Construcción de un banco de prueba de Autómatas Celulares

Diego J. Serrano

Laboratorio de Investigación de Software LIS
Dpto. Ingeniería en Sistemas UTN-FRC

diegojserrano@gmail.com

Resumen:

Los autómatas celulares son modelos matemáticos discretos que consisten en una grilla de componentes idénticos llamados células, las cuales interactúan entre sí con sus vecinas más cercanas. Cada célula se encuentra en un momento determinado en un estado perteneciente a un conjunto discreto de estados posibles y el autómata evoluciona en el tiempo en unidades discretas a través de la modificación de los estados de sus células [1].

Numerosas investigaciones requieren simular la evolución temporal de autómatas celulares durante una gran cantidad de pasos con el objeto de analizar su comportamiento, identificar similitudes con otras configuraciones de autómatas y encontrar coincidencias con el comportamiento de sistemas naturales.

El objetivo del proyecto es generar distintas piezas de software que permitan simular la evolución de diversos autómatas celulares para que sean utilizados por investigaciones específicas. Se presta especial atención al aprovechamiento adecuado del hardware y software disponible en la actualidad y en el desarrollo de algoritmos optimizados que permitan realizar simulaciones largas en relativamente poco tiempo de CPU. Asimismo se busca aprovechar el poder de cómputo de los microprocesadores de placas de video e incluso algún mecanismo de paralelización, como clústeres de high performance computing.

Contexto:

Esta línea de I/D pertenece al proyecto "Relaciones entre los momentos de aprendizaje y reconocimiento de las redes neuronales artificiales y la evolución espacio-temporal de los autómatas celulares"[1] radicado en la de la Universidad Tecnológica Nacional – Facultad Regional Córdoba. El desarrollo del banco de pruebas está incubado en el Laboratorio de Investigación de Software de la misma unidad académica.

Introducción:

Para desarrollar software que simule la evolución de un autómata celular unidimensional deben ponerse en consideración ciertas alternativas con respecto a:

- Representación del estado de autómata
- Cálculo de la evolución de una etapa
- Presentación y almacenamiento de toda la evolución
- Uso de hardware.

Representación del estado de autómata

Para representar el estado de un autómata unidireccional la alternativa más directa o *naive* es la de un arreglo unidimensional donde cada posición almacene un entero o un carácter representando el estado de una célula.

[1] http://www.institucional.frc.utn.edu.ar/sistemas/Investigacion/PROYECTOS2008/PROYECTOS2008_A11.pdf

Este acercamiento permite accesos muy rápidos a cada célula pero ocupan mucha memoria. En los lenguajes de programación actuales los tipos de datos enteros o de carácter ocupan 16 o 32 bits, utilizando arreglos de estos tipos de datos para almacenar el estado de una única célula desperdicia muchos bits. Suponiendo células cuyos estados puedan representarse con uno o dos bits, en cada variable entera de 32 bits se puede almacenar el estado de 32 o 16 células.

Resulta de interés analizar la creación de estructuras de datos especializadas que permitan almacenar el estado del AC empaquetando los bits según el dominio del estado de cada célula. Esto permite mantener en memoria real (es decir sin acceder a la memoria virtual o realizar paginación manual) autómatas muy grandes y evoluciones de muchos pasos. Suponiendo un proceso que pueda acceder a 2GB de memoria real y un autómata de 100.000 células de estados binarios, se pueden mantener en memoria unos 170.000 pasos de evolución. Utilizando un entero de 32 bits por cada célula, poco más de 5000 pasos ocuparían los mismos 2GB.

Almacenar en memoria una evolución muy larga brinda la posibilidad de realizar análisis sobre la evolución, como la identificación de ciclos o la medición de entropía, siempre con todos los datos en memoria y sin realizar accesos a disco que son notablemente más lentos.

Debe analizarse, sin embargo, la complejidad de almacenar los estados en forma empaquetada, pues el acceso directo a una célula a fin de consultar o modificar su estado es considerablemente más complejo que con un arreglo simple. Cada operación requiere cierto esfuerzo en ubicar y desempaquetar (mediante máscaras y desplazamientos de bits) la célula buscada. Si bien este esfuerzo no es grande en términos computacionales y posee la misma complejidad temporal que un acceso directo a un arreglo, al efectuar múltiples operaciones se puede provocar una diferencia de performance notable.

Cálculo de la evolución de una etapa

El cálculo de cada evolución implica realizar una misma operación funcional a cada célula. Dicha función utiliza como argumentos el estado de la célula actual y los de sus vecinas. Según la vecindad del autómata, el cálculo de cada nuevo estado puede obtenerse como una función de 3, 5 o más células.

Dependiendo de la naturaleza del autómata el cálculo puede realizarse de varias formas. Si las células son binarias la función de evolución sólo aplica operaciones de bits. Según Wolfram en este caso la evolución se puede realizar de una forma muy simple desplazando el estado completo del autómata un bit hacia cada lado y copiando el resultado de los desplazamientos. Luego solo se deben realizar operaciones de bits entre todo el autómata y sus copias desplazadas.[1]

Sin embargo, este método solo funciona si los bits están empaquetados y almacenados en una estructura de datos que permita desplazar a todo el autómata a la vez. Si el autómata posee más celdas que el tamaño del tipo de datos más grande del lenguaje utilizado, se deben realizar acarreos de los bits que se pierdan en cada desplazamiento.

Otro enfoque puede plantear el recorrido secuencial de todo el autómata aplicando una función dada a cada célula y sus vecinas. Si el acceso directo a cada célula es costoso se puede plantear un buffer con el tamaño de la vecindad, aplicando sobre el buffer la función de evolución y desplazando las células de a una a la vez. De esta forma se realiza un solo acceso a cada célula, haciendo funcionar al buffer como una ventana que muestra sólo células vecinas. Esta variante realiza un acceso por célula mientras que la variante ingenua realiza por cada célula tantos accesos como el tamaño de la vecindad.

En el caso de los autómatas binarios con vecindades de 3 células en el que cada uno es rotulado con un número que representa la función de evolución según sus salidas, puede ser necesario obtener las operaciones binarias correspondientes sólo conociendo su rótulo. Para ello simplemente debe encontrarse la

función binaria minimizada con algún método tal como el mapa de Karnaugh. Dado que este método es relativamente difícil de implementar con un programa, se puede utilizar como alternativa el algoritmo de Quine-McCluskey.[2]

Cuando las células no son binarias, las funciones de evolución pueden incluir operaciones aritméticas sobre el valor entero de cada estado. Para poder desarrollar un software que pueda evolucionar diversos tipos de autómatas debe considerarse la utilización de algún lenguaje de programación funcional en el que la propia función de evolución pueda ser enviada como un parámetro al procedimiento de evolución.

Presentación y almacenamiento de toda la evolución

La evolución que se obtenga mediante el software debería ser presentada en pantalla de alguna forma adecuada para su consulta y almacenada para su posterior análisis.

Para estas tareas también debe ponerse en consideración el gran volumen de datos que puede generarse con autómatas grandes o evoluciones en mucho tiempo.

La presentación en pantalla puede realizarse con operaciones primitivas de gráficos, dibujando un pixel o una región cuadrada por cada célula, utilizando colores diferentes por cada estado. Esto resuelve el problema de la presentación para pocas células y para cualquier cantidad de estados. Sin embargo si la cantidad de pasos a analizar es mayor que la resolución vertical de la pantalla debe analizarse la alternativa de mostrar todas las evoluciones permitiendo hacer desplazamiento (*scroll*) en la pantalla o mostrar sólo las últimas. Si bien no es complejo permitir desplazamientos, la memoria que se ocupe por el proceso que muestre la evolución terminará siendo varias veces mayor que todo el histórico mismo del autómata.

Puede ser razonable generar un archivo de imagen con la evolución mientras se va calculando cada nuevo estado o al finalizar la simulación. El archivo de imagen puede luego ser consultado con un software estándar que seguramente implemente algún tipo de paginación sobre la imagen y que además ofrezca la posibilidad de hacer zoom sobre la misma.

Como puede ser necesario el análisis posterior de la evolución a los efectos de identificar características intrínsecas del autómata, de la función de evolución o del comportamiento con diferentes configuraciones iniciales, debe plantearse la necesidad de almacenar toda la evolución en archivos que puedan ser consultados por otros programas. Para ello pueden generarse archivos planos, binarios con la misma codificación que los estados en memoria, o con formatos que permitan el análisis léxico y sintáctico con herramientas previamente disponibles, tales como XML o JSON.

Uso de hardware

Existen en la actualidad controladoras de video con microprocesadores y memoria con capacidad y velocidad similares a los de una CPU estándar. Algunas de esas controladoras permiten correr programas que realicen operaciones con la GPU y su memoria pero sin utilizar las funciones gráficas si no aprovechando su poder de cómputo como un segundo procesador [3]. Debería evaluarse el uso para potenciar las simulaciones aprovechando esta tecnología, ya sea con algoritmos paralelos o simplemente dividiendo las tareas, calculando la evolución con la GPU y almacenando el histórico en unidades de disco con la CPU principal.

De la misma forma debería analizarse la posibilidad de paralelizar los algoritmos a fin de utilizar instalaciones de HPC (*High performance computing*) para realizar simulaciones de una gran cantidad de pasos en menor tiempo.

Lineas de investigación y desarrollo

El proyecto tiene como líneas principales el estudio de diversos tipos de autómatas celulares y el desarrollo del banco de pruebas.

Por el lado del estudio puro de los autómatas celulares se requiere investigar acerca de los tipos de autómatas existentes, centrado por ahora en los unidimensionales. Esto involucra las distintas configuraciones de estados y su dominio, vecindades, tipos de funciones de evolución, configuraciones iniciales, diferencias con autómatas circulares, categorización de Wolfram, etc.

El desarrollo del banco de pruebas es esencialmente una investigación de algoritmos y estructuras de datos y de paralelización.

Resultados obtenidos / esperados

Actualmente se ha desarrollado una primera versión del banco de pruebas sin optimizaciones y utilizando en todo momento estrategias triviales. Una versión anterior del software fue presentada en el año 2010 en el "Segundo Encuentro De Jóvenes Investigadores En Neurociencias De Córdoba" organizado por la Facultad de Matemática, Astronomía y Física de la Universidad Nacional de Córdoba[2].

Se espera durante este año la finalización de un banco que implemente algoritmos y estructuras de datos más optimizados, con salidas adecuadas y que realice algún tipo de análisis sobre la evolución de los autómatas que haga correr.

Finalmente en una tercera etapa se debería desarrollar una versión que distribuya la carga del cómputo en subprocesos o en diferentes unidades de proceso, dentro de una misma computadora o en un cluster.

Formación de recursos humanos

Inicialmente los conocimientos y experiencias obtenidos contribuirán a mis tareas como investigador y como docente de asignaturas relacionadas con los algoritmos y estructuras de datos. Además, debido a que la línea de investigación pertenece a un proyecto acreditado, se pueden involucrar en la investigación y el desarrollo del banco de pruebas a becarios de investigación y a alumnos avanzados para que realicen su práctica profesional supervisada.

Bibliografía

[1] Wolfram, Stephen. "Cellular Automata and Complexity" ISBN 0-201-62664-0. Addison Wesley – 1994.

[2] Grimaldi, Ralph "Matemáticas discreta y combinatoria: Introducción y aplicaciones" ISBN 968-444324-2. Addison Wesley - 1997

[3] Farber, Rob "CUDA Application Design and Development" ISBN 978-0-12-388426-8. Elsevier - 2011

[2] http://www.famaf.unc.edu.ar/neuro/