Simulación de un Modelo de Percolación basado en FPGA

CONFERENCE PAPER · SEPTEMBER 2013

DOI: 10.13140/2.1.2166.2406

1 AUTHOR:

Mauricio Raúl Palavecino Nicotra
Universidad Nacional de San Luis
2 PUBLICATIONS **0** CITATIONS

SEE PROFILE

Simulación de un Modelo de Percolación basado en FPGA

Mauricio R. Palavecino, Paulo Centres, Carlos Sosa Páez
Facultad de Ciencias Físico, Matemáticas y Naturales
Universidad Nacional de San Luis
San Luis, Argentina
e-mail: mauriciopalavecino@yahoo.com.ar, {pcentres, sosapaez}@unsl.edu.ar

***Resumen*—En este trabajo se describe la implementación de un sistema diseñado para simular un modelo de Percolación utilizando el método de Monte Carlo, en una plataforma basada en FPGA de bajo costo. Se describe brevemente el fenómeno de percolación y se propone un diseño donde se aprovechan las características de las FPGA para realizar el algoritmo de manera eficiente. Se trabaja sobre redes cuadradas, analizando la percolación de sitios para distintos valores de ocupación de la red. Se muestra una manera de evaluar muy rápidamente, y en paralelo, la capacidad de cada celda de propagar una señal proveniente de cualquier vecino cercano y generar un clúster. Se implementa también el generador de muestras aleatorias, para cargar la configuración de la red en cada paso de Monte Carlo. Una vez implementado el sistema se verifica la calidad de los datos obtenidos y se hace un análisis comparativo de la performance alcanzada en esta plataforma, con la lograda en una computadora personal moderna. De lo anterior surge una mejora, en la velocidad de procesamiento, mayor a un orden de magnitud. Todo el diseño se describe en VHDL y finalmente se proponen algunas posibles mejoras sobre el mismo.**

Palabras claves; Percolación, Monte Carlo, FPGA, HPRC, VHDL.

I. INTRODUCCION

En el campo de HPC (High Performance Computing), la mejora de la performance, es el principal objetivo que se debe alcanzar. Este incremento en la potencia de cálculo se ha logrado, al comienzo, a través computadoras especiales, luego utilizando clúster de GPP (General Propose Processors), posteriormente con la incorporación de procesadores con varios núcleos y más recientemente con clúster de GPU(Graphics Processing Units). Junto a esto, y con el objeto de lograr mayor velocidad de procesamiento, se ha logrado aumentar la frecuencia de trabajo de los procesadores. La idea de incorporar varios núcleos es poder realizar varias tareas simultáneamente, pero la asignación a cada núcleo, de las tareas que se deben realizar en paralelo no es una tarea trivial y no siempre se logra la mejora de performance esperada. Por otra parte, el aumento de la frecuencia de trabajo presenta un problema serio, debido a la potencia que es necesario disipar en los circuitos semiconductores. El uso de FPGA resulta una alternativa interesante, para algunas aplicaciones en HPRC (High Performance Reconfigurable Computing) dado que estos dispositivos, se pueden configurar con mayor libertad para realizar tareas en paralelo, frente a la que la que ofrecen los procesadores multicores. También hay que señalar que las FPGAs han logrado un aumento sostenido de la frecuencia de trabajo, de una generación a la próxima, con un consumo de potencia mucho más razonable que el que aparece en los casos de GPPs o GPUs. Esto hace que actualmente, estos dispositivos con lógica programable logren doblar su performance entre generaciones sucesivas [1].

Desde hace algunos años hay un creciente interés en desarrollar plataformas basadas en FPGAs, destinadas a trabajar en el área de HPRC [2] [3]. Con ellas se ha logrado, en algunos casos, conseguir importantes ventajas frente las otras alternativas existentes [4] [5]. Este tipo de plataformas, en su gran mayoría, están formadas por varias FPGAs interconectadas, con distintas arquitecturas, en algunos casos en combinación con una CPU construida con GPP. Estos sistemas presentan altos costos y mucho tiempo de desarrollo. Por ello, se dice, que para que se justifique este tipo de solución, es necesario lograr una mejora en velocidad de procesamiento de dos órdenes de magnitud con respecto a los GPP y un orden de magnitud respecto a los GPU [5]. Existen muy pocos antecedentes del uso de FPGA aplicadas al fenómeno de Percolación [6]

La creciente oferta de nuevas FPGAs, alienta a implementar algunos sistemas destinados a estas aplicaciones utilizando una única FPGA. Esto se debe al aumento de lógica disponible, nuevas funcionalidades y un costo razonable. Este trabajo muestra como se puede simular el fenómeno de percolación, utilizando el método de Monte Carlo, con un tipo de plataforma de hardware basado en una FPGA de un costo cercano a 100 dólares.

II. PERCOLACIÓN

La teoría de percolación estudia, entre otras cosas, el problema que se presenta, cuando fluido intenta pasar de un lado a otro en un determinado medio, e investiga las condiciones que se deben cumplir para que esto ocurra. Esta teoría permite estudiar otros fenómenos, tales como, la propagación de incendios en un bosque [7], la difusión de enfermedades contagiosas [8], la filtración de un líquido en medios porosos [9, 10], el estudio de conductores iónicos dispersos [11, 12], etc. Existen dos modelos de percolación: percolación de enlaces y percolación de sitios, en este trabajo se va a hacer referencia, solo al modelo de percolación de sitios en una red cuadrada homogénea de dos dimensiones.

Con pequeños cambios, lo desarrollado aquí, puede extenderse al modelo de percolación de enlaces. Para explicar brevemente en qué consiste el fenómeno de percolación en dos

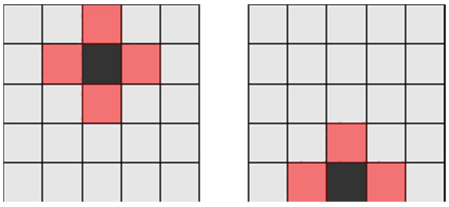

Fig. 1 Ejemplo de vecinos próximos de un sitio

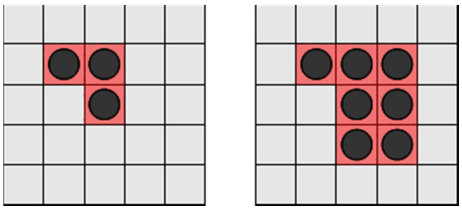

Fig 2 Clúster de 3 y 7 sitios

dimensiones, se parte de una red o matriz, donde cada uno de sus sitios admite solo dos estados posibles: libre u ocupado. Además se considera que un vecino próximo de cualquier sitio, es aquel que tiene un lado en común, por lo tanto (salvo en los bordes de la red) todos los sitios tienen cuatro vecinos próximos como lo muestra la Figura 1.

La posibilidad que un sitio esté ocupado, está dada por una probabilidad ρ. Esta probabilidad puede depender, de varios parámetros, por ejemplo, entre otras cosas, del estado de sus vecinos. En un primer estudio se supone que la probabilidad de ocupación de un sitio, es independiente de sus vecinos. Por lo tanto, cada sitio es ocupado con una probabilidad ρ independiente del estado de sus vecinos. Cuando dos sitios que son vecinos próximos están ocupados se dice que forman un clúster de dos sitios. Si dentro de la red bajo estudio hay *k* sitios ocupados, dentro de los cuales es posible moverse pasando de un sitio a otro, por lados comunes se tiene un clúster de *k* sitios. La Figura 2 muestra dos ejemplos, un clúster formado por 3 y otro formado por 7 sitios.

El modelo más sencillo de analizar, es cuando se considera una red bidimensional de longitud infinita, en este caso para distintos valores de probabilidad ρ se forman clúster de distintos tamaños. Existe un valor de probabilidad, denominada *probabilidad crítica* ρ_c o *concentración crítica*, en la cual emerge un clúster de tamaño infinito, en este caso, se dice que el sistema ha percolado, esto significa que la red se encuentra totalmente conectada por éste clúster.

Una manera de estudiar la percolación es usar el método de simulación de Monte Carlo y trabajar con redes de dimensión finita, a medida que las redes aumentan de tamaño se obtiene información más precisa de este fenómeno. Sin embargo, aun con redes de dimensiones pequeñas se puede estudiar varias propiedades de la percolación, a través de la teoría de escaleo finito.

Un método para estudiar y analizar la percolación consiste en determinar la probabilidad que una red de $M = LxL$, donde L es la dimensión lineal de la red, a una dada concentración percole [13], ver figura 3.

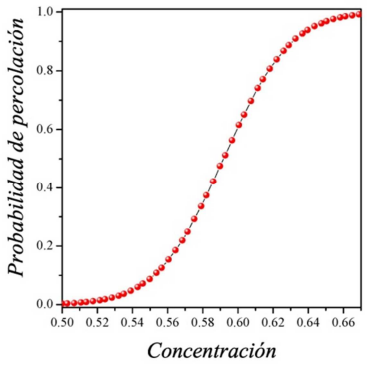

Fig. 3: Curva de la probabilidad de percolación para una red cuadrada

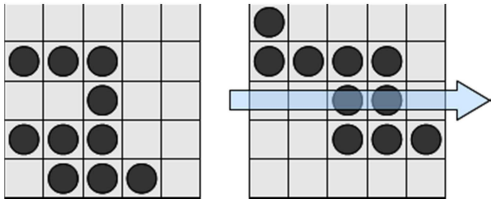

Fig. 4 Ejemplo de clusters

Para obtener cada punto de esta curva se realiza un promedio de los resultados obtenidos en los siguientes pasos, se carga la red con la cantidad deseada de sitios y se determina si el sistema con esta configuración percola, es decir, se determina la existencia, de al menos un clúster que vincule ambos lados de la red. En la figura 4 se ve un ejemplo, donde hay un clúster que no cumple la condición enunciada y otro que si lo hace.

III. DESCRIPCION DEL ALGORITMO

Se llama cubrimiento, ocupación o concentración de la red al cociente del número de sitios ocupados *n*, sobre el total de sitios de la red ($M=LxL$), es decir n/M.

Para obtener la probabilidad de percolación de una red de dimensión LxL, para distintas concentraciones, con un procesador secuencial, es necesario realizar los siguientes pasos: sortear una distribución aleatoria de sitios ocupados dentro de la red, recorrer los LxL sitios de la red, detectar los clúster existentes [14] y finalmente verificar, si existe un clúster percolante, es decir, un clúster que atraviesa la red de un lado a otro. Dado que se usa el método de Monte Carlo, es necesario realizar el proceso descripto un gran número veces para que el resultado sea confiable. Luego de calcular la probabilidad para ese número de sitos ocupados, se selecciona otra configuración de red y se repite el procedimiento anterior para todos los puntos de la curva. La figura 5 muestra la probabilidad de percolación para dos redes con diferentes valores de L. El punto donde ambas curvas se cortan indica el umbral de percolación. Como se puede ver, este procedimiento requiere mucho tiempo cuando se lo implementa con procesadores secuenciales.

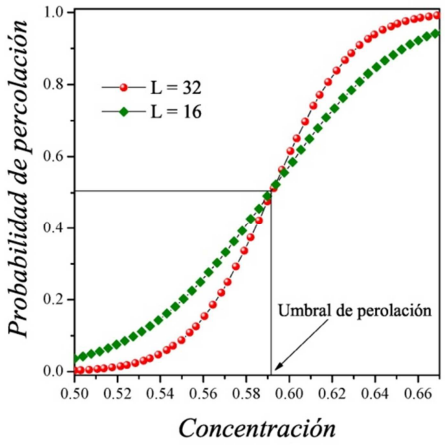

Fig. 4 Curva de Probabilidad de percolación para diferentes valores

IV. IMPLEMENTACIÒN EN LA FPGA

El sistema propuesto se implementó en la placa Nexys3 Spartan-6 FPGA Board de la empresa Digilent [16]. Se trata de una plataforma de desarrollo basada en la FPGA Xilinx Spartan6 XC6LX16-CS324. Esta placa cuenta con varios periféricos conectados a la misma. Para esta aplicación se utilizaron entradas digitales para el ingreso de los datos y un display para visualizar el resultado. También se utilizó una interfaz serial, para contar con la posibilidad de ingresar y leer datos desde una computadora personal.

Los datos que ingresan al sistema son: el cubrimiento de la red y el número de pasos de Monte Carlo con que se va a trabajar. El tamaño de la red se fija antes del proceso de síntesis, mientras que la ocupación de la red y el número de iteraciones se pueden variar en cualquier momento con el sistema funcionando. El diseño consta de dos módulos principales y otros complementarios como se muestra en la figura 5. El primero está destinado a generar una distribución aleatoria de sitios ocupados de la red y el segundo, es el encargado de determinar si existe, para esa distribución, al menos un clúster percolante. Los módulos complementarios corresponden al módulo de control, el módulo de entrada/salida y el sintetizador.

Cada paso de Monte Carlo consiste en generar una muestra aleatoria de sitios ocupados y verificar si la red percola. Cada vez que se produce la percolación se incrementa un contador que lleva la cuenta del número de percolaciones para esa ocupación. Cuando se alcanza la cantidad de pasos de Monte Carlo seleccionada, el estado del contador, actualiza la salida del sistema. La probabilidad para esta ocupación, se obtiene dividiendo este valor por el número de iteraciones con que se trabaja y se determina la probabilidad de percolación para esa cantidad de sitios ocupados.

A. Generador Pseudoaleatorio

Dado que la generación de la configuración de la red se va a realizar a partir de una máquina de estados finitos, sin ninguna señal externa, se trabajará inevitablemente, con una secuencia finita de valores, por lo tanto el generador será pseudoaleatorio. Para lograr una aceptable calidad de las muestras se tendrá en cuenta que la secuencia se repita con un ciclo bastante mayor que el número máximo de iteraciones usadas, y que exista baja correlación entre una muestra y la siguiente. El generador usado se basa esencialmente en un generador de números pseudoaleatorios utilizando registros de desplazamiento realimentados leap ahead [15].

Dado que la salida se toma en paralelo de la salida de algunos flip-flop del registro de desplazamiento, este generador se adapta mejor que un simple LSFR, por la alta correlación que existe entre una muestra y la siguiente en un FSFR simple. En la figura 6 se muestra un esquema del generador utilizado.

Para generar la configuración de la red de $n = LxL$ sitios, se implementa una memoria de n palabras, de un bit de longitud. Esta memoria se carga con una cantidad de unos, igual a la cantidad m de sitios ocupados que se desee. Una vez cargada la memoria, su contenido se transfiere al módulo detector de percolaciones para ser evaluada.

También se implementa un generador de números pseudoaleatorios, denominado LAPAR (Leap Ahead Paralelo), cuya salida se usa como índice para indexar las posiciones de la memoria antes mencionada.

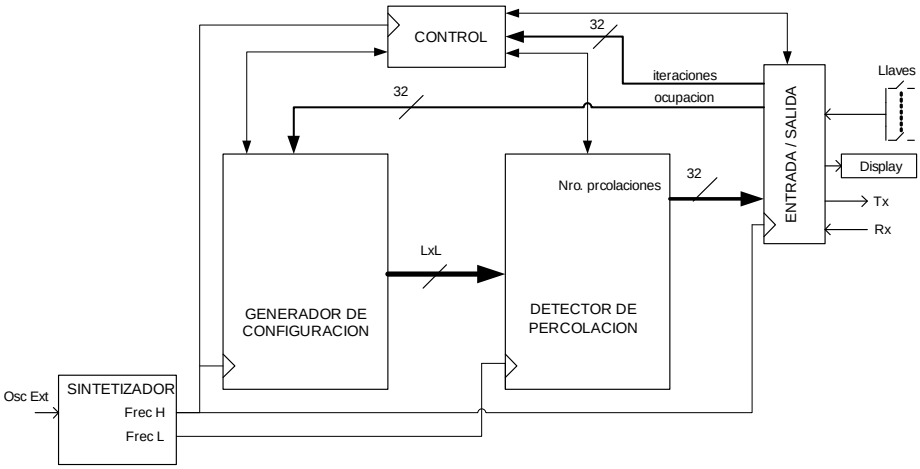

Fig. 5 Esquema general del sistema

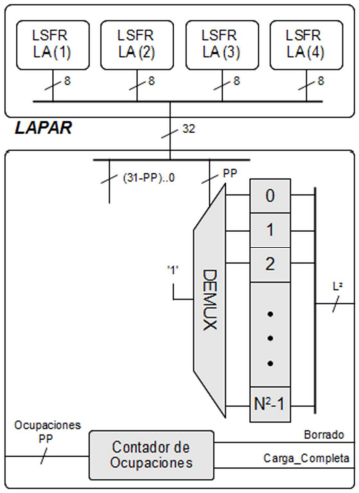

Fig. 6 Esquema del generador pseudorandom

Este generador basa su funcionamientos en cuatro registros de desplazamiento linealmente realimentados (LFSR) del tipo leap-ahead (LA) funcionando en paralelo.

La configuración de la red con m sitios ocupados se realiza de la siguiente manera: se borra el contenido de la memoria, el generador de números pseudoaleatorios se conecta a las líneas de dirección de la memoria y cada vez que se genera un nuevo valor, se lee el contenido de la memoria que corresponde a esa dirección. Si el dato leído es uno, no se hace nada y se espera la llegada de una nueva dirección. Si el dato leído es cero, el contenido de la memoria se cambia por uno y se incrementa un contador. Cuando este contador alcanza el valor *m*, significa que la memoria tiene un nuevo estado listo para configurar la red. De lo explicado surge, la necesidad que se requieran al menos *m* ciclos de lectura/escritura para generar una nueva configuración de red. Como se verá más adelante, esta cantidad de ciclos, son necesarios para cada paso de Monte Carlo y resulta ser la mayor limitación que presenta este diseño, si se tiene en cuanta la velocidad de procesamiento. Con el fin de acelerar la generación de una nueva configuración, en caso de trabajar con una ocupación igual o superior al 50% del tamaño total de la red, se realiza el mismo procedimiento pero la matriz se inicializa con unos y se carga con *n-m* ceros. Con esto, el mayor retardo en la generación de un nuevo estado, se presenta cuando se carga una configuración con un 50% de sitios ocupados, y disminuye para cualquier otra concentración.

B. Detector de Percolación

Este módulo es el encargado de detectar la percolación. Existen dos tipos de detección de percolación en redes finitas, una que considera condiciones de borde abiertas, y otra, que considera condiciones de borde periódicas. Para este trabajo se asumieron condiciones de borde periódicas, es decir, se supone a la primera fila está en contacto con la última y vise versa.

En estas condiciones se debe determinar si existe un clúster percolante. Para esto, se supone que se parte de un borde lateral cualquiera, por ejemplo del borde izquierdo de la red y se analiza si algún sitio ocupado de la primera columna pertenece a un clúster percolante.

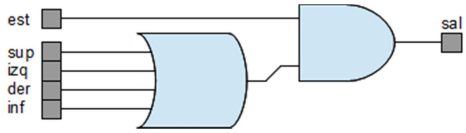

Fig. 7 Esquema de un sito de la red

Un clúster percolante se puede asimilar a un camino eléctrico ente dos sitios extremos de la red, donde cada sitio del clúster es conductor sólo si su estado esá ocupado. Esta es la idea que se toma como base para implementar la detección con la FPGA.

La arquitectura del detector de percolación puede considerarse dividida en dos partes fundamentales. Una que indica el estado de cada sitio de la red, y la otra que determina la presencia de algún clúster percolante. El primer módulo es, por lo tanto, una memoria de LxL bits, y como se ha implementado usando las CLB de la FPGA, el elemento básico es un Flip-Flop. El segundo módulo está compuesto netamente por lógica combinacional, siendo cada elemento básico la celda mostrada en la figura 7.

Las entradas a la compuerta OR corresponden a la salida de los 4 vecinos, el estado del sitio, es el estado del FF y la salida de la compuerta AND indica si es sitio pertenece a un clúster o no. Se puede interpretar que, si la salida de la compuerta AND está en alto, este sitio propaga la señal de algún vecino. Por lo tanto, para saber si existe al menos un clúster percolante para esa configuración, se debe excitar con “unos”, todas las entradas del lado izquierdo de la red y detectar si al menos una de las salidas de lado derecho está en uno. Para detectar la presencia de al menos un “uno” en la columna de la derecha, se conectan las *L* salidas de esa columna a una compuerta OR.

Los dos módulos principales del sistema trabajan en paralelo, ya que mientras se está generando la próxima configuración de la red, se detecta la presencia o no de un clúster percolante, correspondiente a la configuración anterior. La salida de la compuerta OR se conecta a la entrada de habilitación de un contador en el módulo de control y este se incrementa cuando detecta la percolación.

C. Módulos Complemetarios

Para completar el diseño del sistema se agregaron los siguientes módulos:

1) Control

Es el módulo encargado de sincronizar todas las tareas y se inicia con un reset del sistema. A partir de ese momento, lee el dato de ocupación, el número de iteraciones seleccionado externamente y comienza con la ejecución del algoritmo. Una vez obtenido el resultado, lee nuevamente los datos de entrada y realiza nuevamente la misma tarea. El número de percolaciones obtenido se muestra en el display y también es enviado por el puerto serial para su lectura en una PC. En caso de recibir nuevos parámetros para el cálculo, ya sea desde la placa o desde la PC, estos quedan en un registro, que se leen una vez terminado el cálculo anterior.

2) Entrada Salida

Este módulo y el siguiente sirven para seleccionar los datos con que se va a trabajar y también para mostrar los resultados.

Cuando los datos se ingresan desde la placa de desarrollo, se hace uso de 8 llaves de dos posiciones, en combinación con pulsadores presentes en la placa. El resultado obtenido (de 32 bits) se muestra en el display existente.

3) Entrada Salida Serial

Con el fin de simplificar y manejar el equipo desde una PC se agregó una interface serial tipo UART a través del puerto USB presente en la placa.

V. Reslutados Obtenidos

El diseño se describió en VHDL y es portable a los diferentes dispositivos de lógica programable existentes. Es necesario señalar que se utilizó un módulo sintetizador de frecuencia, para generar la frecuencia principal del sistema. Este sintetizador toma como referencia un oscilador externo de 100 MHZ disponible en la placa Este es el único módulo del diseño que no es portable, ya que es un IP Core propietario de XILINX compatible con esta FPGA. Esto no resulta una pérdida importante de portabilidad, ya que casi todas las FPGAs del mercado poseen un módulo equivalente. Es importante contar con este recurso, para poder seleccionar la máxima frecuencia que permita el diseño y de esta manera lograr la mayor performance posible.

Se implementaron redes de 8x8 16x16 y 32x32 y se obtuvieron las curvas mostradas en la figura 8. Los datos obtenidos se verificaron de dos maneras distintas, la primera utilizando una computadora personal con un software para este tipo de tareas y un segundo método que resulta de determinar el cruce de las curvas para redes de distinto tamaño. Este punto de cruce debe corresponder con el valor del umbral de percolación, que para este tipo de red resulta 0,52927[8].

Cuando se compararon los datos obtenidos en esta plataforma y los obtenidos en la PC, la diferencia resultó, para todos los casos menor a 0,1% para 10^6 pasos de Monte Carlo. En el segundo caso, el cruce de las curvas, se estimó a través de una interpolación líneal entre los puntos más cercanos, por lo que la determinación del error no es tan exacta. Sin embargo el error estimado resultó menor al 0,2% para el mismo número de iteraciones.

VI. Analisis de performance

La performance de esta aplicación tiene como principal limitación, el módulo encargado de la generación aleatoria del estado de la red para cada paso de Monte Carlo. El estado de la red se obtiene, como se dijo anteriormente, cargando una memoria de LxL direcciones de un bit de longitud. La cantidad de ciclos de reloj necesarios para configurar la red depende, en primer lugar de la ocupación de la misma, pero también puede variar para una misma ocupación, dada la naturaleza aleatoria del generador de direcciones de memoria. Esto se debe a que es necesario leer el estado de la misma para comprobar si el contenido para esa dirección no ha sido escrito previamente. La mayor probabilidad que la dirección sorteada ya haya sido generada, aumenta con el número de sitios ocupados. Por lo que se ha explicado anteriormente, la mayor cantidad de sitios que se sortean corresponde a un cubrimiento del 50% de la

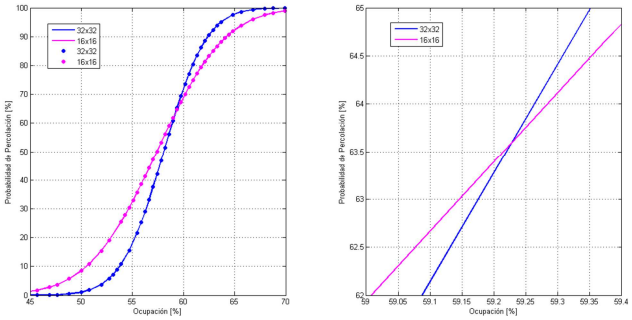

Fig. 8 Curvas de Percolación para 32x32 y 16x16

red. Por lo tanto, para este valor de ocupación se da el mayor retardo del módulo.

Por otro lado, el tiempo necesario para estabilizar la salida del detector de percolación, es menor que el requerido por el generador para configurar la red y varía con la ocupación de la misma. El caso más desfavorable en el detector se da cuando se forma el clúster percolante de mayor longitud (no el más grande), que para una red de 2^n x2^n está formado por 2^{2n-1} sitios (lo que equivale a una ocupación del 50%). En el caso de una red de 32x32, el tiempo necesario para que se estabilice la señal de salida del detector, es el tiempo que tarda en propagarse la señal a través de 512 sitios ocupados de la red. Dada la arquitectura de la FPGA utilizada, cada sitio de la red se implementa en una Slice. Por ello, el mayor tiempo de propagación (Tmax) en términos de retardo de lógica es de 512 veces el retardo máximo de la Slice de la FPGA. Este retardo es de 0,26 ns, por lo tanto Tmax = 512 * 0,26 ns = 133,12 ns. Si se estima un retardo de ruteo del 50% del retardo empleado en la lógica, el tiempo necesario para que se estabilice la salida del detector llega a Td = 199,68 ns. Este tiempo limita la frecuencia de trabajo del detector a Fd = 1 / Td = 5 MHz.

Considerando que, para una ocupación del 50%, son necesarios al menos 512 ciclos de reloj para generar una configuración de red en el generador pseudoaleatorio, la frecuencia máxima a la que podría funcionar generador es Fg = Fd * 512 = 2,56 GHz, sin que el detector funcione por sobre su frecuencia máxima. Sin embargo, el diseño implementado en la FPGA no consigue alcanzar dicho valor, siendo las frecuencias máximas limitadas por la lógica del módulo generador.

Las frecuencias máximas de trabajo para los sistemas de 8x8, 16x16 y 32x32, tomadas del reporte de tiempo de la herramienta de diseño, se muestran en la tabla 1. Se comparó este algoritmo usando la FPGA y una PC (Procesador Athlon II P320 Dual-Core a 2,1 GHz y 3,49GB de RAM).

TABLA I Frecuencias máximas

Dimensión de la red	Frec. Max [MHz]
8x8	137.043
16x16	119.289
32x32	97.704

La tabla II muestra los tiempos que se consumen en ambos casos para distintos tamaños de red. Como puede observarse en la tabla se logró una mejora en la velocidad del algoritmo de 20 veces.

TABLA II Tiempos de ejecución del algoritmo

Dimensión de la red	Tiempo [Seg]		Relacion PC/FPGA
	PC	FPGA	
8x8	11	0,52	21,15
16x16	43	2,09	20,57
32x32	172	8,38	20,53

VII. RECURSOS UTILIZADOS

En la tabla III se muestra los recursos de hardware utilizados para distintos tamaños de red. Como se desprende de la tabla, el crecimiento de los recursos es exponencial, tanto para la lógica combinacional como la secuencial, por lo tanto, se advierte una limitación para implementar redes de grandes dimensiones utilizando esta arquitectura.

Para ampliar el tamaño de la red se debe utilizar una FPGA de mayor capacidad o cambiar la arquitectura del diseño, como se sugiere mas adelante.

TABLA III Recursos de hardware utilizados

Dimensión de la red	Slice LUT's		Slice Registers	
	(%)	Total	(%)	Total
8x8	4 %	376	2 %	380
16x16	12 %	1102	4 %	768
32x32	31 %	2849	12 %	2347
64x64	145 %	13232	46 %	8437

En este diseño, la memoria necesaria para la generación de la configuración de la red, no se ha implementado utilizando los bloques de RAM disponibles en el dispositivo, a fin de darle portabilidad al diseño. Ésta es una de las razones que impiden, trabajar con redes de mayor dimensión en esta FPGA.

VIII. CONCLUSIONES

Se ha logrado implementar en una plataforma de bajo costo basada en FPGA, un sistema que permite realizar la simulación de Monte Carlo de redes bidimensionales de hasta 32x32. Por simplicidad se han usado longitudes que son potencias de 2, aunque es posible extenderla a cualquier número, agregando un pequeño módulo al bloque generador, que ignore las direcciones de memoria que están por encima del tamaño de la red.

Se ha alcanzado una importante mejora en los tiempos de simulación que permiten que un trabajo que en una PC moderna demora 1 hora se realice en menos de 3 minutos. Se cuenta con un diseño escalable y portable, que puede ser directamente implementado en cualquier FPGA. Entre las desventajas hay que señalar dos fundamentales: por una parte, el tamaño de la red alcanzado con esta plataforma es pequeño para muchas aplicaciones y por otra parte el tiempo de diseño es varias veces, el que consume la programación del algoritmo en C. Sin embargo para un número importante de simulaciones, trabajar con esta plataforma puede representar un ahorro importante de tiempo.

IX. TRABAJOS FUTUROS

Sobre la base del diseño propuesto, se puede lograr una importante mejora en la performance y en el aumento del tamaño de la red, si se sacrifica portabilidad.

Como se mostró en la tabla I la frecuencia máxima se ve limitada por el generador de la configuración de la red. Dado que este bloque se puede implementar usando una memoria RAM del tamaño de la matriz y registros de desplazamiento de 32 bits, es posible implementar dos generadores de red no correlacionados, trabajando en paralelo. Con esto se podría reducir, a la mitad, el tiempo de procesamiento. En cuanto al aumento del tamaño de la red, se podría almacenar en memoria la configuración inicial de la red, eventualmente en memoria externa a la FPGA, y procesar la red por partes sobre el hardware disponible. Este procedimiento generaría una pérdida de performance porque se trata de un proceso iterativo, ya que cada vez que se genere un nuevo clúster percolante en cada parte procesada, hay que pasar esa información al clúster vecino hasta que el sistema se estabilice. Sin embargo, como el mayor retardo está en la generación de la configuración de la red, que es una tarea que se realiza en paralelo, se dispone de más tiempo para la ejecución de esta tarea y la pérdida de la performance puede no reflejarse en el sistema.

REFERENCIAS

[1] Prasanna Sundararajan "High Performance Computing Using FPGAs," Xilinx, September , 2010

[2] Rob Baxter1, Stephen Booth,Mark Bull, Geoff Cawood,James Perry, Mark Parsons,Alan Simpson, Arthur Trew. "Maxwell – a 64 FPGA Supercomputer".2008

[3] Rob Baxter1, Stephen Booth, Mark Bull, Geoff Cawood, James Perry, Mark Parsons, Alan Simpson, Arthur Trew "The FPGA High-Performance Computing Alliance Parallel Toolkit". 2007

[4] F. Belletti, M. Cotallo, A. Cruz, L. A. Fern´andez,A. Gordillo, A. Maiorano, F. Mantovani, E. Marinari,V. Mart´ın-Mayor, A. Munoz-Sudupe, D. Navarro,S. Perez-Gaviro, M. Rossi, J. J. Ruiz-Lorenzo,S. F.Schifano, D. Sciretti, A. Taranc´on, R. Tripiccione,J. L. Velasco. "IANUS: Scientific Computing on an FPGA-Based Architecture",2008

[5] Khaled Benkrid, Esam El-Araby, Miaoqing Huang, Kentaro Sano, and Thomas Steinke "High-Performance Reconfigurable Computing" 2012.

[6] A reconfigurable Monte-Carlo clustering processor (MCCP) Cowen, C.P. ; Monaghan,S. FPGAs for Custom Computing Machines, 1994. Proceedings. IEEE Workshop on Digital Object Identifier: 10.1109/ FPGA.1994.315600 Publication Year: 1994 , Page(s): 59 - 65

[7] C. L. Henley, Phys. Rev. Lett. 71, 2741 (1993).

[8] C. Moore and M. E. J. Newman, Phys. Rev. E 61, 5678 (2000).

[9] D. Stauffer and A. Aharony, Introduction to Percolation Theory, 2nd ed. (Taylor & Francis, London, 1985).

[10] M. Sahime, Applications of percolation Theory (Taylor & Francis, London, 1994); Flow and transport in Porous Media and Fractured Rock (VCH, Weinhein, Germany, 1995).

[11] C. C. Liang, J. Electrochem. Soc. 120, 1289 (1973).

[12] A. K. Shukla, V. Sharma, in: B.V.R. Chowdari, S. Chandra, S. Singh, P.C. Srivastava (Eds.), Solid State Ionics: Materials Applications, World Scientific, Singapore, 1992, p. 91.

[13] F. Yonezawa and S. Sakamoto, M. Hori, Phys. Rev. B, 40, 636

[14] J. Hoshen and kopelman, Phys. Rev. B, 14, 3438 (1976).

[15] C. M. Gonzalez, C. A. Gayoso, L. J. Arnone y M. R. Rabini. Implementacion de generadores de n úmeros pseudoaleatorios utilizando registros de desplazamiento realimentados leap ahead. Libros de memoria uea2012

[16] http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pd