{REP}}$ are treated as repulsive. An additional constraint is imposed on the bond angle formed by residues i, $i+1$, and $i+2$ by including the repulsive potential with parameters $\varepsilon_r = 1$ kcal/mol and $\sigma_{i,i+2} = 3.8$ Å, which determine the strength and the range of repulsion. To ensure self-avoidance of a protein chain, we set $\sigma = 3.8$ Å .

ASSOCIATED CONTENT

Supporting Information. The one-RNG-per-thread approach and the one-RNG-for-all-threads approach are exemplified further. We also present pseudocodes for our GPU-based implementations of the Hybrid Taus, Ran2, and additive Lagged Fibonacci algorithms. This information is available free of charge via the Internet at http://pubs.acs.org.

AUTHOR INFORMATION

Corresponding Author

*Phone: 978-934-3661; fax: 978-934-3013; e-mail: Valeri_Barsegov@uml.edu.

ACKNOWLEDGMENT

Acknowledgement is made to the donors of the American Chemical Society Petroleum Research Fund (Grant PRF #47624-G6) for partial support of this research (V.B.). This work was also supported in part by Grant #09-0712132 from the

Russian Foundation for Basic Research, and by Grant #02-740-11-5126 from the Russian Ministry of Education (V.B. and Y.K.).

REFERENCES

(1) Stone, J. E.; Phillips, J. C.; Freddolino, P. L.; Hardy, D. J.; Trabuco, L. G.; Schulten, K. *J. Comput. Chem.* **2007,** *28,* 2618–2640.

(2) Friedrichs, M. S.; Eastman, P.; Vaidyanathan, V.; Houston, M.; Legrand, S.; Beberg, A. L.; Ensign, D. L.; Bruins, C. M.; Pande, V. S. *J. Comput. Chem.* **2009,** *30,* 864–872.

(3) Anderson, J. A.; Lorentz, C. D.; Travesset, A. *J. Comput. Phys.* **2008,** *227,* 5342–5359.

(4) van Meel, J. A.; Arnold, A.; Frenkel, D.; Zwart, S. F. P.; Belleman, R. *Mol. Simul.* **2008,** *34,* 259–266.

(5) Harvey, M. J.; Fabritilis, G. D. *J. Chem. Theory Comput.* **2009,** *5,* 2371–2377.

(6) Anderson, A. G.; W., A. G., III; Schröder, P. *Comput. Phys. Commun.* **2007,** *177,* 298–306.

(7) Yang, J.; Wang, Y.; Chen, Y. *J. Comput. Phys.* **2007,** *221,* 799–804.

(8) Kirk, D. B.; Hwu, W.-M. W. *Programming Massively Parallel Processors. A Hands-On Approach*; Morgan Kaufmann: Burlington, MA, 2010.

(9) Brooks, B. R.; Bruccoleri, R. E.; Olafson, B. D.; States, D. J.; Swaminathan, S.; Karplus, M. *J. Comput. Chem.* **1983,** *4,* 187–217.

(10) Haberthür, U.; Caflisch, A. *J. Comput. Chem.* **2008,** *29,* 701–715.

(11) Doi, M.; Edwards, S. F. *The Theory of Polymer Dynamics*; International Series of Monographs on Physics; Oxford Science Publications: Oxford, U.K., 1988.

(12) L'Ecuyer, P.; Simard, R. *ACM Trans. Math. Software* **2007,** *33,* 22.

(13) Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P. *Numerical Recipes in C - The Art of Scientific Computing*, 2nd ed.; Cambridge University Press: New York, 1992.

(14) Hyeon, C.; Dima, R. I.; Thirumalai, D. *Structure* **2006,** *14,* 1633–1645.

(15) M. Mickler, R. I. D.; Dietz, H.; Hyeon, C.; Thirumalai, D.; Rief, M. *Proc. Natl. Acad. Sci. U.S.A.* **2007,** *104,* 20268–20273.

(16) Dima, R. I.; Joshi, H. *Proc. Natl. Acad. Sci. U.S.A.* **2008,** *105,* 15743–15748.

(17) Gertsman, I.; Gan, L.; Guttman, M.; Lee, K.; Speir, J. A.; Duda, R. L.; Hendrix, R. W.; Komives, E. A.; Johnson, J. E. *Nature* **2009,** *458,* 646–650.

(18) Steven, A. C.; B., H. J.; Cheng, N.; Trus, B. L.; Conway, J. F. *Curr. Opin. Struct. Biol.* **2005,** *15,* 227–236.

(19) Vlad, R. A. J. D. H.; Bahar, I. *Structure* **2005,** *13,* 413–421.

(20) *GPU Gems 3*; Nguyen, H., Ed.; Addison-Wesley: Boston, MA, 2008.

(21) Tsang, W. W.; Marsaglia, G. *J. Stat. Software* **2000,** *5,* 1–7.

(22) Marsaglia, G.; Bray, T. A. *SIAM Rev.* **1964,** *6,* 260–264.

(23) Box, G. E. P.; Miller, M. E. *Ann. Math. Stat.* **1958,** *29,* 610–611.

(24) Marsaglia, G. *DIEHARD: A Battery of Tests of Randomness*: Florida State University: Tallahassee, FL, 1996 (see: http://stat.fsu.edu/geo/diehard.html) (May 1, 2010).

(25) Mascagni, M.; Srinivasan, A. *ACM Trans. Math. Software* **2000,** *26,* 436–461.

(26) Soto, J. Statistical testing of random number generators. *Proceedings of the 22nd National Information Systems Security Conference*; NIST: Gaithersburg, MD, 1999 (see: http://csrc.nist.gov/rng/).

(27) Selke, W.; Talapov, A. L.; Shchur, L. N. *JETP Lett.* **1993,** *58,* 665–668.

(28) Grassberger, P. *Phys. Lett. A* **1993,** *181,* 43–46.

(29) Ferrenberg, A. M.; Landau, D. P.; Wong, Y. J. *Phys. Rev. Lett.* **1992,** *69,* 3382–3384.

(30) Tausworthe, R. C. *Math. Comput.* **1965,** *19,* 201–209.

(31) L'Ecuyer, P. *Math. Comput.* **1996,** *65,* 203–213.

(32) Mascagni, M.; Srinivasan, A. *Parallel Comput.* **2004,** *30,* 899–916.

(33) Marsaglia, G. Random numbers for C: The END? Posted on sci.crypt Google group, 1999.

(34) L'Ecuyer, P.; Blouin, F.; Couture, R. *ACM Trans. Model. Comput. Simul.* **1993,** *3,* 87–98.

(35) Veitshans, T.; Klimov, D.; Thirumalai, D. *Folding Des.* **1997,** *2,* 1–22.

(36) Barsegov, V.; Klimov, D.; Thirumalai, D. *Biophys. J.* **2006,** *90,* 3827–3841.

(37) Ermak, D. L.; McCammon, J. A. *J. Chem. Phys.* **1978,** *69,* 1352–1360.

(38) Zhmurov, A.; Dima, R. I.; Kholodov, Y.; Barsegov, V. *Proteins* **2010,** *78,* 2984–2999.

(39) Risken, H. *The Fokker—Planck Equation*, 2nd ed.; Springer-Verlag: Berlin, 1989.

(40) Isralewitz, B.; Gao, M.; Schulten, K. *Curr. Opin. Struct. Biol.* **2001,** *11,* 224–230.

(41) Freddolino, P. L.; Liu, F.; Gruebele, M.; Schulten, K. *Biophys. J.* **2008,** *94,* L75–L77.

(42) Zink, M.; Grubmueller, H. *Biophys. J.* **2009,** *96,* 1767–1777.

(43) Phelps, D. K.; Speelman, B.; Post, C. B. *Curr. Opin. Struct. Biol.* **2000,** *10,* 170–173.

(44) Bahar, I.; Rader, A. J. *Curr. Opin. Struct. Biol.* **2005,** *15,* 1–7.

(45) Mickler, M.; Dima, R. I.; Dietz, H.; Hyeon, C.; Thirumalai, D.; Rief, M. *Proc. Natl. Acad. Sci. U.S.A.* **2007,** *104,* 20268–20273.

(46) Dima, R. I.; Joshi, H. *Proc. Natl. Acad. Sci. U.S.A.* **2008,** *105,* 15743–15748.

(47) Hyeon, C.; Onuchic, J. N. *Proc. Natl. Acad. Sci. U.S.A.* **2007,** *104,* 2175–2180.

(48) Lin, J. C.; Thirumalai, D. *J. Am. Chem. Soc.* **2008,** *130,* 14080–14084.

(49) Hyeon, C.; Lorimer, G. H.; Thirumalai, D. *Proc. Natl. Acad. Sci. U.S.A.* **2006,** *103,* 18939–18944.

(50) Hyeon, C.; Jennings, P. A.; Adams, J. A.; Onuchic, J. N. *Proc. Natl. Acad. Sci. U.S.A.* **2009,** *106,* 3023–3028.

(51) Tehver, R.; Thirumalai, D. *Structure* **2010,** *18,* 471–481.

(52) Ivanovska, I. L.; de Pablo, P. J.; Ibarra, B.; Sgalari, G.; MacKintosh, F. C.; Carrascosa, J. L.; Schmidt, C. F.; Wuite, G. J. L. *Proc. Natl. Acad. Sci. U.S.A.* **2004,** *101,* 7600–7605.

(53) Ivanovska, I.; Wuite, G.; Joensson, B.; Evilevitch, A. *Proc. Natl. Acad. Sci. U.S.A.* **2007,** *104,* 9603–9608.

(54) Weisel, J. W. *Science* **2008,** *320,* 456–457.

(55) Michel, J. P.; Ivanovska, I. L.; Gibbons, M. M.; Klug, W. S.; Knobler, C. M.; et al. *Proc. Natl. Acad. Sci. U.S.A.* **2006,** *103,* 6184–6189.

(56) Landau, L. D.; Lifshitz, E. M. *Theory of Elasticity*, 3rd ed.; Pergamon: New York, 1986.

(57) Wikoff, W. R.; Liljas, L.; Duda, R. L.; Tsiruta, H.; Hendrix, R. W.; Johnson, J. E. *Science* **2000,** *289,* 2129–2133.

(58) Lidmar, J.; Mirny, L.; Nelson, D. R. *Phys. Rev. E* **2003,** *68,* 051910–051919.

(59) Matsumoto, M.; Nishimura, T. *ACM Trans. Model. Comput. Simul.* **1998,** *8,* 3–30.