Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 237061, 8 pages
http://dx.doi.org/10.1155/2015/237061

Research Article

A Fading Channel Simulator Implementation Based on GPU Computing Techniques

R. Carrasco-Alvarez,[1] J. Vázquez Castillo,[2] A. Castillo Atoche,[3] and J. Ortegón Aguilar[2]

[1] *Universidad de Guadalajara, Boulevard Marcelino García Barragán 1421, 44430 Guadalajara, JAL, Mexico*
[2] *Universidad de Quintana Roo, Boulevard Bahía s/n, Esquina Ignacio Comonfort, 77019 Chetumal, QRoo, Mexico*
[3] *Universidad Autónoma de Yucatán, Avenida Industrias No Contaminantes, S/N, 97310 Mérida, YUC, Mexico*

Correspondence should be addressed to R. Carrasco-Alvarez; roberto.carrasco@red.cucei.udg.mx

Received 13 September 2014; Revised 11 March 2015; Accepted 11 March 2015

Academic Editor: Chunlin Chen

Channel simulators are powerful tools that permit performance tests of the individual parts of a wireless communication system. This is relevant when new communication algorithms are tested, because it allows us to determine if they fulfill the communications standard requirements. One of these tests consists of evaluating the system performance when a communication channel is considered. In this sense, it is possible to model the channel as an FIR filter with time-varying random coefficients. If the number of coefficients is increased, then a better approach to real scenarios can be achieved; however, in that case, the computational complexity is increased. In order to address this issue, a design methodology for computing the time-varying coefficients of the fading channel simulators using consumer-designed graphic processing units (GPUs) is proposed. With the use of GPUs and the proposed methodology, it is possible for nonspecialized users in parallel computing to accelerate their simulation developments when compared to conventional software. Implementation results show that the proposed approach allows the easy generation of communication channels while reducing the processing time. Finally, GPU-based implementation takes precedence when compared with the CPU-based implementation, due to the scattered nature of the channel.

1. Introduction

Currently, the high demand for integrated services (voice, data, and video) means that new data transmission schemes have to be developed for dealing with high transmission data rates and at the same time for offering high levels of quality of service. The fourth generation (4G) of mobile communication systems is still under development; its main goal is to provide a digital communication network (land, mobile, and satellite) with peak data rates of 100 Mbps for high mobility devices and high data rates of 1 Gbps for users or devices in low mobility environments or stationary conditions. The main technologies used in 4G include techniques based on multiple-input and multiple-output (MIMO) antennas, turbo decoding, adaptive modulation, coding schemes and error correction, and orthogonal FDMA (orthogonal FDMA, OFDM) [1, 2]. Current versions of standards that incorporate 4G are LTE-A (long term evolution-advanced) and IEEE 802.16 m WiMAX (Worldwide Interoperability for Microwave Access) mobile. Therefore, the new issues imposed by the standards require new processing algorithms to be tested on high mobility environments affected by Doppler shifts (time-selective channels) and multipath propagation (frequency-selective channels). The temporal channel variability occurs when the characteristics of the transmission medium change over time or when there is a relative motion between the receiver and transmitter, as in communication systems such as LTE and WiMAX. The frequency selectivity appears when multiple copies of the transmitted signal arrive at the receiver due to physical mechanisms such as multipath propagation.

Moreover, knowing the behavior or performance of a mobile communication system under real conditions (in situ test) can be very expensive, owing to the transfer of the communications system and test equipment to the place under study, among other issues. Additionally, the system behavior can not be tested under the same propagation conditions due to the nature of the communication channel. Faced with this problem, an economical alternative is to use mathematical models, which represent the radio channels under consideration. In this sense, we can define a channel simulator as a software tool that permits reproduction of the behavior or the propagation conditions of a mobile communications channel under controlled or laboratory conditions.

On the other hand, GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a CPU in order to accelerate scientific, engineering, and business applications [3]. Recently, several works related to the wireless communication area, which uses GPU devices, have been published [4–7]. Those works follow an implementation strategy in order to handle the channel complexity using multiple cores. For example, in [4] a wireless channel simulator is implemented. In that work, the potential of GPU-based processing is studied in order to improve the runtime performance of computationally intensive accurate wireless network simulation. In [5], the use of general purpose GPUs is investigated in order to provide the computational capabilities required for performing the radio frequency path loss computation. A discussion of the acceleration of wireless channel simulation using GPUs is provided in [6]. In addition, in [7], an implementation of parallel lattice reduction-aided 2×2 MIMO detector using GPUs for the WiMAX standard is presented.

Although several works related to the use of GPUs in communication systems exist, there are currently no works that describe in detail the implementation of a fading channel simulator based on GPUs. In this paper, the methodology for implementing a fading channel simulator (time and frequency selective) via GPU computing is presented.

The proposed methodology considers the use of common GPU software libraries that permit nonspecialized users in GPU programming to easily implement the proposed simulator. On the other hand, the generation of the Rayleigh fading variates is achieved using the filtering method [8–10]. In this case, the filtering method is carried out in time domain by using a finite impulse response (FIR) filter for coloring Gaussian noise samples. Furthermore, it is well known that if the filter order is increased, then the accuracy of the channel statistics can be improved, though at the cost of increasing the computational complexity. Therefore, in this work, we take advantage of GPUs for handling such computational complexity (multiplication and addition operations) in order to implement an accurate communication channel for SISO systems. Moreover, this methodology paves the way for implementing MIMO channel simulators in the future.

The rest of this paper is organized as follows: In the second section, the background of the wireless communication system is stated, specifically as regards the channel communication model. In Section 3, how to simulate the communication channel is explained. Next, in Section 4, the GPU implementation of the fading channel simulator is detailed. Section 5 is devoted to presenting the implementation results when a WiMAX scenario is considered. Finally, the conclusions are presented in Section 6.

2. Communication System

Consider a single-input and single-output (SISO) communication system where the transmission of in-phase $x_i(t)$ and quadrature $x_q(t)$ signals modulated by orthogonal carriers $\phi_i(t)$ and $\phi_q(t)$, respectively, are assumed, which are mixed for obtaining $x(t)$. This signal $x(t)$ is propagated through the communication channel $H(t,\tau)$, which is considered to be a causal time-varying linear system. The signal filtered by the channel reaches the receiver where a noisy version $y(t)$ is detected. It can be expressed mathematically as follows:

$$y(t) = \int_{-\infty}^{\infty} x(t-\tau) H(t,\tau)\, d\tau + u(t), \tag{1}$$

where $x(t) = x_i(t)\phi_i(t) + x_q(t)\phi_q(t)$, and t is a time variable. The impulse response $H(t,\tau)$ states the response of the channel in the instant t when a stimulus is applied in $t-\tau$, which reflects the time variability of the channel impulse response. Likewise, $u(t)$ is the aggregated stochastic noise. This received signal $y(t)$ is demodulated in order to obtain the in-phase and quadrature signals $y_i(t)$ and $y_q(t)$.

For sake of simplicity, if $\phi_i(t) = \cos(2\pi f_c t + \theta)$ and $\phi_q(t) = \sin(2\pi f_c t + \theta)$, where f_c is any carrier frequency and θ is any phase, the system becomes the well known single carrier communication system. It is important to emphasize that an OFDM system implemented with IFFT/FFT produces a base-band signal that is modulated as in a single carrier system.

If we consider that both signals $x_i(t)$ and $x_q(t)$ are band limited to a maximum frequency of $f_{\max}$ and $f_c \gg f_{\max}$ (this condition is always accomplished in real communication systems) it is easy to demonstrate [11, 12] with the aid of the Hilbert transform the existence of base-band equivalent signals $\tilde{y}(t)$, $\tilde{x}(t)$, $\tilde{u}(t)$, and $\widetilde{H}(t,\tau)$ for $y(t)$, $x(t)$, $n(t)$, and $H(t,\tau)$, respectively. In general, these equivalent base-band signals are complex, where the real part corresponds to the in-phase component and the imaginary to the quadrature component; thus, $\tilde{x}(t) = x_i(t) + jx_q(t)$ and $\tilde{y}(t) = y_i(t) + jy_q(t)$ for $j = \sqrt{-1}$. The relations between the original pass-band signals and their baseband equivalents are as follows [12]:

$$\begin{aligned} y(t) &= \operatorname{Re}\left(\tilde{y}(t)\, e^{j2\pi f_c t}\right), \\ x(t) &= \operatorname{Re}\left(\tilde{x}(t)\, e^{j2\pi f_c t}\right), \\ u(t) &= \operatorname{Re}\left(\tilde{u}(t)\, e^{j2\pi f_c t}\right), \\ H(t,\tau) &= \operatorname{Re}\left(\frac{1}{2}\widetilde{H}(t,\tau)\, e^{j2\pi f_c t}\right), \end{aligned} \tag{2}$$

where $\operatorname{Re}(\cdot)$ is the real part of the complex number in parentheses. Considering (2), the base-band equivalent of (1) is

$$\tilde{y}(t) = \int_{-\infty}^{\infty} \tilde{x}(t-\tau)\, \widetilde{H}(t,\tau)\, d\tau + \tilde{u}(t), \tag{3}$$

which can be interpreted as a collection of multiple paths (scatters), where the transmitted signal $\tilde{x}(t)$ is propagated.

The fact that these paths have different lengths and pass through different conditions of propagation causes the received signal from a specific path to be a delayed, attenuated, and phase-shifted version of the $\tilde{x}(t)$. In this sense, for a specific time t_1 and a specific delay τ_1, the channel coefficient $\widetilde{H}(t_1, \tau_1)$ will be a complex variable, where the magnitude represents the attenuation factor and the phase shift factor. On the other hand, due to the constant changes in the environment and the possible relative movement between transmitter and receptor, these factors are time dependent. According to [12], $\widetilde{H}(t, \tau)$ can be modeled as a complex stochastic process composed of the sum of a deterministic part (the ensemble average of $\widetilde{H}(t, \tau)$) and a random part (zero mean random process). From this point, we will only consider the random part (an assumption generally accepted when a channel simulator is developed). The autocorrelation function of this random process is equal to

$$R_{\widetilde{H}}(t_1, t_2; \tau_1, \tau_2) = E\left(\widetilde{H}(t_1, \tau_1)\, \widetilde{H}^*(t_2, \tau_2)\right), \quad (4)$$

where $E(\cdot)$ is the expectation operator and $(\cdot)^*$ represents the complex conjugate. This channel model is difficult to implement; nevertheless, some assumptions can be asserted which simplify the model. The first is the absence of correlation between the different scatters, and the second is that each scatter is a wide-sense stationary process, which together comprise the well known wide-sense stationary uncorrelated scattering (WSSUS) model. Therefore, (4) transforms into

$$\begin{aligned} R_{\widetilde{H}}(t_1, t_2; \tau_1, \tau_2) &= R_{\widetilde{H}}(t_1, t_1 - \Delta t; \tau_1, \tau_2)\, \delta(\tau_1 - \tau_2) \\ &= P_{\widetilde{H}}(\Delta t; \xi), \end{aligned} \quad (5)$$

where $\xi = \tau_1 = \tau_2$, $\Delta t = t_1 - t_2$, and $P_{\widetilde{H}}(\Delta t, \xi)$ is the autocorrelation function with respect to the time difference variable Δt for the scatter located in the delay variable ξ. From (5), it is possible to calculate the scattering function, which is defined as the Fourier transform of the correlation function with respect to the time difference variable Δt, as follows:

$$S(f; \xi) = \mathscr{F}\{P_{\widetilde{H}}(\Delta t; \xi)\}, \quad (6)$$

where $\mathscr{F}\{\cdot\}$ is the Fourier transform operator. This scattering function $S(f; \xi)$ indicates how the Doppler spectrum is for a given delay value in the variable ξ.

In many communication standards, a discrete number of scatters are considered instead of a continuous number, as suggested in previous equations. If this assumption is considered, then

$$\widetilde{H}(t, \tau) = \sum_{k=0}^{K-1} A_k(t)\, \delta(\tau - \tau_k), \quad (7)$$

where k is an index variable that enumerates the $K-1$ discrete scatters and $A_k(t)$ is a complex variable that encloses the gain and phase shift factor of such scatter. If a WSSUS channel is considered, the correlation function of (7) is

$$P_{\widetilde{H}}(\Delta t; k) = E\left(A_k(t)\, A_k^*(t - \Delta t)\right) \quad (8)$$

with scattering function

$$S(f; k) = \mathscr{F}\{P_{\widetilde{H}}(\Delta t; k)\}. \quad (9)$$

3. Channel Simulation

In order to perform a computational simulation of the communication channel, it is necessary to deal with the discrete version of the baseband equivalent channel presented in (7). This discrete channel results in band-limiting and sampling (7) in time and time-delay domains at a rate of $f_s \geq 2 f_{\max}$. Thus, it is defined as

$$h[n, m] = h(t, \tau)\big|_{t=nT_s, \tau=mT_s}, \quad (10)$$

where $T_s = 1/f_s$, $h(t, \tau) = \widetilde{H}(t, \tau) \otimes B(\tau)$, the symbol $\otimes$ represents the convolution operator, and $B(\tau)$ is a function for band-limiting the channel to $f_{\max}$, which, for practical purposes, could be a time windowed cardinal sine function. Substituting (7) into (10) results in

$$\begin{aligned} h[n, m] &= \sum_{k=0}^{K-1} A_k(nT_s)\, B(mT_s - \tau_k) \\ &= \sum_{k=0}^{K-1} A_k[n]\, B(mT_s - \tau_k), \end{aligned} \quad (11)$$

where $h[n, m]$ corresponds to the coefficients of the FIR filter for simulating the communication channel, n enumerates the samples in the time domain, and $0 \leq m \leq M - 1$ enumerates the taps of the filter. Likewise, M can be calculated as $\lceil(\tau_{\max} + t_B)/T_s\rceil$, where $\tau_{\max}$ is the maximum delay of the paths in the channel $\widetilde{H}(t, \tau)$, and $2t_B$ is the length of the filter $B(\tau)$. This filter could be anticausal; nevertheless, it is possible to introduce a delay in order to convert this filter into a causal filter and therefore physically feasible.

In order to implement (11), it is necessary to generate K uncorrelated discrete Gaussian stochastic complex processes at rate f_s. In the state of the art many algorithms for obtaining these stochastic processes are stated, as mentioned in [13–16] and references therein. Such processes must be filtered (colored) in order to accomplish the desired scattering function. It is important to note that these filters only affect the frequency components below a maximum Doppler frequency $f_{\max_D}$; therefore, it is possible to generate the samples at a rate of at least $f_{ls} \geq 2 f_{\max_D}$, where typically $f_{ls} \ll f_s$, and then to use any upsampling technique for accomplishing the f_s rate.

The impulse response of the filter for coloring the kth process is the discrete version (at rate f_{ls}) of the following expression:

$$G_k(t) = \sqrt{\mathscr{F}^{-1}\{S(f; k)\}}. \quad (12)$$

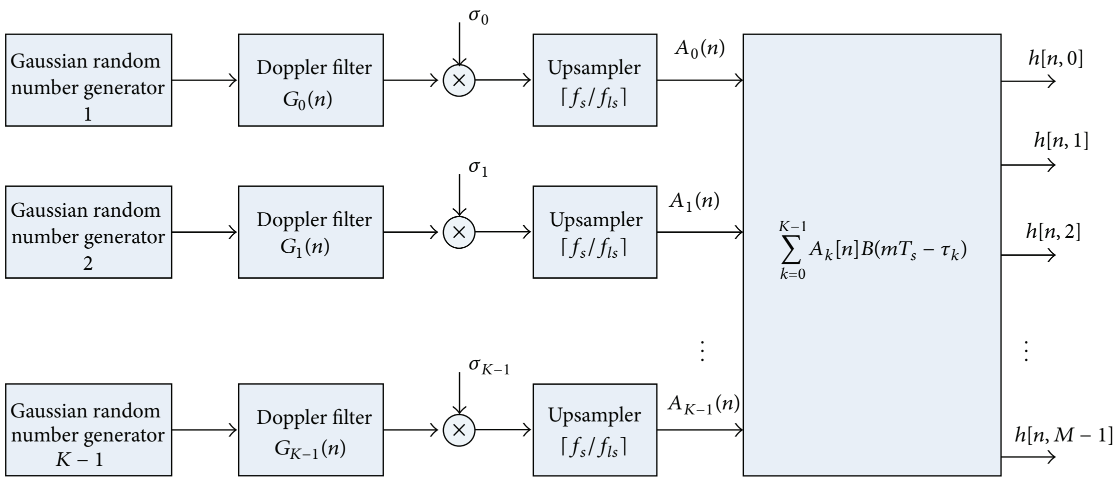
Finally, an interpolation technique such as splines, polynomial, or basis expansion is used for obtaining the samples at f_s rate. The entire process is presented in Figure 1 and summarized in Algorithm 1.

4. GPU Implementation

The emergence of GPUs has allowed complex algorithms to be executed almost in real time. GPU is conceptualized as a set of streaming multiproccesors (SM), where each SM is

```
Require: Scattering function
Require: Define the gain $\sigma_k^2$ that correspond to the variance of the process $A_k[n]$ for all the $K$ paths
  (1) for all $k$ such that $0 \le k \le K-1$ do
  (2)     Generate the zero mean unitary variance complex Gaussian stochastic process at rate $f_{ls}$ samples per second
  (3)     Multiply the stochastic process by $\sqrt{P_d(k)}$ for ensuring the gains of the paths
  (4)     Filter the process with discrete $G_k(t)$
  (5)     Interpolate the process for obtaining samples at rate $f_s$
  (6) end for
  (7) for all $n$ do
  (8)     Obtain $M$ filter's coefficients
          $h[n,m] = \sum_{k=0}^{K-1} A_k(nT_s)B(mT_s - \tau_k)$
  (9) end for
```

Algorithm 1: Channel generation procedure.



Gaussian random number generator 1
Doppler filter $G_0(n)$
σ_0
Upsampler $\lceil f_s/f_{ls} \rceil$
$A_0(n)$
Gaussian random number generator 2
Doppler filter $G_1(n)$
σ_1
Upsampler $\lceil f_s/f_{ls} \rceil$
$A_1(n)$
Gaussian random number generator $K-1$
Doppler filter $G_{K-1}(n)$
σ_{K-1}
Upsampler $\lceil f_s/f_{ls} \rceil$
$A_{K-1}(n)$
$\sum_{k=0}^{K-1} A_k[n]B(mT_s - \tau_k)$
$h[n,0]$
$h[n,1]$
$h[n,2]$
$h[n,M-1]$

Figure 1: Structure of the fading channel simulator.

characterized by a single instruction multiple data (SIMD) architecture. Therefore, in each clock cycle, each processor of the multiprocessor executes the same instruction, operating on multiple data streams; that is, each of these processors has the possibility of accessing a shared memory (common to all processors belonging to the same SM) and a local cache memory. In addition, all the processors have access to the global GPU (device) memory. Figure 2 illustrates the GPU hardware architecture.

Our strategy for implementing the fading channel simulator is aimed at improving the overall performance by chaining software functions (called kernels) representing each communication step. In order to implement the parallel fading simulator as illustrated in Figure 3, we distinguish five stages in the GPU design methodology as follows.

4.1. Gaussian Random Number Generator. In this stage, the CUDA Random Number Generation (`cuRand`) library [17] is employed in order to obtain Gaussian random numbers (GRN) by means of efficient generation of high-quality pseudorandom numbers. Particularly, `curand_init` function is launched for creating a random number generator in a massively parallel scheme. There are seven types of random number generators in `cuRand`; in this study, we have selected the XORWOW algorithm, which is a member of the Xor_shift family of pseudorandom number generators, with customized parameters for operating on GPUs.

The `curand_normal2` function generates two normally distributed pseudorandom numbers in each call. Because the underlying algorithm is based on the Box-Muller transform, it is suitable for generating random complex numbers; that is, each call generates real and imaginary parts at the same time.

There is a CUDA kernel for computing a set of K independent GRN vectors. Each vector corresponds to a path, which is computed in chunks by the GPU multiprocessors and then stored on device global memory. The implementation of the GNR generator is presented in the Algorithm 2, where the function `setup_kernel` initializes the threads of the same block with a different sequence number but the same seed and offset (zero offset). Furthermore, `generate_normal_kernel` computes several pseudorandom values with Gaussian distribution through the calling of `curand_normal2`.

4.2. Parallel Doppler FIR-Filter. The Doppler filter uses the resulting coefficients obtained by sampling (12) and

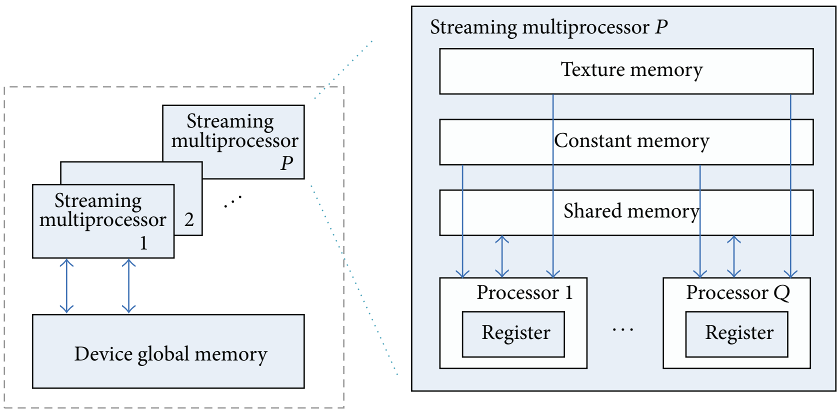


FIGURE 2: GPU data distribution for P multiprocessors with Q processors each.

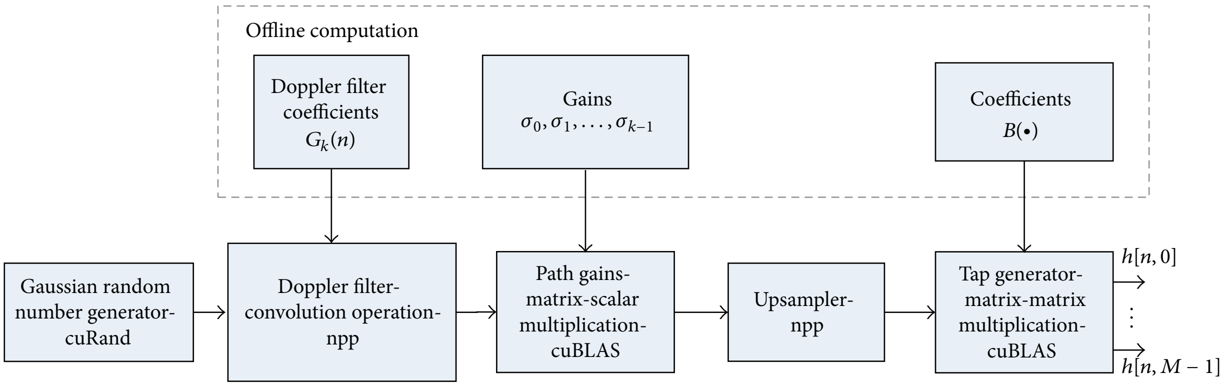

Offline computation
Doppler filter coefficients $G_k(n)$
Gains $\sigma_0, \sigma_1, \dots, \sigma_{k-1}$
Coefficients $B(\bullet)$
Gaussian random number generator-cuRand
Doppler filter-convolution operation-npp
Path gains-matrix-scalar multiplication-cuBLAS
Upsampler-npp
Tap generator-matrix-matrix multiplication-cuBLAS
$h[n, 0]$
$h[n, M-1]$

FIGURE 3: Proposed GPU design flow.

```
_global_ void setup_kernel(curandState *state)
{  int id = threadIdx.x + blockIdx.x *6;
   curand_init(1234*blockIdx.x, id, 0, &state[id]);
}
_global_ void generate_normal_kernel(curandState *state, int n, float *result)
{  int id = threadIdx.x + blockIdx.x *6;
   float2 x;
   curandState localState = state[id];
   for(int i=0; i<n; i++)
   {  /* Generate pseudorandom normals */
      x=curand_normal2(&localState);
      result[id]= x.x;
   }
    /* Copy state back to global memory */
   state[id] = localState;
}
```

ALGORITHM 2: Pseudorandom noise generation code.

the random numbers generated in the previous subsection. Since the filter coefficients are fixed for all channel realizations and paths, they are stored in the constant memory of GPU. This memory is devoted to storing and broadcasting read-only data to all threads on the GPU. In addition, the results of GRN are stored in shared memory, since many threads must access them simultaneously. The filtering is conceptualized as a convolution, so a kernel that performs the convolution in parallel is used.

There is a set of K independent 1D signal convolutions to be computed, one for each path. However, the filtering is performed using the NVIDIA Performance Primitives library

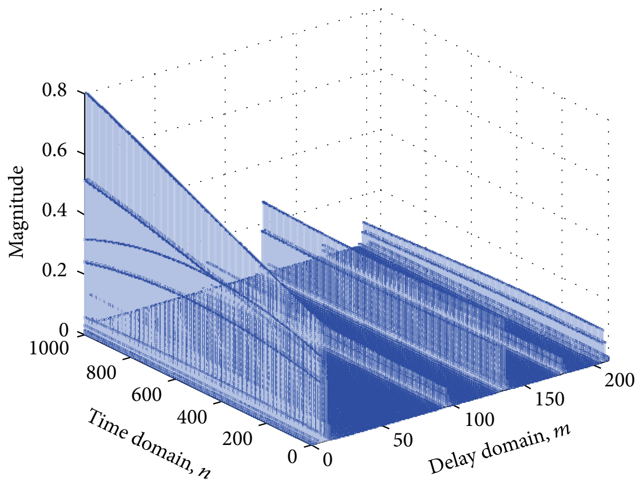


FIGURE 4: Impulse response realization of the fading channel simulator considering the vehicular class B ITU multipath channel model ($f_{\max_D} = 2000$ Hz, $f_s = 10$ Msps).

(`npp`) [18]; specifically, one of the `nppiFilterRow` functions is used, which performs a 1D filtering on 2D data, each row being a channel path.

4.3. Path Gain Implementation. The path gain is implemented with a multiplication function. The resulting colored noise from the previous stage is multiplied by a scalar. This could be carried out with a specific kernel or by using a standard library, such as CUDA Basic Linear Algebra Subroutines (`cuBLAS`) [19] or `npp`. The proposed implementation uses the `nppiMulC` function of the `npp` library.

4.4. Upsampler. The upsampler stage is responsible for generating noise samples at the rate f_s, implemented as an interpolation. The usual interpolation available for GPUs is the linear interpolation offered by texture memory; `npp` offers other methods for more accurate results. In this case, the `nppiResize` function with a cubic interpolation is used. It returns the interpolated value for a given coordinate within two known noise values.

4.5. Tap Generator. Multiple paths have been treated separately. In this stage, they are correlated using predefined (computed offline) coefficients according to (11). This correlation operation can be seen as the multiplication of N upsampled scaled colored noise (path) by the coefficient matrix $\mathbf{B}[m,k] = B(mT_s - \tau_k)$. This could be carried out with a programmer's own implementation or by using a standard library, such as `cuBLAS` as well. This proposal uses the `cublasSgemm` kernel that performs a matrix-matrix multiplication with optional scalar product.

5. Implementation Results

In order to corroborate the functionality of the proposed fading channel simulator in modern communication systems such as WiMAX, it was configured with the following parameters [20, page 404]: a maximum frequency Doppler $f_{\max_D} = 2000$ Hz and a sample rate $f_s = 10$ Msps, $f_{ls} = 10 f_{\max_D}$. In addition, the vehicular class B ITU multipath channel model was considered, which consists of six discrete paths with relative power $[-2.5, 0, -12.8, -10, -25.2, -16]$ dB at delay time $[0, 300, 8900, 12900, 17100, 20000]$ nsec, respectively. For implementing the filter $\mathbf{B}[m,k]$, a raised cosine function with a roll-off factor of 0.5 and a duration of $6T_s$ sec was considered. This delay results in the generation of $M = \lceil 20\,\mu\text{ sec} + 0.6\,\mu\text{ sec}\rceil / (0.1\,\mu\text{ sec}) = 206$ taps. In Figure 4, a resulting GPU-based realization of the fading channel according to the specified parameters for $N = 1024$ time samples is presented. It is important to note that the offline computed data (see Figure 3) are transferred to GPU simulator by text files.

TABLE 1: Channel emulator implementation comparison. Time consumption for 1 mega samples generation and 10 channel realizations (in milliseconds).

	Matlab[1]	CUDA Libs[2]
Min	1640.895	37.376
Max	1821.171	75.186
Mean	1760.821	46.935

[1]CPU: Intel Core i5 3.4 GHz 16 GB.
[2]GPU: GeForce GTX 780M 4 GB.

The simulation was carried out using an iMAC computer with the following specifications: OS 10.9.4 (Maverics), Intel Core processor i5 (3.4 GHz), 16 GB of RAM, graphic card GeForce GTX 780 M with 4 GB of RAM, and 1536 CUDA cores.

For evaluating the time performance, the parameters used in the previous test have been maintained; however, the parameter N was fixed to $N = 1 \times 10^6$ samples. In this sense, Table 1 presents the average, maximum, and minimum time consumption for a CPU-based implementation (Matlab) versus the proposed GPU-based methodology (CUDA). It is clear that the GPU methodology has gains of 30-fold (mean

Table 2: Time consumption by module computing 10 channel realizations.

Time (%)	Module
78.18%	Matrix-matrix multiplication
13.62%	Initializing random number generator[1]
3.83%	FIR filter
2.37%	Upsampling
1.62%	Gaussian number generation
0.01%	Path gain

[1]The seed initialization is carried out only once before the first channel realization.

Table 3: Time consumption comparative (in milliseconds): CPU-based implementation (Matlab) versus GPU-based implementation (CUDA).

N (samples)	Matlab[1]	CUDA[2] Libs	x-fold (gain)
5,120	31.5466	0.614496	51
10,240	38.5282	1.350240	28
20,480	54.7829	2.331968	23
81,920	179.8391	5.785952	31
327,680	633.4515	17.09622	37
655,360	1204.584	25.36316	47
1,000,000	1769.243	37.81030	47
1,310,720	3024.966	47.43065	64

[1]CPU: Intel Core i5 3.4 GHz 16 GB.
[2]GPU: GeForce GTX 780M 4 GB.

value) when compared with CPU-based implementations, which is attractive if parallel versions of the channel simulator are required, as could be the case in MIMO applications.

Table 2 reports the time percentage for accomplishing each task of the channel simulator in the GPU. It should be noted that in this table the reading and device memory allocation—the most time-consuming tasks—are not considered. These tasks are performed only once at the initialization stage of the simulation.

On the other hand, Table 3 and Figure 5 present the overall time consumption in milliseconds for CPU- and GPU-based implementations when the number of samples is fixed to $N = 5120$, 10240, 20480, 81920, 327680, 655360, 1000000, and 1310720 samples. This shows that while the time consumption in the CPU-based implementation increments exponentially, it remains almost linear in the GPU-based implementation.

Similarly, the good performance achieved with the GPU implementation with respect to the CPU implementation can be observed in the x-fold gain reported in Table 3. This gain is calculated as the time consumption quotient of both implementations. The behavior of this gain has been reported for each of N samples stated in the previous paragraph.

Finally, it is important to emphasize that the presented approach can deal with several path realizations. This suggests that the developed fading channel simulator can be considered for generating large MIMO channels, which represents a new simulation paradigm.

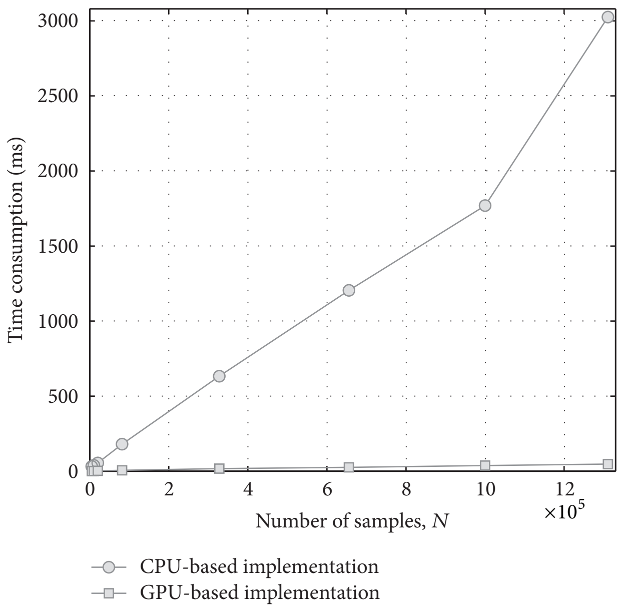


Figure 5: Time consumption comparative: CPU-based implementation versus GPU-based implementation.

6. Conclusions

The principal result of this study is the introduction of a methodology for designing fading channel simulators via GPU devices. Such a methodology permits nonspecialized users to easily implement channel simulators in parallel. As was shown, the use of GPUs in the development of fading channel simulators greatly saves simulation time when channel realizations are generated for testing communication systems. Moreover, a case of study for WiMAX systems demonstrated the functionality of the implemented channel simulator. We believe that the proposed parallel channel simulator can aid in testing mobile communication systems based on LTE and WiMAX. Additionally, the presented approach based on GPU will allow the design of more sophisticated simulators of complex channel models such as triply selective MIMO fading channels (i.e., time, frequency, and space selective).

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the Programa para el Desarrollo Profesional Docente (PRODEP) 2014 and CONACYT, Ciencia Básica, 2014 (CB2014-241272), Mexico.

References

[1] O. Longoria-Gandara and R. Parra-Michel, "Estimation of correlated MIMO channels using partial channel state information

and DPSS," *IEEE Transactions on Wireless Communications*, vol. 10, no. 11, pp. 3711–3719, 2011.

[2] M. Bazdresch, J. Cortez, O. Longoria-Gándara, and R. Parra-Michel, "A family of hybrid space-time codes for MIMO wireless communications," *Journal of Applied Research and Technology*, vol. 10, no. 2, pp. 122–142, 2012.

[3] NVIDIA, "High-performance computing," 2014, http://www.nvidia.com/object/what-is-gpu-computing.html.

[4] P. Andelfinger, J. Mittag, and H. Hartenstein, "GPU-based architectures and their benefit for accurate and efficient wireless network simulations," in *Proceedings of the 19th Annual IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '11)*, pp. 421–424, July 2011.

[5] B. J. Henz, D. Richie, E. Jean, S. J. Park, J. A. Ross, and D. R. Shires, "Real-time radio wave propagation for mobile ad-hoc network emulation using GPGPUs," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '13)*, 2013.

[6] S. Bai and D. M. Nicol, "Acceleration of wireless channel simulation using GPUs," in *Proceedings of the European Wireless Conference (EW '10)*, pp. 841–848, Lucca, Italy, April 2010.

[7] H. Yang, T. Kim, C. Ahn, J. Kim, S. Choi, and J. Glossner, "Implementation of parallel lattice reduction-aided MIMO detector using graphics processing unit," *Analog Integrated Circuits and Signal Processing*, vol. 73, no. 2, pp. 559–567, 2012.

[8] M. Pätzold, *Mobile Radio Channels*, John Wiley & Sons, 2nd edition, 2011.

[9] G. L. Stüber, *Principles of Mobile Communication*, Springer, 3rd edition, 2011.

[10] I. Cyril-Daniel, "A MATLAB-based object-oriented approach to multipath fading channel simulation," Tech. Rep., Hi-Tek Multisystems, Québec, Canada, 2008.

[11] M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems: Modeling, Methodology and Techniques*, Information Technology: Transmission, Processing and Storage, Springer, Berlin, Germany, 2nd edition, 2000.

[12] P. Bello, "Characterization of randomly time-variant linear channels," *IEEE Transactions on Communications*, vol. 11, no. 4, pp. 360–393, 1963.

[13] J. V. Castillo, A. C. Atoche, O. Longoria-Gandara, and R. Parra-Michel, "An efficient Gaussian random number architecture for MIMO channel emulators," in *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS '11)*, pp. 316–321, Beirut, Lebanon, October 2011.

[14] L. Vela-Garcia, J. V. Castillo, R. Parra-Michel, and M. Pätzold, "An accurate hardware sum-of-cisoids fading channel simulator for isotropic and non-isotropic mobile radio environments," *Modelling and Simulation in Engineering*, vol. 2012, Article ID 542198, 12 pages, 2012.

[15] J. Vázquez Castillo, L. Vela-Garcia, C. Gutiérrez, and R. Parra-Michel, "A reconfigurable hardware architecture for the simulation of Rayleigh fading channels under arbitrary scattering conditions," *AEU*, vol. 69, no. 1, pp. 1–13, 2015.

[16] V. Kontorovich, S. Primak, A. Alcocer-Ochoa, and R. Parra-Michel, "MIMO channel orthogonalisations applying universal eigenbasis," *IET Signal Processing*, vol. 2, no. 2, pp. 87–96, 2008.

[17] NVIDIA, CUDA random number generation library (cuRAND), 2014, https://developer.nvidia.com/curand.

[18] NVIDIA, "NVIDIA performance primitives NVIDIA developer zone," https://developer.nvidia.com/npp.

[19] NVIDIA, CUDA basic linear algebra subroutines (cuBLAS), 2014, https://developer.nvidia.com/cublas.

[20] J. G. Andrews, A. Ghosh, and R. Muhamed, *Fundamentals of WiMAX: Understanding Broadband Wireless Networking*, Prentice Hall, Upper Saddle River, NJ, USA, 2007.