Application of Genetic Programming to Verilog Hardware Design

K. Matthew Belland, Paul Booth, Jared Kirschner, Seungwhan Moon
{Kristopher.Belland,Paul.Booth,Jared.Kirschner,Seungwhan.Moon}@ students.olin.edu
623.210.0490
Department of Electrical and Computer Engineering
Franklin W. Olin College of Engineering, Needham, MA

Faculty Advisor

Ursula Wolz
Franklin W. Olin College of Engineering, Needham, MA

The design of computer architecture is a highly creative process in which optimizations are often quite non-intuitive. As other difficult and time-consuming processes have seen impressive developments through computer learning algorithms, this research investigates whether computer learning—specifically genetic algorithms—can be of assistance in developing efficient hardware logic solutions. Several genetic programming approaches are applied to basic logic problems and evaluated to assess their potential towards general computer organization and design. This work shows an alternative method by which computers could self-organize to perform tasks efficiently, even when the ideal hardware implementation is not known to the designer.

In the genetic program, several Boolean logic circuits are evolved by mimicking natural evolutionary processes. The genetic algorithm requires a representation of the hardware (an "organism") which can be changed through mutations and crossover exchanges with other organisms. Two separate representations of hardware organisms are designed and tested: layers and trees. Each organism describes a module in Verilog, a hardware description language, which simulates actual hardware components such as logic gates. A higher-level program in Python automates testing of the Verilog modules and carries out the evolution of the organisms. Each new generation of organisms is selected from the previous generation weighted according to their relative fitnesses, a numerical representation of the effectiveness of the program for the current design task. A number of options for fitness functions are explored to take into account size of the program generated, speed of convergence to an acceptable solution, and accuracy of the program's output.

The genetic program is tasked with evolving solutions to several simple Boolean functions as well as a specific cellular automation problem ("Scoot Bot") which decides how to best navigate through its environment to obtain food based on its sensor inputs. The simplest studied Boolean function would always have digital high outputs in response to its inputs. This constant function case is analyzed because the evolved solutions are easy to evaluate visually and the most efficient solutions, with only two gates, are all known. Due to the nature of genetic algorithms, the amount of time required to evolve a solution to a constant Boolean vector function varied greatly. Furthermore, there was surprising variance in the elegance of the implementation depending on the evolution parameters. In some cases, organisms with over 15 gates would be used to produce a single correct output; in others, organisms with only 2-3 gates were used. The genetic program eventually produced an organism with a two-gate logic circuit for all outputs—the smallest theoretical solution.

The "Scoot Bot" problem represents a specific example of a design problem where the optimal Boolean vector output function is unknown to the designer, a problem completely unlike emulating a particular, known Boolean function. An organism is placed on a grid of spaces with values indicating either the presence (1) or lack (0) of food. The organism receives this sensory information from the four spaces immediately adjacent to its location. Its four outputs control whether or not the organism tries to move in each of these four directions. Given a limited number of steps, the goal of the organism is to collect as much food as possible. The genetic program was able to successfully evolve ideal solutions to a variety of maps (e.g. symmetric, patterned, random). The most successful organisms shared the ability to move towards food while avoiding a shutdown. A shutdown occurs when there is no net motion in response to a particular set of inputs, causing the organism to stop moving until the end of the simulation. Several strategies for avoiding shutdowns were observed: persistent enabling of motion in a direction, persistent disabling of motion in a direction, and the utilization of a varied signal (clock input) used in some simulations. The most successful organisms adapted from these basic behaviors, responded to stimuli, and approached a maximally fit solution. These initial results of hardware evolution show promise for future applications of computer learning to hardware design and computer organization, particularly when optimization is a major concern or when a solution is not known *a priori*.

The authors would like to acknowledge the contributions made by Professor Ursula Wolz and the students in the Computer Architecture class taught at Franklin W. Olin College of Engineering during the Fall 2011 semester.

REFERENCES

[1] Banzhaf, W., Nordin, P., Keller, R. E., & Francone, F. D. (1998) *Genetic programming: An introduction*.
[2] Koza, J. R. (1992) *Genetic programming: On the programming of computers by means of natural selection*.
[3] Narayanan, S. (2005). *Hardware Implementation of Genetic Algorithm Modules for Intelligent Systems* (Master's thesis, University of Cincinnati, Cincinnati, OH). Retrieved from http://rave.ohiolink.edu/etdc/view?acc_num=ucin1122909070