Programación genética usando dispositivos digitales reconfigurables

Antonio Luque Estepa
aluque@zipi.us.es

27 de junio de 2001

Resumen

Los algoritmos genéticos son una forma rápida y robusta de resolver ciertos problemas computacionalmente difíciles. Se basan en la evolución en el tiempo de una serie de elementos. La programación genética utiliza estos algoritmos para crear programas de ordenador. Aquí se describe cómo los dispositivos lógicos rápidamente reconfigurables pueden ser usados para implementar de una manera eficiente la programación genética, y cómo la evolución se puede evaluar en dichos dispositivos.

Índice General

1 Introducción

Los dispositivos digitales programables, particularmente las FPGAs, son ampliamente utilizados para realizar rápidamente prototipos de sistemas digitales complejos. Dado el abaratamiento progresivo de estos dispositivos, hoy día pueden ser empleados para realizar de una manera fácil y barata computadores digitales simples, que más tarde se pueden emplear para resolver problemas numéricos más o menos complejos.

La inexistencia hasta hace relativamente poco tiempo de un dispositivo digital rápidamente reprogramable en hardware hacía que fuese difícil emplearlos para ejecutar de manera eficiente algoritmos complejos.

Entre estos algoritmos se encuentran los algoritmos genéticos, de probada eficiencia para resolver problemas de optimización o de búsqueda. Una etapa en particular de estos algoritmos (el cálculo de los diferentes valores de la función de evaluación) puede simplificarse enormemente si se codifica en hardware en vez de en software.

Como se verá, en algunos casos los cálculos de la función de evaluación son diferentes en cada iteración del algoritmo, por lo que es necesario un programa cambiante. Hasta la llegada de dispositivos rápidamente reconfigurables en campo, esto sólo podía lograrse mediante software. Ahora es posible diseñar un dispositivo digital que pueda reprogramarse a sí mismo en hardware para adaptarse al algoritmo.

2 Algoritmos genéticos

Los algoritmos genéticos son usados principalmente para resolver problemas de búsqueda o de optimización. Están originalmente inspirados en la evolución de los seres vivos y en las leyes de selección natural que garantizan la supervivencia de los más aptos. Como se explicará a lo largo de esta sección, los seres vivos más aptos son análogos a las mejores soluciones para un problema dado.

Históricamente, la primera vez que se formularon en forma matemática los principios básicos de la selección natural fue en 1975 [1]. Anteriormente, estos principios habían sido enunciados de una forma cualitativa por Charles Darwin en *El origen de las especies*.

Estos principios consisten básicamente en la competencia que se produce en la naturaleza entre individuos de la misma especie por los recursos disponibles. Los individuos que están más capacitados para conseguir los recursos tendrán más posibilidades de sobrevivir y de reproducirse. Puesto que en la reproducción, los descendientes de un individuo heredan parte de las características de su progenitor, se tiene que sucesivas generaciones serán cada vez más aptas, en el sentido de adaptarse al medio.

Además, en la naturaleza existe el fenómeno de mutación, mediante el que las características genéticas de los individuos varían aleatoriamente en ciertas ocasiones. Gracias a esto, se introducen en una población nuevas características que no estaban presentes en las generaciones anteriores.

En la realización artificial de los algoritmos genéticos los individuos que componen la población son posibles soluciones al problema planteado. Su mayor o menor adaptación al medio se corresponde con la idoneidad de la solución, y la reproducción se realiza combinando varias soluciones posibles para obtener otra que, se espera, será mejor.

Los algoritmos genéticos no garantizan que se encuentre una solución óptima a un problema concreto. Es posible que después de muchas generaciones las solución no converja, o, en caso de converger, no lo haga al valor óptimo. En lso problemas en los que existen algoritmos conocidos de probada eficacia, probablemente éstos superarán a los algoritmos genéticos tanto en rapidez como en exactitud de la solución.

Donde los algoritmos genéticos (AG) presentan su mayor ventaja es en el caso de problemas complejos para los que no existen técnicas de resolución conocidas. En estos casos se puede usar un AG para obtener una solución bastante buena en un tiempo razonable [2].

2.1 Procedimiento

Un AG comienza con una población de individuos que representan las posibles soluciones al problema. Esta población inicial puede elegirse aleatoriamente o por algún otro procedimiento.

A cada individuo de la población se le asigna una puntuación que representa su idoneidad como solución al problema. Esta puntuación se calcula por medio de la denominada *función de evaluación*. A los individuos con más alta puntuación se les permite reproducirse, cruzando su dotación genética unos con otros. Así se crea una nueva generación que comparte características con los mejores elementos de la generación anterior.

Repitiendo el proceso anterior se van consiguiendo nuevas generaciones que cada vez representan una mejor solución al problema planteado.

En el mejor de los casos, el proceso convergerá y llegará un momento en el que la diferencia entre una generación y la siguiente será ten pequeña que se pueda considerar que el algoritmo ha finalizado y se acepte la solución obtenida (en forma de individuos de la población) como válida.

2.2 Codificación

Para poder tratar computacionalmente las características de una determinada solución (el equivalente a la dotación genética de un individuo en la naturaleza) es necesario establecer alún método para codificar estas características numéricamente.

La opinión generalizada ([1], [3]) es que la mejor manera de codificar las características es usando el sistema binario.

Cada característica relevante de un individuo se almacena con un número determinado de bits, y se conoce como *gen*. La concatenación de todos los genes de un individuo es una cadena de bits llamada *cromosoma*. En ocasiones, al conjunto de parámetros necesario para describir un individuo se le denomina *genotipo*, mientras que el individuo propiamente dicho es conocido como *fenotipo*.

2.3 Función de evaluación

La función que determina la idoneidad de una solución para el problema debe ser determinada específicamente para cada problema, como cabía esperar. Es ésta un función que toma como parámetro un cromosoma y devuelve un valor escalar que mide cómo de bueno es el individuo para el problema concreto que se trata (i.e. calcula la bondad del fenotipo a partir de su genotipo).

La función de evaluación es particularmente fácil de determinar en los problemas de optimización, en los que coincide con la función que se desea optimizar. En otros tipos de problemas no es tan fácil conseguir una función de evaluación aceptable y debe estudiarse con cuidado.

2.4 Reproducción

La reproducción es el proceso de combinar individuos de una generación para crear una generación nueva. Al seleccionar los miembros de una generación que se van a reproducir, se debe dar mayor prioridad a aquellos que poseen una mayor puntuación según la función de evaluación. De esta manera, los individuos más idóneos podrán reproducirse varias veces, mientras que los menos apropiados probablemente no se preproduzcan en absoluto y desaparezcan de la población.

En el caso más simple, dos descendientes se obtienen de dos progenitores mediante cruce. En el cruce, los cromosomas de ambos progenitores son cortados en algún punto aleatorio. Posteriormente, las mitades de los cromosomas son recombinados, produciéndose dos nuevos cromosomas, cada uno con parte del cromosoma de cada uno de los antepasados.

El cruce no se produce entre todas las parejas, sino sólo en una fracción de ellas. En las parejas en las que no hay cruce, simplemente se copian los cromosomas de los progenitores en los de los descendientes.

Después de generar a todos los descendientes se producen las mutaciones. A una pequeña parte de los nuevos individuos (típicamente a menos del 1%) se le alteran aleatoriamente uno o más genes de su dotación cromosómica.

2.5 Algunas aplicaciones

Entre las aplicaciones más comunes de los algoritmos genéticos se encuentran [2]:

1. Optimización de funciones numéricas. En este campo donde se ha concentrado la mayor parte del esfuerzo investigador sobre algoritmos genéticos, disponiéndose ya de algoritmos muy eficientes para muchos problemas irresolubles por otras técnicas.
2. Procesamiento de imágenes. La identificación de personas a través de fotografías o el alineamiento preciso de dos imágenes distintas del mismo lugar son algunas de las aplicaciones desarrolladas hasta ahora.
3. Diseño de sistemas. Se han empleado algoritmos genéticos para realizar diseños óptimos de estructuras mecánicas como puentes o válvulas, y de redes neuronales.

3 Programación genética

3.1 Concepto

A partir de los algoritmos genéticos para resolver problemas numéricos se deriva la programación genética ([6], [7]), que trata de resolver el problema de crear un programa de ordenador que resuelva un problema concreto [5].

Mediante programación genética es posible hacer que un ordenador realice un determinado trabajo, sin decirle cómo debe hacerlo. Es lo que se conoce como programación automática (a veces también llamada *síntesis de programas* o *programación inductiva*).

En la programación genética la población de individuos que evolucionan está constituida por programas de ordenador. Cada programa tiene asociada un valor de su idoneidad para la resolución del problema que se trate (valor medido por la función de evaluación). Al igual que en un algoritmo genético, los programas se reproducen en cada iteración, dando lugar a una nueva generación [8].

Las dos principales diferencias entre un algoritmo genético y un problema de programación genética son la codificación y la evaluación.

3.2 Codificación

En un algoritmo genético para resolver un problema numérico, cada individuo representaba en general un vector n-dimensional que almacenaba los valores buscados para cada una de las variables del problema. Como ya se vio en 2.2, estos valores pueden codificarse en forma binaria, lo que proporciona una manera compacta y eficiente de obtener el genotipo de cada individuo.

Sin embargo, la tarea de codificar numéricamente un programa de ordenador no es trivial. Normalmente, los programas suelen codificarse con la forma de un árbol, en el que las terminaciones son las variables de entrada al programa, o ciertas constantes, y los nodos intermedios representan las operaciones a las que deben someterse estas variables y constantes. Así se obtienen programas de diferente tamaño y forma, pero con una estructura en común, que puede codificarse numéricamente.

3.3 Evaluación

En un problema de optimización numérico por algoritmo genético, la función de evaluación era la propia función a optimizar, y el proceso de evaluación consistía tan sólo en calcular el valor de la función para cada individuo.

Pero para evaluar la idoneidad de un programa que debe resolver una tarea, se deben calcular sus respuestas ante todas las entradas posibles. Esto implica someter al programa a multitud de estímulos y evaluar todas sus salidas, y después resumir todas estas salidas en un parámetro que sirva para comparar diferentes programas entre sí.

Se aprecia que el esfuerzo computacional que requiere el cálculo de la función de evaluación en programación genética es mucho mayor que el requerido en un algoritmo genético. El tiempo empleado en las otras partes del método (la generación de la población inicial, y la reproducción en cada iteración) es pequeño comparado con el necesario para la evaluación.

En cada iteración, la población puede contener cientos de miles o millones de individuos (programas), y cada uno de estos debe contrastarse contra cientos o miles de entradas diferentes. Si se tiene en cuenta que el procedimiento puede tener decenas, cientos o miles de iteraciones se aprecia la magnitud del problema.

Tradicionalmente, la evaluación de todos estos programas se hacía en un ordenador que simulaba cada programa para cada una de las entradas a que se sometía. Pero el carácter serial de los ordenadores tradicionales hacía este proceso poco eficiente.

El procedimiento puede ser mejorado con la utilización de dispositivos reconfigurables

en campo, como las FPGAs.

3.4 Utilización de dispositivos reconfigurables para la programación genética

Los arrays de puertas programables en campo (FPGAs) son dispositivos digitales masivamente paralelos. Una vez que una FPGA se configura para una tarea determinada, sus células lógicas operan en paralelo a la velocidad que marca el reloj digital. Hasta hace poco, la configuración de una FPGA era un proceso que llevaba un tiempo considerable. En tiempos más recientes, con el advenimiento de FPGAs rápidamente reconfigurables, se formó la idea de hardware evolutivo [9], [10].

Usando este concepto, es posible codificar cada individuo en el hardware de la FPGA, lo que acelera el cálculo de sus salidas. Más aún, si se codifican varios individuos a la vez en una FPGA con las suficiente capacidad, se aprovecha el paralelismo y se computan varios programas a la vez. Con un dispositivo que se pueda reconfigurar a una velocidad lo bastante rápido, la ganancia de velocidad obtenida por la codificación en hardware supera a la pérdida debida a la necesidad de reconfigurar la FPGA en cada paso.

4 Ejemplo de aplicación

4.1 La FPGA Xilinx XC6216

La serie XC6200 de Xilinx consta de FPGAs rápidamente reprogramables, que pueden ser utilizadas satisfactoriamente en la programación genética.

La codificación de los bits de configuración es abierta y está disponible para todo el mundo. Cada celda se configura con 24 bits, de significado simple y público.

El proceso de mapeado y rutado de una aplicación específica es rápido (del orden de segundos). Lo que es mejor, no es necesario descargar toda la configuración a la FPGA cada vez, puesto que los bits de configuración están directamente mapeados en la memoria del procesador, y pueden ser accedidos aleatoriamente.

No hay ninguna configuración de bits que pueda ser dañina para la FPGA, lo que posibilita la creación aleatoria de configuraciones, necesaria en la programación genética.

La FPGA Xilinx XC6216 contiene 4096 celdas idénticas, dispuestas en un array bidimensional de 64x64 [12]. Cada celda contiene varios multiplexores y un biestable, siendo capaz de implementar todas las funciones booleanas de dos variables, y la mayoría de funciones de tres. Cada celda se puede conectar siempre a sus cuatro vecinas más próximas, y en ocasiones a otras celdas más distantes.

4.2 Redes mínimas de ordenación

La aplicación que se va a describir aquí de la programación genética consiste en descubrir un algoritmo para la ordenación de una lista. El algoritmo (llamado red de ordenación) consiste en una serie finita de secuencias comparar–intercambiar entre dos elementos de la lista, y que deben ser ejecutadas en el orden establecido.

En cada paso del algoritmo se ejecuta la secuencia de comparación entre los dos elementos i y j especificados en ese paso. Si el elemento que se encuentra en la posición menor es mayor que el otro, las posiciones de ambos se intercambian.

Con una red adecuadamente diseñada se puede ordenar una lista de un tamaño dado en un número fijo de pasos. Para un número de elementos relativamente pequeño, las redes de ordenación son más eficientes que otros algoritmos, como el Quicksort [11]

El objetivo es conseguir una red de ordenación para un número de elementos dado, con el mínimo número de pasos posible. Se han conseguido redes, probadas como mínimas, para 4, 5, 6, 7 y 8 elementos (constituidas, respectivamente, por 5,9,12, 16 y 19 pasos).

La verificación de la corrección de una red crece con el número de elementos a ordenar. Cualquier red puede ser verificada probándola con las $n!$ permutaciones de n enteros distintos. Sin embargo, el *teorema cero-uno* [11] demuestra que si una red ordena de forma no decreciente n bits para todas las 2^n secuencias de n bits, entonces ordenará correctamente cualquier secuencia de n enteros en una sucesión no decreciente. El número de combinaciones con las que hay que probar una red se reduce mucho usando este teorema. Así, por ejemplo, para una red de 20 elementos, sólo hay que probar con $2^{20} = 1048576$, en lugar de con $20! \simeq 2.4329 \cdot 10^{18}$.

4.3 Implementación sobre la FPGA

El problema que se ha descrito en la sección 4.2 se puede resolver con la ayuda de una tarjeta PCI HotWorks, que contiene una FPGA Xilinx XC6216. La tarjeta permite al PC que la contiene la reprogramación de la FPGA.

Las tareas del algoritmo genético se reparten entre el procesador del PC y la FPGA. El PC es responsable de generar la población aleatoria inicial, y de configurar la FPGA con los bits necesarios para medir la idoneidad de cada individuo de la población. La FPGA debe aplicar entradas y medir salidas paar cada programa individual, calculando así su función de idoneidad.

En la figura 1 se observa el mapeado de las 4096 celdas de la FPGA en 8 zonas, cada una con una tarea diferente. Esta disposición se realiza con el objeto de aprovechar al máximo las posibilidades de paralelismo que tiene la FPGA. La disposición mostrada en la figura 1 se encuentra dos veces en la FPGA, con el objeto de obtener más paralelismo aún.

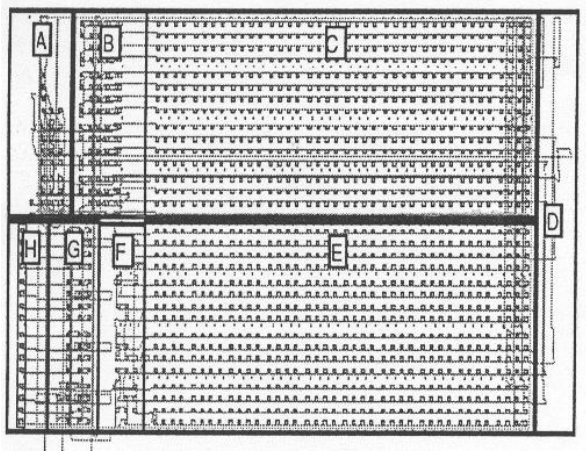

Figura 1: Disposición de los elementos sobre la FPGA

En la zona B se sitúa un contador de 16 bits, controlado por una lógica de la zona A. El contenido de B en cada momento representa un caso de los que se va a someter a examen. Los contenidos de B se van introduciendo sucesivamente en la zona C.

Dentro de las zonas C y E, cada columna representa una operación comparación–intercambio dentro de la red de ordenación. Los datos se van pasando de izquierda a derecha por la zona C, y luego de derecha a izquierda por la E. La zona D se encarga de trasladar los datos del final de C al principio de E.

Con el procesamiento en C y E se consigue un gran paralelismo, puesto que no es necesario esperar a que un dato haya pasado por todas las operaciones de comparación–intercambio para procesar el siguiente. Hay 40 columnas entre C y E, lo que implica que pueden procesarse 40 datos al mismo tiempo.

La salida final de E entra en F, donde se comprueba si la ordenación es correcta. Si lo es, se incrementa el contador G, que almacena la puntuación obtenida por la red concreta que se está analizando. El valor final de G (el valor de la función de idoneidad para la red) se guarda en el registro H, de donde puede ser leído.

Entre una red y otra, sólo hay que modificar la configuración interna de las zonas C y E. Cada celda de las columnas contiene celdas AND y OR, que deben ser conectadas adecuadamente para que la columna efectúe la operación de comparación–intercambio que se pretende.

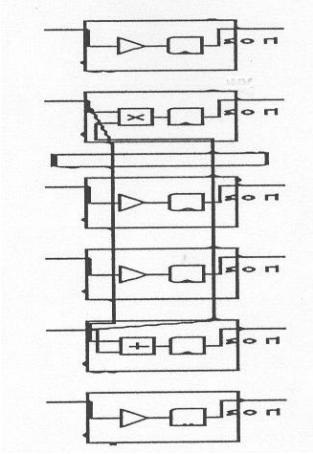

Figura 2: Comparaciçon–intercambio entre los bits 2 y 5

Como ejemplo, en la figura 2 se observa la configuración de celdas de una columna que efectúa la operación de comparación–intercambio entre sus entradas 2 y 5.

La configuración de la FPGA descrita aprovecha el paralelismo de estos dispositivos de seis maneras diferentes:

1. Las diferentes zonas A, B, C, D, E, F, G y H realizan distintas tareas al mismo tiempo.
2. Por otra parte, se dispone de dos áreas como la mostrada en la figura 1 en dos zonas distintas del chip.
3. La FPGA evalúa las entradas a la red de manera independiente del PC, lo que permite a éste ir calculando la siguiente generación al mismo tiempo.
4. Dentro de cada columna como la mostrada en la figura 2 las operaciones AND y OR se efectúan en paralelo
5. Gran cantidad de operaciones se realizan en paralelo dentro de la lógica A, el contador , la lógica F, el acumulador G y el registro H. En particular, la tarea de la zona

lógica F (determinar si la ordenación de los bits es correcta) necesita de muchos pasos sucesivos en un microprocesador en serie, pero se realiza fácil y rápidamente en una FPGA.

6. Lo más importante: los 80 pasos de las zonas C y E, y los 7 de F y G, que se realizan en paralelo, permiten que 87 casos sean analizados simultáneamente.

5 Otras aplicaciones

El campo de la programación y de los algoritmos genéticos en conjunción con la utilización de dispositivos digitales programables está en gran medida aún por desarrollar.

Existen proyectos de desarrollar chips que sean capaces de reprogramarse a sí mismo, y hacerlo mediante programación genética, con el objetivo de llegar al programa óptimo para resolver un problema.

La unión de redes neuronales y programación genética podría acercarnos al logro de conseguir dispositivos artificiales que aprendan de sí mismos y evolucionen.

Otras aplicaciones más cercanas de los algoritmos genéticos incluyen la obtención de mapeados en una FPGA o la partición óptima de circuitos en dos o más dispositivos.

Referencias

[1] J. M. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, 1975.

[2] D. Beasley, D. R. Bull, R. Martin, *An Overview of Genetic Algorithms*, University Computing, 1993, **15**(2), 58-59

[3] L. Davis, *Handbook of Genetic Algorithms*, Van Rostrand Reinhold, 1991

[4] John R. Koza, Forrest H. Bennet III, Jeffrey L. Hutchins, Stephen L. Bade, Martin A. Keane, David Andre, *Evolving Computer Programs using Rapidly Reconfigurable Field-Programmable Gate Arrays and Genetic Programming*, FPGA 98, Monterrey, USA, 1998.

[5] John R. Koza, Forrest H. Bennet III, David Andre, Martin A. Keane, *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kauffmann, San Francisco, 1999.

[6] David E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[7] John R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, 1994.

[8] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, Frank D. Francone, *Genetic Programming: An Introduction*, Morgan Kauffmann and Heidelberg, 1997.

[9] Tetsuya Higuchi (ed.), *Proc. Intl. Conf. Evolvable Systems: From Biology To Hardware (ICES 96)*, Vol. 1259, Springer-Verlag, 1997.

[10] Eduardo Sánchez, Marco Tomassini (eds.), *Towards Evolvable Hardware*. Lecture Notes in Computer Science, Vol. 1602, Springer-Verlag, 1996.

[11] Donald E. Knuth, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, 1973.

[12] Xilinx, *XC6000 Field Programmable Gate Arrays: Advanced Product Information*, Version 1.8, 1997.