Autómata Celular Estocástico paralelizado por GPU aplicado a la simulación de enfermedades infecciosas en grandes poblaciones

Stochastic Cellular Automata approach for infectious disease simulation on large populations parallelized by GPU

Héctor Cuesta-Arvizu*, Adrián Trueba-Espinosa*, José Ruiz-Castilla*

RESUMEN

Un gran número de áreas de la ciencia están siendo beneficiadas por la reducción de tiempo de cómputo gracias al uso de las Unidades Gráficas de Proceso (GPU). En el caso de la Epidemiología, tales unidades agilizan la simulación de escenarios con poblaciones grandes, escenarios en los que el tiempo de procesamiento es muy significativo. El presente artículo introduce la simulación de eventos epidemiológicos basado en un modelo de Autómatas Celulares Estocásticos (AC), el cual ofrece la implementación de las características principales de una enfermedad infecciosa a gran escala: contacto, vecindario, trayectorias y transmisibilidad. Un caso de estudio es simulado en una implementación del algoritmo AC para una enfermedad infecciosa de tipo SEIRS (Susceptible, Expuesto, Infectado, Recuperado y Susceptible). Una población de 1 000 000 de individuos es paralelizada a través de un algoritmo de balanceo de procesos implementado en el lenguaje de programación C-CUDA. El resultado dado por el software paralelizado por GPU es comparado contra un análisis hecho del modelo paralelizado por multi-hilos CPU. Los resultados demuestran que el tiempo de cómputo puede ser reducido significativamente gracias al uso de C-CUDA.

ABSTRACT

In science, a large number of areas are being benefited by the reduction of computational time with the use of Graphics Processing Units (GPU). In the case of Epidemiology, the benefit consists in the speedup of simulation of scenarios with bigger populations in which the processing time is large. This article introduces an epidemiological event simulation with a model based on Stochastic Cellular Automata (SCA). This model provides an implementation of the main features of a large-scale infectious disease: contact, neighborhood, trajectories and transmissibility. A case study is simulated on an implementation of the SCA algorithm for an infectious disease type SEIRS (Susceptible, Exposed, Infected, Recovered and Susceptible). A population with 1 000 000 of individuals is parallelized through a process balancing algorithm implemented in C-CUDA. The result given by the GPU parallelized software is compared against a parallelized model analysis made by multi-threaded CPU. The results show that the computation time can be significantly reduced through the use of C-CUDA.

Recibido: 21 de marzo de 2012
Aceptado: 3 de julio de 2012

Palabras clave:
Epidemiología; Autómata Celular; GPU; modelo estocástico; CUDA; simulación.

Keywords:
Epidemiology; Cellular Automata; GPU; stochastic model; CUDA; simulation.

INTRODUCCIÓN

La Epidemiología es definida como el estudio de la distribución y factores determinantes de la salud pública, así como su aplicación en la prevención y control de problemas de salud en una población [1]. Los eventos epidemiológicos provocados por un virus nuevo como el H1N1 [2] y las nuevas variedades de bioterrorismo pueden provocar grandes pérdidas humanas y económicas [3]. Es prioritario el estudio y la simulación de la dispersión de patógenos en una población.

*Centro Universitario UAEM. Universidad Autónoma del Estado de México (UAEM). Texcoco, Av. Jardín Zumpango s/n, fraccionamiento El tejocote, C. P. 56259, Estado de México. México. Correo electrónico: hmcuesta.data@gmail.com; web: http://cux.uaemex.mx/~investigacion/MCC.html

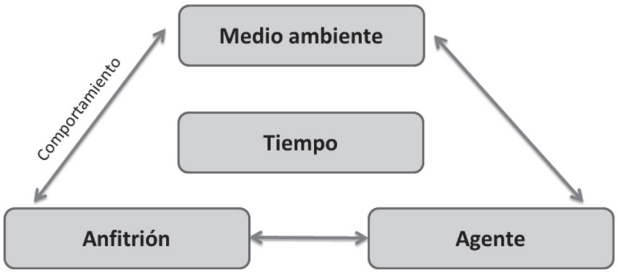

Figura 1. Triángulo epidemiológico de las enfermedades infecciosas.

En la figura 1 podemos apreciar los factores que propician la distribución de un agente patógeno en una población a través del tiempo y el medio ambiente. En ésta, la interacción del medio ambiente y el anfitrión (individuos de la población) son provocados por un comportamiento demográfico, geográfico o social.

La modelación de eventos epidemiológicos puede ser tratado desde el punto de vista matemático como variantes del modelo SIR (Susceptibles, Infectados y Recuperados) desarrollado en 1927 por W. O. Kermack y A. G. McKendrick [4]. En este modelo se categorizan los diferentes grupos en la población, de acuerdo a su estado durante un brote epidémico. Este tipo de modelos puede ser simulado en sistemas de ecuaciones diferenciales o por cadenas ocultas de Markov [5].

A continuación se describe el sistema de ecuaciones para el modelo SEIRS (Susceptible, Expuesto, Infectado, Recuperado y Susceptible) y se explica en que consiste:

(a) $\frac{dS}{dt} = -\beta * S * I + \rho * R,$

(b) $\frac{dE}{dt} = \beta * S * I - \sigma * E,$

(c) $\frac{dI}{dt} = \sigma * E - \gamma * I,$

(d) $\frac{dR}{dt} = \gamma * I - \rho * R.$

Dadas las cuatro fases de la enfermedad (a, b, c, d), se indican aquéllas por las cuales transitarán los individuos durante la enfermedad infecciosa [1].

Definición. β es la transmisibilidad de la enfermedad. S,E,I,R son los estados por los cuales pasan los individuos de la población. ρ, σ, γ representan los periodos que los individuos deben permanecer en el estado correspondiente.

Desde el punto de vista computacional, los Autómatas Celulares (AC) están siendo ampliamente utilizados [6] gracias a su facilidad para representar sistemas de reglas naturales en pasos discretos y a su capacidad para representar la interacción entre individuos a través de su vecindario con los individuos circundantes. Esto convierte a los AC en una opción para las simulaciones del método de Montecarlo, en la que el uso de un ambiente estocástico puede simular el comportamiento de la naturaleza y así experimentar con distintas estrategias de simulación.

Por otra parte, el desarrollo de dichas simulaciones representa un alto costo en tiempo de procesamiento, esto debido a que se tienen que generar cientos de miles de números aleatorios y a que se tienen que realizar millones de interacciones entre los individuos de la población estudiada. La aparición de equipo de cómputo que cuenta con Unidades de Procesamiento Gráfico (GPU) ha permitido a los investigadores hacer uso del poder de cómputo generado por ese gran número de núcleos de procesamiento (los cuales pueden trabajar en paralelo operaciones simples [7]).

MATERIALES Y MÉTODOS

Modelo epidemiológico SEIRS

El modelo epidemiológico SEIRS describe el curso de una enfermedad infecciosa (tal como se observa en la figura 2). Dicho modelo opera: comenzando con una población susceptible (S), la cual entra en contacto con una población infectada (I); tomando en cuenta un periodo de incubación o exposición al patógeno (E); una vez que el periodo de infección se cumple, considerando que el individuo entra en un estado de recuperado (R) y conserva dicho estado por un periodo de inmunidad al patógeno; una vez perdida dicha resistencia, el individuo regresa a un estado susceptible. Este tipo de enfermedades se denominan "endémicas", ya que el remanente de individuos que regresa a la población susceptible poco a poco estabiliza la curva de las diferentes poblaciones.

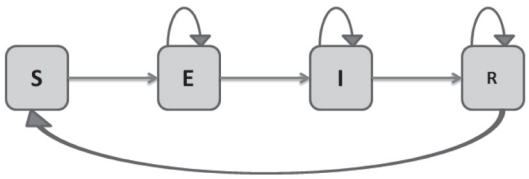

Figura 2. Ciclo de vida del modelo SEIRS.

Autómata Celular Estocástico

En la década de los 50, Von Neumann y Stanislaw Ulam conciben el concepto de AC como un modelo para representar eventos físicos a través de reglas discretas. Cada célula es fijada en un enrejado (*lattice*)

de *n* dimensiones finitas. Dicha célula modifica su estado en base al estado de las células circundantes (cuyo alcance puede variar según sea su vecindario -véase figura 3-). Cada vez que las reglas son aplicadas a todas las células se considera una generación (periodo de tiempo).

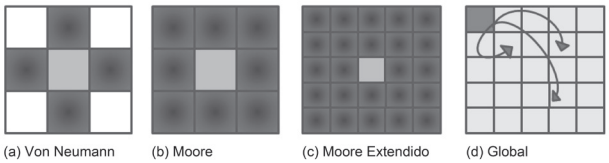

Figura 3. Estructura y vecindarios de la célula, local (a, b, c) y global (d).

a) Formalismo. Un Autómata Celular es una red de autómatas definidas sobre un grafo regular con una máquina de estados finita idéntica para cada unidad celular. Generalmente, los Autómatas Celulares están definidos sobre retículos euclídeos de una o dos dimensiones que forman un grafo homogéneo, y sobre cuyos puntos enteros se colocan las unidades celulares [6].

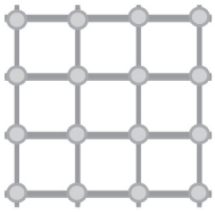

Figura 4. Autómata Celular representado en un grafo de Cayley.

Formalmente, un Autómata Celular se define como $M = \{C,Mq\}$ donde $C = (V,Q)$ es un grafo de Cayley (figura 4) [8], y donde un conjunto finito $V = |N|$ son los vértices y Q los arcos [3]. Mq es una copia de una máquina de estados finitos para cada vértice, con un alfabeto de entrada $Q^d = Qx...xQ$ (d = veces) y una función de transición local definida como:

$$\delta : QxQ^d \rightarrow Q$$

$$(X_i, X_{i_2},..., X_{i_d}) \rightarrow \delta(X_i, X_{i_2},..., X_{i_d}) \in Q.$$

b) Modelo de contacto. La forma en la que una célula tiene interacción con otras está condicionada al modelo de contacto con el que cuente el AC. Para el caso del modelo SEIRS, tenemos las siguientes distribuciones de contacto:

i. el número de contactos a través de la distribución de toda la población, determinado por $C_{tot} = (N_{tot} \cdot CR)/2$ (donde N_{tot} es el tamaño de la población total del AC y CR es el número de contactos promedio por individuo);

ii. el número de contactos a través de una distribución basada en la población infectada está determinada por $C_{\inf} = (N_{\inf} \cdot CR)/2$ (donde $N_{\inf}$ es la población infectada del AC y CR es el número de contactos promedio por individuo).

En ambas distribuciones, el número de contactos es dividido entre 2 ya que toda interacción consta de 2 contactos.

Conceptos básicos de GPU y C-CUDA

Las GPUs son procesadores de varios núcleos que ofrecen alto rendimiento y que originalmente fueron utilizados para el procesamiento gráfico o el entretenimiento (diseño CAM-CAD o videojuegos).

La *Compute Unified Device Architecture* (CUDA) es un modelo de programación de propósito general donde el usuario inicializa conjuntos de procesos (*threads*) en la GPU. Cuenta con un manejo explicito de la memoria de ésta, obteniendo una aceleración garantizada en los accesos a memoria.

La capacidad de realizar simultáneamente múltiples operaciones provee un paralelismo con muchas posibilidades para el cómputo en general y para muchas de las áreas donde hay problemas (análisis de conjuntos de datos muy grandes, procesamiento de textos, tratamiento de datos multimedia y, en general, cualquiera de los llamados "problemas de datos a gran escala" [9]). Actualmente se puede hacer uso de diferentes medios para ello, a saber, OpenCL, CUDA, etcétera. Este artículo se enfoca en el procesamiento por GPU con CUDA, para lo cual se usó una tarjeta NVIDIA Geforce GTX260 con 30 MP (Multi-Procesadores), 8 SP (Procesadores Escalares) por cada procesador (MP), 16 384 registros por MP y 1 GB de memoria global.

CUDA es una Interfaz de Programación de Aplicaciones (API, *Application Programming Interface*), provista por las tarjetas gráficas NVIDIA, que nativamente está implementada en lenguaje C, aunque también existen implementaciones del API para otros lenguajes (ver figura 5), o bien otras implementaciones específicas (por ejemplo, Java -a través de JCuda- permite obtener un gran rendimiento para problemas con un alto paralelismo, ofreciendo la posibilidad de agilizar el software). No obstante, aun con estas ventajas tecnológicas, no todo es paralelizable y es necesario identificar qué partes del algoritmo optimizar, así como organizar la memoria de tal forma que los accesos paralelos sean óptimos y con esto evitar cuellos de botella.

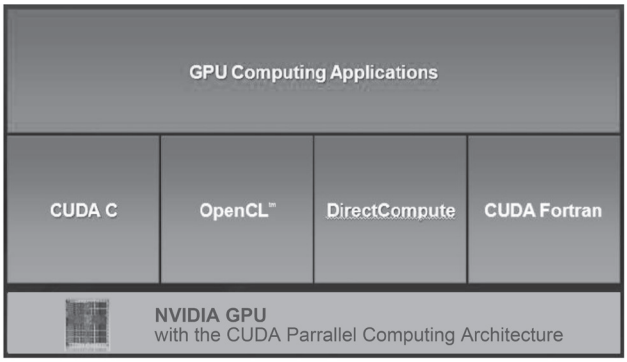

Figura 5. Ambiente de desarrollo para la arquitectura CUDA [10].

Paralelizar los programas sirve básicamente para resolver problemas en menos tiempo y, al ejecutarse el programa tiempo de ese modo, se facilita trabajar con problemas que requieren mucho poder de cómputo en un tiempo de ejecución razonable.

Funcionamiento de CUDA

En la figura 6 se observa cómo esta organizada internamente la GPU; en ésta podemos observar conceptos como *thread* (hilo), *block* (bloque) y *grid* (cuadrícula). La cuadrícula tiene dos dimensiones de bloques y, a su vez, cada bloque tiene tres dimensiones de hilos (en el que cada hilo tiene una dirección).

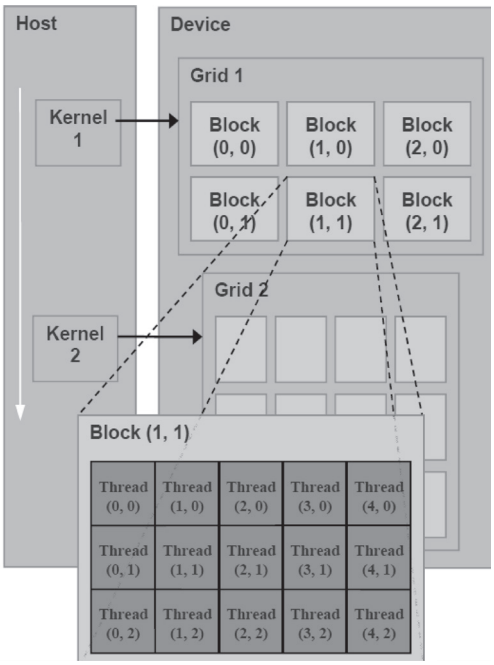

Figura 6. Organización de hilos, bloques y cuadrículas [10].

Un kernel en CUDA es una función que, al ejecutarse, lo hará en *n* distintos hilos en lugar de forma secuencial. Se ejecuta el kernel con 5 cuadrículas × 2 bloques, lo que dará 10 hilos del kernel. Un bloque es un grupo de hilos que se pueden sincronizar y comunicar entre ellos. Una cuadrícula es una colección de bloques que permiten ejecutar múltiples bloques en un kernel [11].

Implementación

El presente articulo muestra un análisis comparativo del rendimiento provisto por el paralelismo por CPU a través de un *ThreadPool* (un contenedor dentro del cual se crean y ejecutan hilos) programado en Lenguaje C#, contrastado con un paralelismo con GPU programado en C-CUDA.

Para ambas implementaciones se consideran condiciones de frontera periódica del AC que permiten conservar un vecindario uniforme en cualquier región de la matriz en un hiperespacio toroide.

Tabla 1.
Características del hardware.

Parámetro	Valor
Procesador CPU	Intel Core i5 2,30 GHz 4 CPUs
Memoria RAM	4 GB DDR3
Disco duro	500 GB 7 200r
Tarjeta de video	NVIDIA Geforce GTX260, 1 GB memoria

a) Implementación usando multi-hilos. La implementación por CPU está desarrollada en lenguaje C# a través de un *ThreadPool* que maneja una estructura de 4 hilos (uno por núcleo del procesador Intel Core i5), así como de un balanceador de procesos asíncrono que resuelve la cola de mensajes entre procesos.

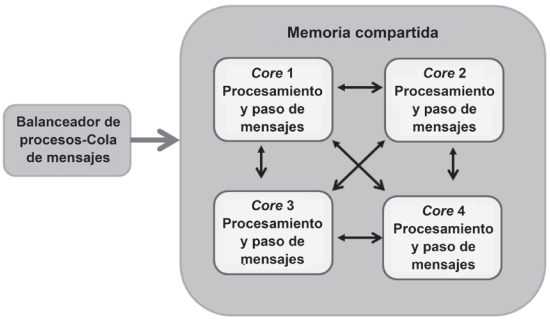

Figura 7. Diagrama de bloques del CPU.

Para este caso, se considera que con el lenguaje de alto nivel no se cuenta con el control de detalles de bajo nivel -como el paso de mensajes- y solo se paralelizan partes del código que consideran un origen de datos en memoria compartida (ver figura 7). El objeto es tratar de aprovechar la topología del procesador subyacente.

b) Implementación usando C-CUDA. La implementación por GPU está dada por una descomposición de la matriz de proceso del AC usando un modelo de memoria compartida en un hiperespacio toroide, donde las condiciones de frontera periódica del AC permiten conservar un vecindario uniforme en cualquier región de la matriz.

La distribución de los procesos es obtenida a través de la descomposición de matriz dada:

$$A = \begin{pmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & \ddots & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{pmatrix}$$

Se tiene pues que cada sub-bloque del AC es igual a $A_{i,j}$, que a su vez es una matriz de $nb \cdot nb$. Dichos bloques son mapeados a procesos asignando $A_{i,j}$ al balanceador de procesos.

Cada una de las matrices generadas cuenta con la misma estructura que la original pero de dimensiones reducidas, de tal forma que se puede distribuir en los núcleos de la GPU y tener un arranque multi-punto. Finalmente, se puede notar que la paralelización por descomposición presentada aquí corresponde a la descomposición de Cholesky [10], la cual es un ejemplo del método "divide y vencerás".

RESULTADOS Y DISCUSIÓN

a. Caso de estudio

Para este caso se utilizaron datos del virus de la influenza, que es un virus de tipo ARN de la familia de los *Orthomyxoviridae*. Se tomó una población cerrada en un ambiente idealizado, donde las variables demográficas o geográficas no son consideradas (ver tabla 2).

Tabla 2.
Parámetros para el caso de estudio.

Descripción	Valor
Numero de individuos en la población	1 000 000
Periodo a simular	365 días
Transmisibilidad	0,3
Periodo de incubación	5 días
Periodo infeccioso	6 días
Periodo de resistencia a la enfermedad	15 días
Nacimientos o decesos	0
Número de casos índice	10

Reglas para la dispersión de la enfermedad (véase figura 2):

- Un individuo (S) cambia su estado a (E) después de su interacción con un (I) y evoluciona según sus periodos de incubación e infeccioso.
- Luego de permanecer en (R) por su periodo de resistencia, el individuo regresa a un estado (S).
- Se considera un vecindario de tipo Moore extendido a 4 niveles (ver figura 3-c).
- Es considerado un inicio del evento multi-punto por la distribución aleatoria de los casos índice (individuos infectados).
- Se considera una condición de frontera periódica, donde todos los individuos cuentan con las mismas reglas de vecindad.

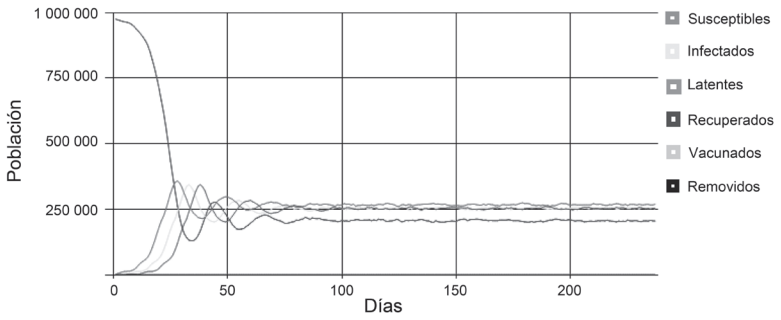

Figura 8. Curva de la enfermedad infecciosa tipo SEIRS.

En la figura 8 se observa la curva de evolución de la enfermedad, resultado de la simulación, la cual es congruente con la curva de una epidemia endémica remergente por la naturaleza el modelo SEIR [1].

b. Evaluación del desempeño

Debido a la naturaleza de los AC, la generación de números aleatorios es fundamental en el rendimiento de la aplicación. Como se observa en la figura 9, la velocidad con la que se generan 100 000 números aleatorios en CPU es de 18 s (0,3 min), en contraste con la versión por GPU que solo toma 3 s (0,05 min), lo que da un factor casi 6 veces más rápido.

Sabiendo el hecho de que la distribución es uniforme y usando la misma base (*seed*) para la generación de los números aleatorios, se determina una reducción considerable en tiempo de cómputo para el caso del procesamiento por GPU.

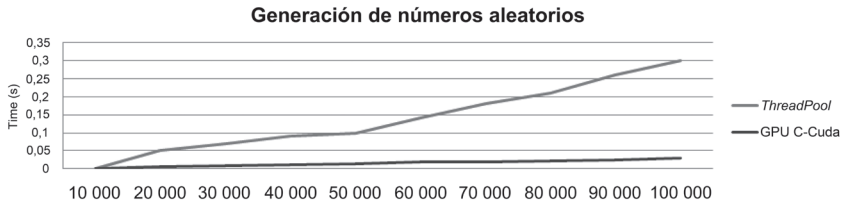

Figura 9. Gráfica de contraste CPU vs. GPU para la generación de números aleatorios.

Por el lado de los contactos entre individuos del AC, con una población de 1 000 000 de células en una matriz de A 1 000 × 1 000, por iteración se observa que el CPU tarda 260 s; por el lado de la GPU, la resolución se da en 35 s, lo que habla de un factor de 7 veces más rápido.

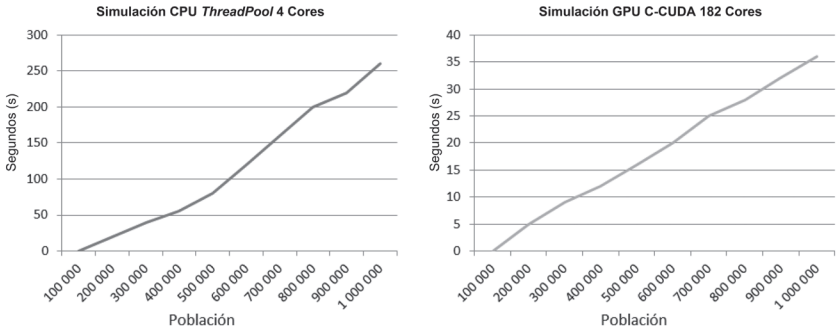

Figura 10. Gráfica de contraste CPU vs. GPU para los contactos entre individuos en una sola generación.

En la tabla 3 se observa el rendimiento en términos de segundos de la simulación del AC (considerando a cada iteración del AC como un día), con lo cual es posible notar la coherencia con los valores ofrecidos en la figura 10. La simulación se ejecutó 10 veces ofreciendo resultados similares con una desviación de 0,03 s para la versión con GPU y 0,18 s para la versión con CPU. Los tiempos de respuesta fueron tomados mediante la escritura de un registro (log) de registro de inicio y fin en el código de la simulación.

Tabla 3.
Comparativa de rendimiento para la simulación AC del caso de estudio.

Caso de estudio	CPU	GPU	Factor de ganancia (CPU/GPU)
1 día	260 s	35 s	7,5
30 días	7 680 s	1 030 s	7,4
100 días	25 300 s	3 550 s	7,1
200 días	52 800 s	6 940 s	7,6
365 días	95 200 s	12 980 s	7,3

CONCLUSIONES

En este trabajo se presenta el análisis de rendimiento para una simulación de eventos epidemiológicos con CA en poblaciones grandes aplicando dos métodos de procesamiento paralelo.

Se observa una diferencia de 7 veces el rendimiento reportado por la implementación por GPU por sobre el tiempo reportado por la implementación en CPU. Dicho incremento en el ahorro de tiempo de cómputo posibilita la simulación de ciudades de alta población y facilita el análisis de eventos epidemiológicos, así como la toma de decisiones de los sistemas de salud pública (como por ejemplo en estrategias de vacunación, en cercos sanitarios o aislamientos, en suspensión de clases en escuelas de las zonas afectadas, entre otros).

Adicionalmente, el trabajo en progreso plantea hacer uso de distribuciones diferentes en la creación de números aleatorios y hacer contrastación de simulaciones en implementaciones diferentes (modelado matemático, inferencia estadística, problemas inversos o sistemas multi-agente) para corroborar los datos aquí mostrados.

REFERENCIAS

[1] Merrill, R. M. (2010). *Introduction to Epidemiology.* Fifth edition. Jones and Bartlett Publishers: pp. 185-207.

[2] Velasco-Hernandez, J. X. and Leite, M. C. (2011). A model for A(H1N1) epidemic in Mexico including social isolation. *Salud Publica de Mexico* 53(1): pp 40-47.

[3] Mikler, A. R., Bravo-Salgado, A. and Corley, C. D. (2009). Global Stochastic Contact Modeling of Infectious Diseases. *International Joint Conference on Bioinformatics, Systems Biology and Intelligent Computing 2009*: pp. 327-330.

[4] Kermack, W. O. and McKendrick, A. G. (1927). Contribution to the Mathematical Theory of Epidemics. *Proceedings of the Royal Society* 1154: pp. 700-721.

[5] Allen, L. J. S. (2005). *An Introduction to Stochastic Epidemic Models.* Department of Mathematics and Statistics. Texas Tech University: pp. 1-50.

[6] Cuesta-Arvizu, H., Bravo-Salgado, A., Mikler, A. R. y Trueba-Espinosa, A. (2011). Modelado para estudio de brotes epidémicos usando un Autómata Celular Estocástico Global. *IEEE ROC&C 2011*. CP-12.

[7] Xu, C., Kirk, S. R. and Jenkins, S. (2009). Tiling for Performance Tuning on Different Models of GPUs. *Second International Symposium on Information Science and Engineering* 2009: pp. 500-504.

[8] West, D. B. (1996). *Introduction to Graph Theory.* Prentice-Hall: pp. 98-132.

[9] Huang, O., Huang, Z., Werstein, P. and Purvis, M. (2008). GPU as a General Purpose Computing Resource. *Proceedings of the 2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*: pp. 151-158.

[10] Kirk, D. and Hwu, W. W. (2008). *Programming Massively Parallel Processors: the CUDA experience*. Presented at the Taiwan 2008 CUDA Course. June 30-July 2.

[11] NVIDIA. (2012). *NVIDIA Technical Brief: NVIDIA GeForce GTX 200 GPU Architectural Overview*. Recuperado de http://www.nvidia.com/docs/IO/55506/GeForce_GTX_200_GPU_Technical_Brief.pdf