CoDi-1Bit : A Simplified Cellular Automata Based Neuron Model

Felix Gers[1] **Hugo de Garis**[1] **Michael Korkin**[2]

[1]ATR, Human Information Processing Laboratories
2-2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan.
tel. +81-774-95-1079 ,fax. +81-774-95-1008 , flx@hip.atr.co.jp &
degaris@hip.atr.co.jp, http://www.hip.atr.co.jp/~flx, &
http://www.hip.atr.co.jp/~degaris
[2] Genobyte Inc. 1319 Spruce St., Suite 210, Boulder, CO, 80302, USA.
tel. + 1 303 545 6790, fax. + 1 303 545 9667, korkin@genobyte.com,
http://www.genobyte.com

Abstract. This paper presents some simplifications to our recently introduced "CoDi-model", which we use to evolve Cellular Automata based neural network modules for ATR's artificial brain project "CAM-Brain" [11]. The great advantage of CAs as a modeling medium, is their parallelism, which permits neural system simulation hardware based on CoDi to be scaled up without loss of speed. Simulation speed is crucial for systems using "evolutionary engineering" technologies, such as ATR's CAM-Brain Project, which aims to build/grow/evolve a billion neuron artificial brain. The improvements in the CoDi model simplify it sufficiently, so that it can be implemented in state of the art FPGAs (e.g. Xilinx's XC6264 chips). ATR is building an FPGA based Cellular Automata Machine "CAM-Brain Machine (CBM)" [13], which includes circuits for neural module evolution and will simulate CoDi about 500 times faster than MIT's Cellular Automata Machine CAM-8 currently used at ATR.

Keywords

Cellular Automata, Evolutionary Engineering, Evolvable Hardware, Neural Networks, Genetic Algorithms, Genetic encoding, Artificial Brains, Cellular Automata Machine (CAM-8), CAM-Brain Machine (CBM).

1 Introduction

ATR's CAM-Brain Project aims to build/grow/evolve brain-like-systems, into which human knowledge can be inserted as initial conditions. The brain-like structures are based on cellular automata (CA) which grow inside cellular automata machines (CAMs). Evolutionary engineering techniques, combined with huge computing power, may then be able to evolve such brain-like systems with a level of functionality similar to that of biological brains. The expectation is that

it will be much easier to investigate the behavior of engineering-based brains than biological brains, and that the engineering solutions will reveal insights into the working mechanisms of the evolved systems, which might then be rediscovered in biological brains.

In the CAM-Brain Project, the CAs act as the medium in which to conduct the growth, and in a second step, the neural signaling, of brain-like structures.

To apply evolutionary engineering techniques to the creation of huge neural systems, e.g. an artificial insect brain, we need a simulation tool that has to satisfy two conditions. First of all, it must be possible to simulate most of the important characteristics of biological neural networks. Secondly, the evolution times must be manageably short. The bottle-neck in evolutionary algorithms is the fitness measurement time, i.e. the time needed to simulate the systems. Hence, in order to develop what would presently be considered large-scale systems, on the order of a billion neurons, we believe that scalability by massive paralelism is essential.

With CAs as a simulating tool, the simulataneous demands of generality and scalability can be met. Artificial neurons and neural networks can be modeled at any level of detail, because CA cells can symbolize a molecule in a biological cell membrane, a whole neuron, or any structure in between. Since CAs are only locally connected, they are ideal for implementation on purely parallel hardware. CAs are both structurally flexible and inherently parallel.

However, for parallelization to be advantageous, it is essential that the rules for the underlying CA be simple enough to be stored locally. Indeed, Our CoDi CA based model [11] first grows and then executes neural structures by means of a set of rules small enough to be embodied in large-scale parallel hardware by means of reconfigurable VLSI circuits – Field Programmable Gate Arrays (FPGA's). For simulation purposes, we have executed the model on MIT's Cellular Automata Machine CAM-8 with a 16 bit look-up-table.

This paper summarizes the main ideas and features of the CoDi model, and introduces some simplifications to it that lead to the "CoDi-1Bit" model and its implementation in ATR's FPGA based CAM-Brain Machine (CBM) [13]. The CBM, which is a Xilinx XC6264 chip based hardware device which can evolve 3D CoDi based neural net modules including a complete Genetic Algorithm.

The remainder of this paper consists of the following. Section 2 describes our CoDi-1Bit cellular automaton model's growth and neural signalling phases. Section 3 presents results from some experiments in which simulations of small-scale structures were effected on the CAM8 machine. Section 4 describes some features of the CAM-Brain Machine (CBM), the FPGA machine which at time of writing is under construction. Section 5 discusses the results achieved, and previews the main orientations of our future research.

2 General Description of the CoDi CA

When designing our CA-based neural networks using the new "CoDi-1Bit" model, our objective was to implement them directly in evolvable hardware, e.g. FP-

GAs. Therefore, we tried to keep the CAs as simple as possible, by having a small number of bits to specify the state, keeping the CA rules few in number, and having few cellular neighbours. At this stage, no learning algorithm has been included into the CA, as we rely on evolutionary learning based on a Genetic Algorithm (GA).

We use a von Neumann neighbourhood [2], i.e. a cell in a 3D space looks at its 6 nearest neighbors and at its own state. It should be possible to extend our model to more than 3 dimensions, since the general approach taken is independent of the dimensionality of the CA-space.

We use two different CAs for the growth phase and the signaling phase in our neural networks. The cell interactions are similar in both. The main difference between the two phases, and hence the two CAs, is that the type of a cell can only change during the growth phase, e.g. a blank cell can change to an axon or a dendrite cell. (See later sections for details.) Cell types do not change during the signaling phase, although cells do change their "activity values" and exchange neural signals. (See later sections for details).

Since cell-to-cell interactions are similar in the growth and signalling phases, they can be incorporated into the same CA with a dynamic "phase switch". This might be useful when a future learning algorithm needs to grow new connections, depending on the activity in already existing axonal or dendritic connections. In the following three subsections (2.1- 2.3) we describe how the CoDi CA can be implemented as a partitioned CA, the CoDi cell-to-cell interaction model itself and the genetic coding, i.e. the initialization, of the CoDi CA.

2.1 Implementation of CoDi as a Partitioned CA

The states of our CAs have two parts, which are treated in different ways. The first part of the cell-state contains the cell's type and activity level and the second part serves as an interface to the cell's neighbourhood by containing the input signals from the neighbors. Characteristic of our CA is that only part of the state of a cell is passed to its neighbours, namely the signal and then only to those neighbours specified in the fixed part of the cell state. This CA is called "partitioned" [11], because the state is partitioned into two parts, the first being fixed and the second is variable for each cell (see figure 1).

The advantage of this partitioning-technique is that the amount of information that defines the new state of a CA cell is kept to a minimum, due to its avoidance of redundant information exchange.

For a CA implementation on a look-up-table (LUT) machine such as MIT's CAM-8 or in FPGAs, it is crucial to minimize the amount of cell-to-cell communication, to keep the number of states in the CA manageable.

This is a major advantage for the implementation of CAs in parallel hardware. Large LUTs or complex programs cannot be stored locally, and hence are non-parallel. Using the partitioning-technique, improves the practicability of a neural network simulated directly in parallel hardware.

The next few subsections detail the new CoDi model and its implementation as a partitioned-CA.

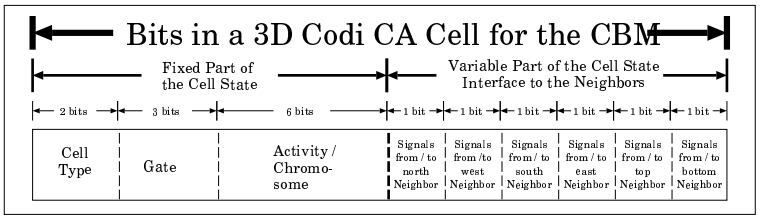

Fig. 1. State representation in the CoDi-1Bit model. During the growth phase 6 of the bits are used to store the chromosome's growth instructions. The same 6 bits are later used to store the activity of a neuron cell during the signaling phase.

2.2 CoDi Cell Interaction

The CoDi model is named after the essential features of its axonal and dendritic cell behavior, i.e. to "collect" or to "distribute" the cellular values from/to the CA cells' neighbors. The CoDi model works with four basic types of CA cells, neuron body, axon, dendrite and blank.

- Blank cells represent empty space. They do not participate in any cell interaction during the signaling of the neural network.
- Neuron bodies consist of one CA cell. The neuron body cells collect neural signals from the surrounding dendritic cells and apply an internally defined function to the collected data. In the CoDi model the neurons sum the incoming signal values and fire after a threshold is reached. This behavior of the neuron bodies can be modified easily to suit a given problem. The output of the neuron bodies is passed on to its surrounding axon cells.
- Axonal cells distribute data originating from the neuron body.
- Dendritic cells collect data and eventually pass it to the neuron body.

These types of cell-to-cell interaction cover all kinds of cell encounters. Axonal and dendritic cells do not need to know anything about the cell type of their neighbors. They behave in the same way towards any type of neighbor. Therefore it is not necessary to distinguish between the different cases of cell-to-cell interactions, e.g. axon-neuron, axon-dendrite, dendrite-dendrite etc, so that designing separate rules for each case is unnecessary. The two basic interactions cover every case, and they can be expressed simply, using a small number of rules.

To make axonal and dendritic signal trails (which can be dense in the CA-space) without blockers or sheath cells as in earlier models [4], the four basic cell types need further specification. To the 2 bits which specify the basic cell type, we added 3 more bits to set the "gate" of a 3D CA cell. (See figure 1.) A neuron cell uses this gate to store its orientation, i.e. the direction in which the axon is pointing.

In an axon cell, the gate points to the neighbor from which the neural signals are received. An axon cell accepts input only from this neighbor, but makes its own output available to all its neighbors. In this way axon cells distribute information. The source of information is always a neuron cell. Dendritic cells collect information by accepting information from any neighbor. They give their output, (e.g. a Boolean OR operation on the binary inputs) only to the neighbor specified by their own gate. In this way, dendritic cells collect and "sum" neural signals, until the final sum of collected neural signals reaches the neuron cell.

We will see that each axonal and dendritic cell "belongs" to exactly one neuron cell. This configuration of the CA-space is guaranteed by the preceding growth phase (section 2.3). Figure 2 shows the signaling phase with the CoDi-model. Each neuron is given two dendritic trees and two axonal trees. One of the axonal trees distributes inhibitory signals of the neuron, the other distributes excitatory signals. A neuron has two dendritic trees to maximize the amount of information that can be passed to the neuron body in each time step.

We use binary neural signals which can fill a trail contiguously. The direction of motion is given by the underlying signal trails.

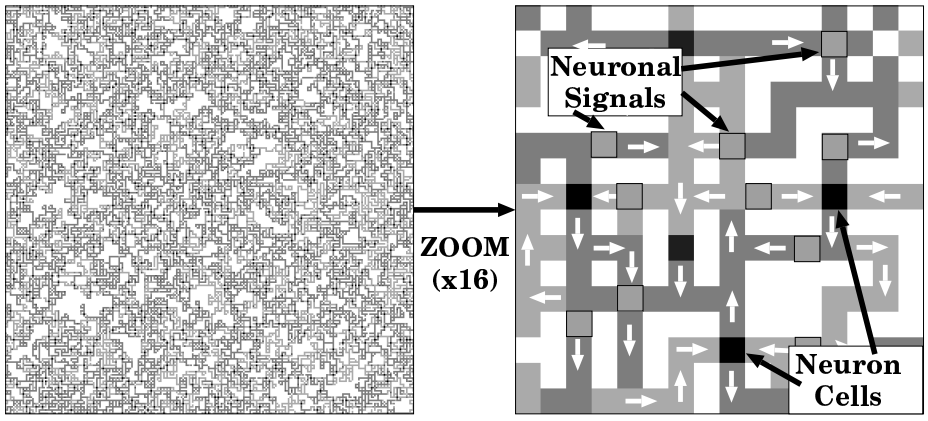

Fig. 2. (Left) A CA-space with $256 \cdot 256$ cells in the signaling phase with the CoDi-model. (Right) A zoom (x16) with five neuron cells (black) with two dendrites and two axons each. The arrows inside the axonal (dark grey) signal trails and dendritic (light grey) signal trails indicate the direction of information flow during the signaling phase. In the growth phase the signals in the dendritic trails travel the other way, because they are emitted by the neuron. To improve the visualization of neural trails, the underlying chromosome is gridded with a checker-board pattern of "grow-straight" instructions. This causes the neural trails to grow on a 2-cell grid.

Synapses The CoDi model does not use explicit synapses, because dendrite cells that are in contact with an axonal trail (i.e. have an axon cell as neighbor) collect the neural signals directly from the axonal trail. This results from the behavior of axon cells, which distribute to every neighbor, and from the behavior of the dendrite cells, which collect from any neighbor.

However axon-axon and dendrite-dendrite interactions of cells that do not belong to the same neuron are not possible. This is because axon cells take their information only from the neighboring axon cell that is closer to the "source" neuron on their axonal trail. Dendritic cells behave in the opposite way, by passing signals to the dendritic neighbor that is closer to the neuron or to the neuron cell itself. This configuration of the CA-space results from the neural growth phase.

These two simple behaviors of axon and dendrite cells (i.e. collect and distribute), which are exact opposites, allow axonal and dendritic trails to be dense in the CA-space, without interacting in an unintended way. The CoDi-model remains functional even if every cell in the CA-space is used.

The strength of a neuron-neuron connection (a synapse) is represented by the number of their neighbouring axon and dendrite cells. The exact structure of the trails and the position of the axon-dendrite neighbor pairs determine the time delay and strength (weight) of a neuron-neuron connection. This principle implies that a single neuron-neuron connection can consist of of several synapses with different time delays with independent weights.

2.3 Genetic Encoding, Initialization and Growth of the CoDi CA

Biological Systems have to self-assemble. The genetic encoding of biological brains is indirect, sophisticated and incomplete. There is not enough genetic information in vertebrates to encode the detailed connectivity of their brain circuits.

However, these characteristics of biological brain growth need not apply to electronic brains. Electronic circuits do not have to grow. They can be copied into hardware as initial settings. The genetic encoding can be direct. For example, a neural circuit can be stored as a spatial pattern in bitmap format. The genetic information can be complete, so that a given chromosome always produces exactly the same neural circuit and vice versa.

For the CoDi model, we have kept the biological feature of growing a system. We chose a genetic encoding that is indirect, but very different from biological genetic encoding. We wanted our encoding to be one to one, so that a given chromosome leads to exactly one neural network. In this way we ensure that a chromosome has a unique fitness value and hence do not have to spend time calculating an average fitness for each chromosome.

The advantage of a grown system is that the structural integrity of the basic components (in our case the neurons) is preserved in the presence of mutations (or alternations in the chromosome). All non-blank cells in the CA-space can be assigned to exactly one neuron, (the neuron from which they were grown). This is the case for any chromosome. Though the connectivity of the neural network

may be unsuitable for a given problem after the growth according to a arbitrary chromosome, a consistent structure of the neural network is always guaranteed. The importance of this feature becomes apparent when we compare a) the effect of genetic operators on a neural system that grows and b) neural systems that are mapped directly to their chromosomes. In the first case a mutation leads to a small, local change in the direction of a signal trail. In the case of direct mapping, a mutation and the resulting local change of a cell, which may belong to a signal trail of a complex axonal tree, can lead to disconnection of a large amount of circuitry. This circuitry would then be disconnected from its source neuron, and loose the integration into the basic structure of the neural network model. The situation gets worse, if we look at the way new neural connections come into being. The probability that a new and useful signal trail evolves if all cells in the CA-space are mutated independently (as is the case for direct mapping), are very small. However a mutation in a grown neural network (for example from "turn right" to "split right") leads to the growth of a new neural signal trail that is consistently integrated into the structure of the source neuron and hence the whole network. This feature of a grown system reduces the search space for possible CA-patterns to the patterns that represent consistent neural structures. The selection of an indirect genetic encoding is a logical consequence of the decision to grow our networks.

The CoDi model uses a distributing chromosome to encode its structure [11]. The chromosome is initially distributed throughout the CA-space, so that every cell in the CA-space contains one instruction of the chromosome, i.e. one growth instruction, so that the chromosome belongs to the network as a whole. This technique has several advantages compared to previous approaches [4].

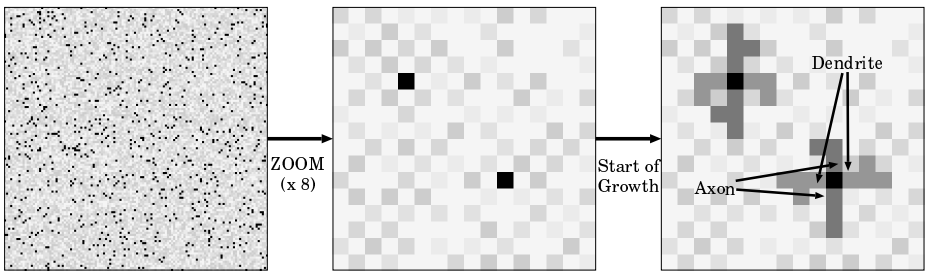

Fig. 3. (Left) An initial pattern of $128 \cdot 128$ 2D CA cells with a neuron-density of 5%. The (black) neuron cells are randomly positioned. (Middle) A zoom (x8) with two neuron cells on a random (2-cell gridded) chromosome. (Right) The beginning of the growth phase, after three CA-steps.

The neuron bodies can be positioned arbitrarily in the CA-space. (See figure 3.) Growth signals are passed to the direct neighbors of the neuron cell according to its chromosome information. The blank neighbors, which receive a neural

growth signal, turn into either an axon cell or a dendrite cell. (See figure 3 (right).) In the earlier CoDi model [11] the growth signals (and signals in general were 2 bits wide), so that they included information containing the cell type of the cell that was to be grown from the signal. To reduce the Codi model to one bit signals we had to find a different way to pass this information. The solution is to work with an oscillating CA, i.e. the growth signals are time multiplexed. In a given time-step only axons pass their growth signals and all blank cells that receive signals interpret them as "grow an axon" signals, whereas in the next time-step dendrites grow. Different ratios of axon to dendrite time-steps are possible, e.g. 3 axonal time-steps on one dendritic time-step.

The cells just grown, then receive the next growth signal from the neuron body cell, which sends them continuously. To decide in which directions axonal or dendritic trails should grow, the grown cells consult their chromosome information which encodes the growth instructions. These growth instructions can have an absolute or a relative directional encoding. An absolute encoding masks the 6 neighbors (e.g. directions) of a 3D cell with 6 bits. The cell contiguous to the face (direction) whose bit is set, will itself become an axon cell or dendrite cell at the next clock tick. For a relative encoding, 5 bits are sufficient, as the direction from which the initial growth signal comes is known to a cell (be stored in its gate field). The growth information can be interpreted relative to this direction, so one bit less information is needed. The absolute encoding has the advantage of being easier to formulate in logic equations and we use it in the FPGA design of the CBM (see section 4). A relative encoding saves one bit in the cell state, which can be crucial for LUT machines. To avoid a 17 bit LUT we used the relative encoding on the CAM-8 (which offers a 16 bit LUT). We will use the relative encoding to illustrate the growth phase with a distributed chromosome.

5 bits of chromosome information in each cell offer $2^5 = 32$ different growth instructions, which are : ("block growth", 5 simple turns, 10 simple splits, 10 3-direction splits, 5 4-direction splits and "grow in all directions"), see figure 4). Using its chromosome instructions and knowing from which direction the growth signals come, a cell can determine the neighbors to which it transmits the incoming growth signals. (See RHS of figure 4.) If, for example, a growth signal comes from the left neighbor to a cell whose chromosome instruction is "split right", it will pass the growth signal to its right and bottom neighbors. After a cell is grown, it accepts growth signals only from the direction from which it received its first signal. This "reception direction" information is stored in the "gate" position of each cell's state.

Figure 5 shows two cut-outs of neural networks at the end of a growth phase. The growth phase stops when growth is saturated, i.e. when no more trails (as specified by the instructions of the network's chromosome) can be formed.

In most of our figures, we have used gridded neural structures, in order to render the axonal and dendritic trails more visible. To force neuron trails to grow on a 2-cell grid, the underlying chromosome is initialized in a "checker-board" pattern. The "black" squares of the checker-board are filled with "grow-straight"

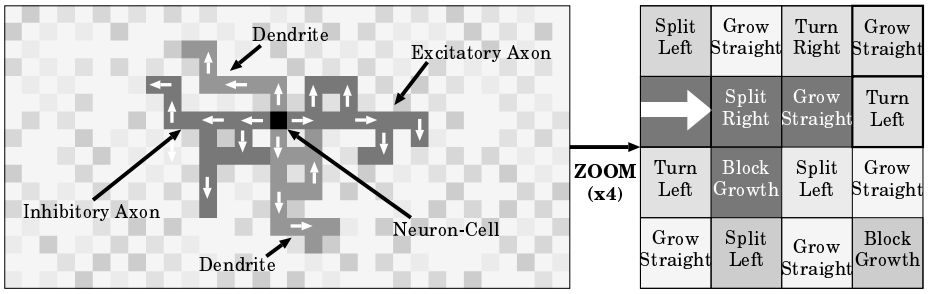

Fig. 4. (Left) A neuron in the CoDi-model with two dendrites and two axons. The arrows inside the axonal and dendritic signal trails indicate the direction of information flow during the growth phase. The neuron grows on a random (2-cell gridded) chromosome in the CA-space. (Right) A signal trail grows according to the underlying chromosome instructions. The trail continues to grow over the two striped cells in the next two time steps. The branch of the trail that leads downward is blocked by the blocker instruction in the chromosome.

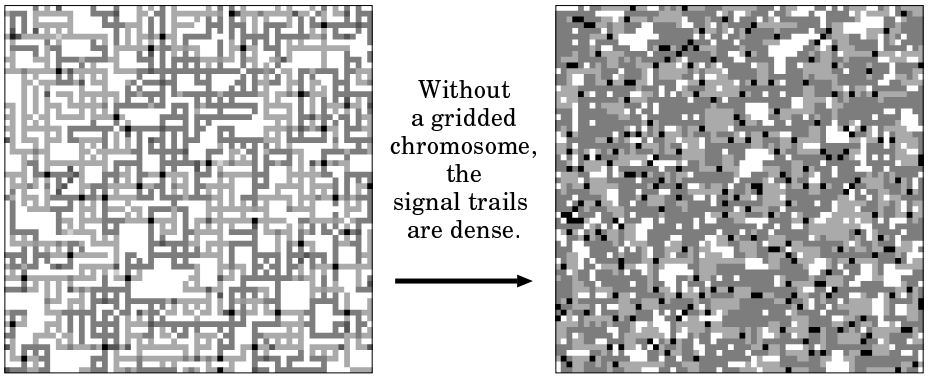

Fig. 5. Two cut-outs of neural networks at the end of the growth phase. (Left) A network grown from a gridded chromosome. (Right) A network grown without gridded chromosome instructions.

instructions and the white squares are filled with arbitrary growth instructions. Since gridding is only a visualization trick, an unconstrained evolutionary algorithm can grow trails which are dense (the RHS of figure 5 shows an example of an ungridded neural network).

Beside the "turns", "splits" etc. instructions, there are "block-growth" instructions in the chromosome. They cause a cell in an axonal or dendritic trail to block the passage of the incoming growth signals to its neighbors, so the

branch of the trail stops growing. With a high density of blocker instructions in a chromosome, an area in the CA-space can be transformed into a more or less impenetrable "wall". Figure 6 shows two "walls" with different densities of

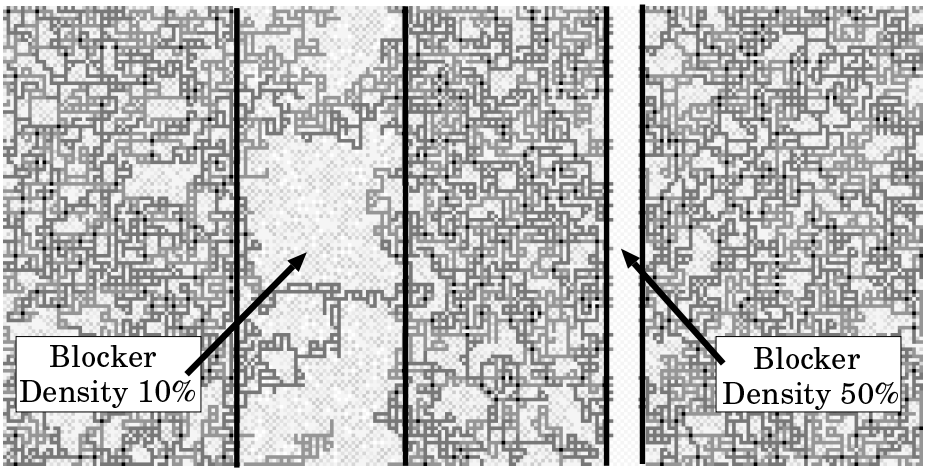

Fig. 6. The use of "walls" to modularize neural circuits in the new CAM-Brain model. The density of blocker instructions on the chromosome determines the probability of trail-growth through a wall and hence the penetrability of the wall.

blocker instructions in the two regions. A combination of walls can partition a neural network into more or less independent modules.

The grown circuitry and the chromosome are connected locally, so that a local learning algorithm can make local changes in the phenotype and the genotype of the network. This enables the use of Lamarckian evolution (with local learning algorithms) in addition to Darwinian evolution.

The distributed chromosome technique of the new CAM-Brain model makes maximum use of the available CA-space and enables the growth of any type of network connectivity. The local connection of the grown circuitry to its chromosome, allows local learning to be combined with the evolution of grown neural networks.

2.4 CoDi on the CAM-Brain Machine (CBM)

Why is the simplicity of our CA-model important? Our long term aim is to built CAs to model brain-like structures. Biological brains, the highly optimized results of millions of years of evolution, are obvious sources of inspiration for our work.

Even the tiny brains of insects consist of about a million neurons with complex connectivity. In order to model a 3D neuron with CAs in a way that preserves the essential functionality of real neurons, we estimate that we need at least a cube of 10^3 CA cells. So the minimum size of CA-space to model an insect brain is 10^9 cells. To update a billion CA cells at acceptable speeds will require the use of parallel hardware. The ideal case for CA updating is to give each CA cell its own processor, so that the whole CA-space is updated, in parallel, in one clock cycle.

To build parallel hardware with 10^9 independent CA processors (for a small mammalian brain it would be 10^{12} CA processors), the components must be very simple in their internal logic, with very little memory. Large look-up-tables (LUTs), as are common in traditional CA implementations, will be impossible. Hence, for a CA to be executable on parallel hardware, the rules must be few enough to be expressible within the limited number of logic gates per CA processor inside the parallel hardware. For this reason we tried to keep our new CAM-brain model as simple as possible.

With the CoDi-1Bit model, illustrated in this paper, it is possible to implement one CA cell with about 150 Xilinx XC6264 "functional units". This inspired us to build our CAM-Brain Machine (CBM)[13].

3 Evolvability Simulation Results

This section presents some results which were undertaken to show that the CoDi-1Bit model is actually evolvable. It also briefly discusses representation issues.

Three experiments were performed to show that it would be worthwhile to invest in the financial cost to build a fully hardware implemented model of the CoDi-1Bit design in the CAM-Brain Machine (CBM). Note that the emphasis here is NOT on the intrinsic interest of the evolved functionality, but rather the evolvability of the model.

3.1 The Comparator

The first software simulation experiment (on a Sun Sparc 10 workstation) aimed to evolve a binary comparator, i.e. two 3 bit numbers were input at the face of a cube of 16*16*16 CA cells. At the opposite face was an output whose binary value should be 1 if the input signal A was greater or equal to the input signal B, otherwise 0. The signal phase was run over 50 clockcycles, with only the output from the final 20 clocks being used in the fitness definition (to allow time for the signal to circulate through the circuit). If the desired output actually occured, the fitness value incremented a point. The total points over all 64 cases (8 cases of A, 8 cases of B) were divided by 64*20. The population size was 10. The chromosome length was about 20K bits. The fitness rose to about 90 percent after about a day's run, and then very slowly increased, but our human patience ran out. We were not prepared to run the experiment for days. We came to the conclusion *that to truly test the evolvability of the CBM, we would need a CBM* .

3.2 The Timer

The next two experiments were "single case" types, so that the fitness measurement time would be within human patience limits. The second experiment was to evolve a timer, i.e. the output signal should be 0 for the first 40 clocks (of 100 total), the next 30 clocks should have a 1 putput, and the remaining 30 clocks should be 0. The inputs were the same 6 inputs as before, but all set at 1. The same output point was used. The 16*16*16 cube was wrapped around on the side edges, but not at the top and bottom. The fitness definition was chosen so that if a desired 1 occured, the fitness value incremented 5 points, otherwise, incremented zero points. If a desired 0 occured, the fitness incremented by 1 point. The total over the 100 clocks was divided by ((40+30)*1)+(30*5) = 220. After about 100 generations, the desired output evolved perfectly, i.e. fitness was 100 percent. This first success showed that the CoDi-1Bit model could do at least this much.

3.3 The Full Sinus Wave

The third and final experiment undertaken was to evolve a sinusoid output. The inputs were the same as in the second experiment. There was a 2 bit output with the following representation. If the first bit B1 was a 1 and the second bit B2 was a 0, then a fictitious counter was incremented by 1. If B1=0 and B2=1, then the counter was decremented by 1. Other cases left the counter unchanged. An arbitrary sinusoid amplitude and wavelength target shape was chosen, that the counter is supposed to follow as closely as possible, with the counter value being the vertical "y" axis, and the clock count being the horizontal "x" axis. The circuit should evolve so that the outputs B1 and B2 change over time, such that the counter gives the appropriate "y" values over time.

The fitness value was defined to be the inverse of the sum of the squares of the differences between the desired y values and the actual counter values. An incremental or stepwise evolutionary approach was taken that de Garis developed in 1990 [de Garis 1990], namely that the evolution used several intermediate fitness definitions, where the resulting population of GA chromosomes evolved with fitness definition FD1, became the starting generation of chromosomes with fitness definition FD2, etc. Often this approach gives better results than using only one fitness definition. In the case of the sinusoid, at first a quarter wavelength was evolved (where the target sinusoid had an amplitude of 20 and wavelength of 200 and the whole curve shifted up by 30. The fitness measurement started after 20 clocks, at which point, the counter was set initially at the value 30. After about 100 generations, the evolved output followed the desired target within a few percent. The resulting population became the starting population for a second evolution phase where the aim was to evolve a half sinusoid. Again, after a day, and several hundred generations, the actual curve followed the target curve within a few percent. Two further steps were used, to evolve a three-quater wave and a full wave. The three-quarter wave evolved as well as the previous two, but the full wave did not quite get the actual curve to return to the "base line"

(set at 30). It looked as though the limits of the evolvability of the 4K CA cell module had been reached. An alternative fitness definition which weighted the later clocks was attempted, but had no effect. Fig.7 shows the result of the full sinusoid, both target and actual output.

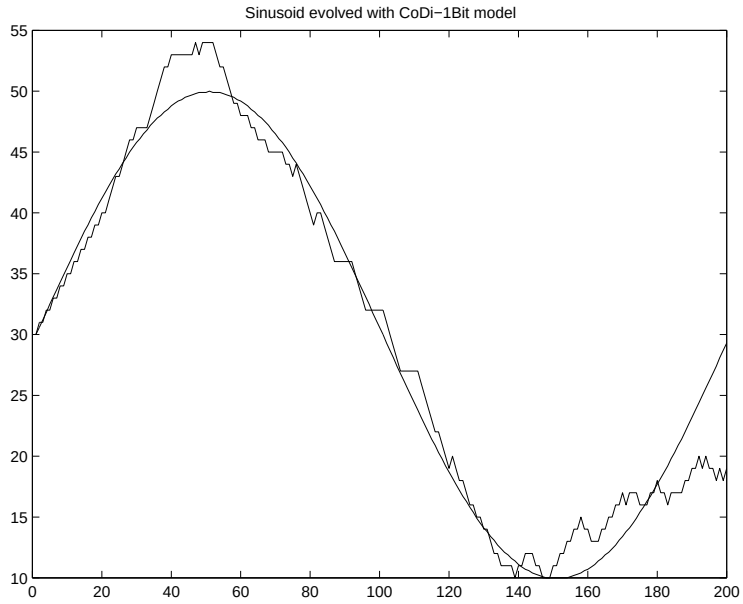

Fig. 7. Sinus wave generator evolved by CoDi model simulation on CAM-8

These few results were thought to be enough to show that the CoDi-1Bit model is evolvable enough to proceed with the construction of the CAM-Brain Machine.

3.4 Representation Issues

Finally, something needs to be said about the representation of the input and output signals. Of course, one is free to interpret these binary inputs and outputs as one chooses, e.g. as binary numbers, as biological "spikes" in a convolution/deconvolution model, as incrementers/decrementers, etc. The issue of representation is an important one which needs to be given a lot more thought in future CBM work.

4 CBM Architecture

The CAM-Brain Machine (CBM) is a special purpose hardware device based on Xilinx's programmable hardware (FPGA) chips (XC6264). The CBM is not just a faster version of the CAM-8 machine. It is based on the CoDi-1Bit neural model and introduces a number of highly specialized features. In this respect, CBM is not intended for general purpose cellular automata applications, such as hydrodynamic modeling, lattice gas simulation, or thermal stress analysis. The CBM consists of the following five major blocks :

- Cellular Automata Module
- Genotype/Phenotype Memory
- Fitness Evaluation Unit
- Genetic Algorithm Unit
- Module Interconnection Memory

Each of these blocks are discussed in detail below.

4.1 Cellular Automata Module

The cellular automata module is the hardware core of the CBM. It is intended to accelerate the speed of brain evolution through a highly parallel execution of cellular state updates. The CA module consists of an array of identical hardware logic circuits or cells arranged in a 3D structure of two modules of 16*16*16 (4096 cells) or one module of 16*16*32 (8192 cells). Cells forming the top layer of the module are recurrently connected with the cells in the bottom layer. A similar recurrent connection is made between the cells on the north and south, and the east and west vertical surfaces. Thus a fully recurrent toroidal cube is formed. This feature allows a much higher axonal and dendritic growth capacity by effectively doubling each of the three dimensions of the block.

4.2 Universal Cell

Each hardware logic cell contains circuitry to implement several different CBM functions, neuron, axon, dendrite and blank cell. Depending upon a particular chromosome of the neural module, each of these "universal" cells will perform one of the 4 possible functions in a given module during the signaling phase and may change its type from blank to an axon or a dendrite type during the growth phase. A blank cell can also be initialized to function as a neuron before the beginning of the growth phase.

4.3 CA Module Implementation in Hardware

The CA module is implemented with new Xilinx FPGA devices XC6264. These devices are fully and partially reconfigurable, feature a new co-processor architecture with data and address bus access in addition to user inputs and outputs,

and allow the reading and writing of any of the internal flip-flops through the data bus. An XC6264 chip contains 16384 logic function cells, each cell featuring a flip-flop and some Boolean logic capacity, capable of toggling at a 220 MHz rate. Logic cells are interconnected with neighbors at several hierarchical levels, providing identical propagation delay for any length of connection. This feature is very well suited for a 3D CA space configuration. Additionally, clock routing is optimized for equal propagation time, and power distribution is implemented in a redundant manner. To implement the CA module, a 3D block of identical logic cells is configured inside each XC6264 device, with CoDi specified 1-bit signal buses interconnecting the cells. Each XC6264 device, available in a QFP304 (quad flat pack) package, has up to 180 input/output pins. The number of available pins limits to 180 the maximum number of external connections to be made when interconnecting several XC6264 chips to form the 4096-cell space. Another limitation is imposed by the amount of logic needed to implement all the necessary functions in the CoDi cells. With these limitations in mind, the best arrangement for each XC6264 part is a block of 4*4*8 cells (total of 128), which results in 160 externally connected cell surfaces, 4 surfaces of 4*8 and 2 surfaces of 4*4 cells. To assemble a 4096-cell module, 32 XC6264 chips are needed, configured as 2*4*4 chips on 4 boards (dual surface mounted). The amount of hardware logic available for each CA cell is 16384/128 = 128 Xilinx function blocks. This amount is sufficient for implementing all necessary CoDi model functions for growth and signaling, cell interconnection, and other requirements. All cells exchange signals and perform their respective functions synchronously at the rising edge of the master clock running at 30 MHz. The clock rate of 30 MHz (33 ns cycle) permits eleven layers of logic between reregistration to be used for CA cell circuitry design. (Each logic layer's propagation delay is 2.5-3 ns). During every clock cycle (33 ns), all 4096 cells are updated. This yields a speed increase of 614.4 times over the CAM-8 speed for the 1-bit 3D CoDi cells update rate of 200 Mcells/s. This number demonstrates "raw" speed increase, which is somewhat reduced by the module reconfiguration time. (See below).

4.4 Power Consumption and Heat Management

Dynamic power consumption of the module hardware is proportional to the clock rate, the number of flip-flops toggling simultaneously, and the capacitive loading of all the connections inside and outside the FPGAs. Static power consumption is expected to be negligible. Calculations yield a dynamic power consumption of the order of 4 watts for each XC6264 device. With a total of 32 FPGAs, the power consumption of the module is 128 watts (25.6 amperes). It is expected that the rest of the CBM circuitry will additionally require 100 watts. Given the estimated power dissipation, thermal management of the device can be accomplished using standard forced airflow techniques.

4.5 Module Inputs and Outputs

Although the module architecture is fully recurrent, some of the recurrent connections between top and bottom, east and west, etc., are reserved for external inputs and outputs. There are 8 cells on each surface of the module reserved to be used as external inputs to the module (a total of 48) and another 8 cells on each surface to be used as external outputs (a total of 48). Input and output locations are evenly spread over each surface of the module cube. In order to establish connections to external signals, an input cell must be evolved as either a dendrite or a neuron pointed inside. An output cell must be evolved as an axon or a neuron, pointed outside.

4.6 Genotype and Phenotype Memory

There are two modes of CBM operation, namely evolution mode and run mode. In the evolution mode, memory space is used to store the chromosome bitstrings of the evolving population of modules (module genotype). For a module of 4096 cells there are 20480 bits of memory needed. For each module the genotype memory also stores information concerning a maximum of 128 neurons locations and orientations inside the module. This includes, X, Y, Z coordinates (12 bits), gating code (3 bits), input functions (excitatory/inhibitory) (5 bits), giving a total of 20 bits per neuron. Thus, the total chromosome memory requirement for one module is 20480+20*128 = 2880 bytes.

In run mode, memory is used as phenotype memory for the evolved modules. The phenotype data describes grown axonal and dendritic trees and their respective neurons for each module. For phenotype storage there are 6 bits required per cell, i.e. for gating (3 bits), cell type (2 bits), and signal value (1 bit). Neuron cells additionally require 4 bits for the accumulator value stored. Phenotype data is loaded into the CA module to configure it according to the evolved function. Every 1024 cell board on a 4 board module will have 8 Mbytes of its own genotype/phenotype memory, with four 32-bit data buses for parallel access. 32 Mbytes of memory can store over 10 thousand modules at a time. This amount is sufficient to evolve 10 thousand modules at high speed, or to run a simulated brain with one million neurons. A large memory will be based in the main memory of the host computer (Pentium-Pro) connected to the CBM through a PCI bus, capable of transferring data at 132 Mbytes/s. Genotype/phenotype memory is connected to the hardware CA module and is used for rapid reconfiguration of the neural module by loading new chromosome data (or phenotype data) into the hardware registers of each cell through the XC6264 data bus access. Thus the fast hardware for the CA module is time- multiplexed between multiple neural modules in a large brain.

4.7 Hardware Reconfiguration Speed

The module reconfiguration speed is limited by the speed of the XC6264 data bus access and the width of the memory data bus. Each XC6264 device is capable

of accessing 32 internal flip-flops at a 25 MHz rate, which results in 150 million 5-bit cell reconfigurations per second. Each of the 2-chip sections on each of the four boards has its own memory bus. During parallel access to 16 FPGA's a rate of 2.4 billion cell reconfigurations per second is achieved. Thus, a 4096-cell FPGA hardware module can be reconfigured in 1.7 microseconds. There is a two-stage pipelined configuration register in every cell which permits the loading of the configuration data of a new module (phenotype and genotype) while running a previous module. At the end of the signaling or growth phase, the new configuration is instantly available. This feature guarantees continuous running of the CA module at full speed with practically no interruptions for reloading.

4.8 Overall CBM Speed Estimate

The goal of the CBM architecture is to exploit the full theoretical speed of parallel cell update, equal to 122.88 billion cells per second, or 614.4 times faster than the CAM-8. In order to achieve this, each module must be run for at least the time needed to load a module's configuration data (1.7 microseconds) which is equivalent to 51 update cycles. Also, a dual buffered set of signal FIFOs (see sections 3.3 and 3.5) will be provided to avoid interruptions of input/output signal data reloading. However, with some unavoidable overhead, the overall speed is expected to be around 110 billion cell updates per second.

4.9 Fitness Evaluation Unit

When a useful module is being evolved, each instance of a module must be evaluated in terms of it's fitness for a targeted task. During the signaling phase, each module generates a sequence (an array) of 48 output signals, or vectors, which is compared with a target array in order to guide the evolutionary process. This comparison gives a measure of performance, or fitness, of the module. Fitness evaluation is supported by a hardware unit which consists of an input vector array stack, a target vector array stack, and a fitness comparator. Both stacks are 6-byte wide SRAMs (FIFOs) storing up to 2048 input and target vectors each to support the signaling phase of up to 2048 cycles. During each clock cycle an input vector is read from its stack and fed into the module's inputs. At the same time, a target vector is read from its stack to be compared with the current module output vector by the fitness evaluation unit. The fitness comparator computes a Hemming distance between each output vector and a corresponding target vector, and accumulates the result for the whole duration of the signaling phase. At the end of the signaling phase, a final measure of the module's fitness is instantly available. Multiple target and input arrays are stored in the host computer memory.

4.10 Genetic Algorithm Unit

To evolve a module, a population of 100 modules is evaluated by computing every module's fitness measure, as described above. The ten best modules are

then selected for further reproduction. A hardware support will be provided to store and update a "current best ten" list. This list is an array of the current ten best module numbers. Each number is an address of the module chromosome in the chromosome memory. After each generation of modules, the ten best are mated and mutated to produce 100 offspring modules to become the next generation. Mating and mutation is performed by the host computer software on the chromosome memory. Because this process can be performed in parallel with the module evaluation, and only takes place once in a generation, it is expected that there will be no significant slowdown in the evolutionary process.

4.11 Module Interconnection Memory

In order to support the run mode of operation, which requires a large number of evolved modules to function as an artificial brain, a module interconnection memory is provided. It consists of an output vector array stack, similar to the input array and target stack, and a larger memory for inter modular pathway storage. When each module of a large brain is configured in the CA hardware core (by loading its phenotype), an input stack is loaded with an array of input vectors. These vectors are previously stored output vectors recorded during the signaling phase of other modules, connected to this module. A large module interconnection memory will store the connection map and the current state of signals "traveling" between modules. There will be software and hardware support provided for combining output arrays from up to 8 modules, to be used as inputs for one module. In addition, an externally connected set of inputs and outputs is provided in hardware, which will allow the reception of signals from external sensors and the sending of signals to external effectors for robotic control. When running a simple million-neuron brain which contains ten thousand 100-neuron modules, the CBM is capable of updating each module for 100 cycles at the rate of about 27 times per second, allowing real-time control of robotic devices.

5 Conclusions

This paper presented a simplification of the CoDi model (called "CoDi-1Bit") for the evolution of neural structures, based on cellular automata. We reduced the signals to one bit with an oscillating cellular automata. This simplification made it possible to implement CoDi-1Bit with about 150 "functional units" of a Xilinx XC6264 chip per CA cell. This led to the development of the CAM-Brain Machine (CBM), whose architecture is also described. Hence, with the new CoDi model, it will be possible to evolve neural structures directly in hardware.

References

1. Toffoli, T. & Margolous, N. *'Cellular Automata Machines'*, MIT Press, Cambridge, MA , 1987.
2. von Neumann, J. *'Theory of Self-Reproducing Automata'*, ed. Burks A.W.University of Illinois Press, Urbana , 1966
3. Codd, E.F. *'Cellular Automata'*, Academic Press, NY 1968.
4. de Garis, H. *'CAM-BRAIN: The Evolutionary Engineering of a Billion Neuron Artificial Brain by 2001 which Grows/Evolves at Electronic Speed Inside a Cellular Automata Machine (CAM)'*, in 'Towards Evolvable Hardware', Springer, Berlin, Heidelberg, NY , 1996.
5. Lloyd, S. *'A Potentially Realizable Quantum Computer'*, Science **261**, 1569-1571 1993.
6. Koza, J.R. *'Genetic Programming: On the Programming of Computers by the Means of Natural Selection'*, Cambrige, MA, MIT Press , 1992
7. Koza, J.R. & Bennet, F.H. & Andre, D. & Keane, M.M. *'Toward Evolution of Electronic Animals Using Genetic Programming'*, ALife V Conference Proceedings, MIT Press , 1996.
8. Sipper, M. *'Co-evolving Non-Uniform Cellular Automata to Perform Computations'*, Physica D **92**, 193-208 1996.
9. Carter, F. L. *'Molecular Electronic Devices'*, North-Holland, Amsterdam, NY, Oxford, Tokyo , 1986.
10. Gers, F. A. & de Garis, H. *'Porting a Cellular Automata Based Artificial Brain to MIT's Cellular Automata Machine 'CAM-8'*, SEAL'96 Conference Proceedings S7-3 , 1996.
11. Gers, F. A. & de Garis, H. *'CAM-Brain: A New Model for ATR's Cellular Automata based Artificial Brain Project'*, ICES'96 Conference Proceedings S7-5 , 1996.
12. Margolus, N. *'Crystalline Computation'*, (Preprint).
13. Korkin M. & de Garis H. &, Gers F.A. & Hemmi H. *'CBM (CAM-Brain Machine) : A Hardware Tool which Evolves a Neural Net Module in a Fraction of a Second and Runs a Million Neuron Artificial Brain in Real Time'*, Genetic Programming Conference, July 1997, Stanford, USA , 1997.

This article was processed using the LaTeX macro package with LLNCS style