# Porting a Cellular Automata Based Artificial Brain to MIT's Cellular Automata Machine 'CAM-8'

Felix Gers Hugo de Garis

ATR Human Information Processing Laboratories.
2-2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan.
Phone: +81-774-95-1079 , Fax : +81-774-95-1008 ,
email: flx@hip.atr.co.jp & degaris@hip.atr.co.jp
URL : http://www.hip.atr.co.jp/~flx/ATRCAM8

**Abstract**

This paper shows how an implementation of a Cellular Automata based Artificial Brain with more that 100,000 neurons was ported to MIT's Cellular Automata Machine CAM-8, a non trivial problem, whose solution is vital for the 'CAM-Brain' Project. Since modern electronics allows, or will soon allow, brain-like systems with millions and even billions of artificial neurons, it is necessary to add evolutionary technologies to traditional human engineering skills. This paper shows how this can be done, as well as stressing the importance of using Cellular Automata as a dominant computation paradigm in the next decade or two. We believe this importance follows from the size of such billion-neuron systems and the general development of nano-electronics towards pico-second switching times. Pico-second-switching implies that only locally connected circuits can be used, due to the signal delay problem. We will have to develop purely parallel computing models with highly local interactions. Precisely these three criteria, namely evolvability, local connections and pure parallelism are satisfied by using Cellular Automata in parallel hardware.

**Keywords**

Artificial Brains, Evolutionary Engineering, Neural Networks, Genetic Algorithms, Cellular Automata, Cellular Automata Machine (CAM-8).

## Introduction

This paper shows how it was possible to port the Cellular Automata rules used to grow/evolve an artificial brain (the 'CAM-Brain' Project) from a work station to MIT's CAM-8 cellular automata machine [1] - definitely a non trivial problem and a vital step in the CAM-Brain work, if the project is to be successful. This paper also shows Cellular Automata (CA) [2] will have a bright future in general computing and the simulation of complex systems.

The prospect of nano-technology [3], quantum-computation [4] and the expected progress in traditional micro-electronics make it foreseeable that computer systems will continue to shrink to molecular scales. But the operation of systems which are huge in comparison to the molecular size of their basic components will necessitate working in parallel, otherwise the computational speed of the system will be inversely proportional to its size.

Molecular scale devices with pico-scale switching times [?] will create signaling delay problems. In a pico-second light travels only 0.3 mm, so that all signaling connections will have to be highly local, and that distributed operating systems will have to become mandatory.

Due to the huge number of components in molecular scale computer technologies, traditional "blueprint" design techniques will be quite impractical. These devices will have to self-assemble and the resulting complexities will make them impossible to understand in detail. However it is still possible to successfully build a functional device, without understanding how it works in detail, by using "evolutionary engineering" techniques. The power of evolutionary engineering can be combined with the high level design criteria of a human engineer, by having the engineer define the coarse functionality and the basic interactions of the smallest parts in the system, and having an evolutionary algorithm fill in the details. The power of an evolutionary engineering tool (such as Koza's Genetic Programming [5]) to generate solutions that human designers would never be able to accomplish, is shown in

question how a computational model for a highly local system should be chosen so that the described method of design can be applied. One answer is that the model should have the properties of general CA. In CA, the microscopic interactions are defined by CA-rules and -states. Higher level modules and their functions can be programmed by designing the CA-space and the connections between CA-spaces. The knowledge that a human designer already has concerning a system can be fed into that system as an initial pattern. This reduces the search space for the evolutionary algorithm.

The CAM-Brain Project is essentially the evolution of a brain-like structure based on CA. The CA act as the vehicle to conduct the growth, and in a second step, the neural signaling, of the brain-like structure.

# 1 Knowledge Combined with Evolution

In recent years, neuro-scientists have put a lot of effort into trying to understand and to reconstruct the basic functions of the brain. Almost all this research has followed a bottom up approach. For example, slices of the visual and auditory cortex have been extracted and investigated for their possible functionality in a larger context. The idea is that the many little insights will be amalgamated to reveal the functionality of larger and more complex parts, until finally the brain is reasonably well understood. The success of this approach depends on the functional composition of the brain and its major components. If it is possible to divide the brain into more or less independent compartments which are small enough for human researchers to comprehend their function, then this approach has, considering the large number of researchers working in the field, a good chance of succeeding.

But it is possible that many regions of the brain behave differently. Their behaviors may result from a sophisticated interaction between many components. In this case, it is not possible to extract a component and assign a certain function to it. The behavior of a region has to be considered and understood as a whole.

To cover this latter case, we add evolutionary engineering to the first approach. Our aim is to create a brain-like-system into which human knowledge can be inserted as initial conditions. Evolutionary algorithms combined with huge computing power may then be able to evolve artificial brains with the demanded all-over functionality of biological brains. We think that it is easier to investigate engineering-based brains than biological brains, and that the engineering solutions will reveal insights into the working mechanisms of the evolved system as a whole.

# 2 Using CAs as a Tool

To apply evolutionary technologies to a huge neural system 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 managably short. The bottle-neck of evolutionary algorithms is the fitness measurement time, i.e. the time needed to simulate the system. Hence it must be possible to simulate the system on massively parallel hardware.

With CA as a simulating tool, both demands 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. In our model shown in figure 1, a neuron body consists of a square $(11 * 11)$ of cells.

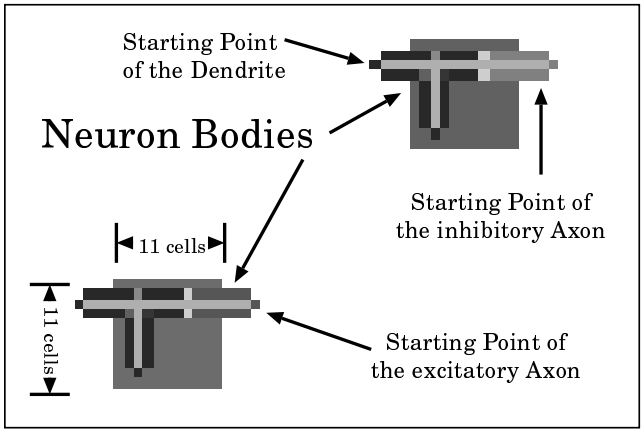


Figure 1: Our Neuron Model.

Changes to the model can be made easily, by adding a new feature or adding CA-states and additional rules. Since these model changes are hardware independent, they can be evolved by an evolutionary algorithm.

Since CAs are only locally connected, (our CAs for example use only the four nearest neighbors), they are ideal for implementation on pure parallel hardware. We use MIT's Cellular Automata Machine CAM-8 [1] for our CAM-Brain modeling.

## Brain

In the first stage of the CAM-Brain project a 2-dimensional neuronal structure is grown inside a CA space. The CA can be simulated on a workstation or the Cellular Automata Machine CAM-8. With 128 Mega cells of 16 bits in the CAM-8, and the current CA CAM-Brain model, it is possible to grow a network with more than 100,000 neurons. The instructions to grow the network are held in a "chromosome" (see figure 3). With an evolutionary algorithm working on this chromosome, it is possible to evolve and adapt the structure of the neural network to a specific task. These principles have already been described in many papers by de Garis e.g.[7], who evolved an artificial retina which could detect the vector velocity of a moving bar of "light".

### 3.1 The Neuron Model

The network consists of two types of neuron, excitatory and inhibitory. At first, neuron bodies are grown from an initial pattern, as shown in figure 2. The starting positions of the seed cells can be either user specified or placed under genetic control.

To each grown neuron is attached a "chromosome" block. These chromosome blocks are stored in a subspace "under" the actual CA-space as shown in figure 3.
With these growth instructions, each neuron grows an axonic and a dendritic tree, which later will be used to conduct neural signals. The CA trails in our model are inspired by Codd [8], i.e. they are three cells wide, with "sheath" cells outside, and the signals moving down the middle. A collision of a dendritic and an axonic branch results in the formation of a synapse. The synapses are not weighted. The weighting is expressed indirectly via the length of the dendritic trail that a signal has to pass through. A signal decreases linearly in strength while traveling along a dendrite, so the weighting can be adapted by evolving the length of the dendritic trail.

This kind of genetic coding has an advantage and a disadvantage. The advantage is that the chromosome is short. A neuron needs about 10 growth signals, (equivalent to 80 bits), to grow its axonic and dendritic trails. We have grown networks of more than 100,000 neurons on the CAM-8, so short chromosomes are necessary for the evolutionary algorithm keep the amount of population data to a manageable size.

dom. As growth signals split at a junction, the same signal advances into the forked trails. Due to this "fractal" nature, not every possible tree structure can be grown; However as evolutionary experiments carried out by de Garis [7] have shown, the evolvability of these "fractal" circuits is still acceptable.

Each neuron grows its dendrite and axon with its individual chromosome information, until either all the chromosome information is used, or there is no more growth possible in the given CA-space. Alternatively, the chromosome can be wrapped around, so that the available space imposes the limit. Figure 4 shows examples of early and saturated growths.
With the CAM-8 hardware it is possible to simulate a CA-space of 128 million 16-bit-cells and to update them faster than 6 times a second. As our implementation needs 32 bits for each CA cell, we have a space of 64 million cells available, in which we can evolve more than 100,000 neurons, assuming that each neuron is given a space of $25^2$ cells. Figure 5 shows a 2D CAM-8 space with a network of more than 10,000 neurons. We chose the neuronal density to be lower than the maximum in order to highlight the axonic and dendritic trees.

## 4 CAM-8 Cell Implementation When the Neighborhood is Too Large

The CAM-8 hardware has 16 bits of state memory for each cell and hence a $2^{16} = 64$ K address look-up-table (LUT) is needed to map the possible cell states to their successors. Hence an automata cell with e.g. 4 neighbors, and 3 bits for the state value, can be updated in one machine cycle, (i.e. 3 bits for each neighbor and 3 bits for itself = 15 bits).

The CAM-Brain CA uses the von Neumann neighborhood (= 4 neighbors) and 8 bits for the cell state, hence 40 bits, which allow our CA, $2^{40}$ possible rules to govern the behavior. However, in our case, almost all of the $2^{40}$ rules are useless, because they refer to situations in the automata that never occur, due to the restrictions imposed by the initial pattern that is loaded into the cell-space. In our case, about 32,000 rules of the $2^{40}$ rules are actually used.

There are two ways to adjust the 40 bit "neighborhood" to the 16 bit LUT of the hardware. One can try to rewrite the CA rules with fewer states and then use the full range of rules, but in our case

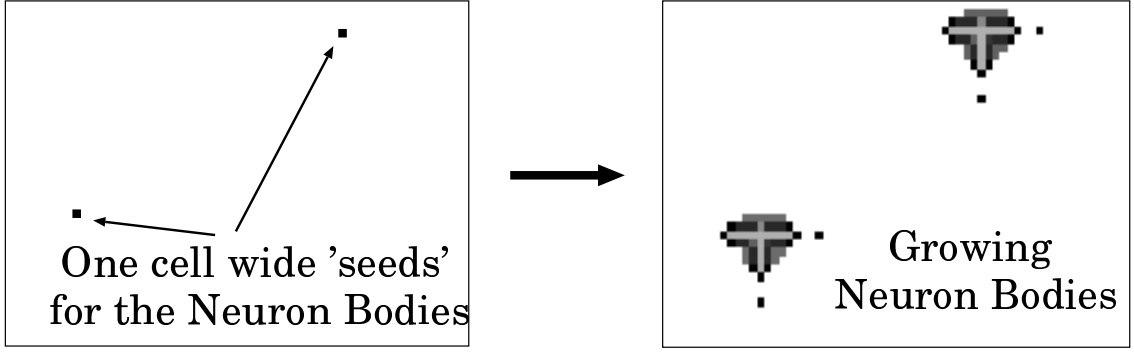


Figure 2: The growth of the neuron bodies (right) from an initial pattern (left).

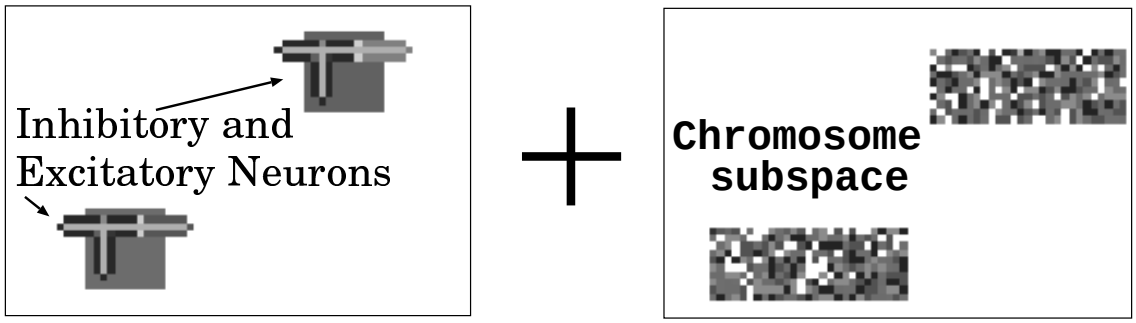


Figure 3: The neurons (left) with their individual chromosomes (right) A chromosome contains 6 different kinds of growth signals : Grow straight, turn (left, right, both) and split (left, right). Every CA-cell of a chromosome represents one growth signal.

this is not possible.

Alternatively, the CA rules can be ranked. The first 16 bits (= LUT size) of the 40 bit string are transfered to the first of several LUTs. These 16 bits will be the same for many of the rules, so that they can be renumbered or re-ranked so that each occurring bit combination of these 16 bits is assigned to a number between 0 and the number of different bit combinations. We will call this renumbered table a rank-table. The first rank-table will not need all the 16 bits of the LUT, so that some of the remaining left-most bits of the bit-string can be attached to the RHS of the first rank, loaded into the LUT and transfered into the second rank-table. This process is repeated until all 40 bits are ranked, so that the final rank leads to the new cell states and the update is complete. Figure 6 shows schematically the working mechanism.

This algorithm can be applied to a LUT which is too large for a given piece of hardware, such as the CAM-8, by transforming it into a number of smaller LUTs. In the case of CAM-Brain, a 40 bit LUT was transformed into 13 LUTs of 16 bits each. To illustrate the result of the algorithm, figure 7 shows the contents of a CAM-8 cell during the execution of the CAM-Brain program. After the last cycle one CA-update is completed and the cell contains the new cell state, which is then sent to the display. Every CA cell consists of 32 bits, in two 16 bit subcells. 16 bits of these 32 bits can be sent to the LUT at a time. They are shown in figure 7 with a striped background. In the first CAM-8 cycle the state of the center cell (Center = 8bits), and the state of the upper neighbor (Top: T= 8bits) are sent to the LUT, which contains the first rank-table. In return, the first rank (Rank 1 = 11 bits) is stored in each cell. In the second cycle, this Rank 1 and the first 5 bits of the state of the Right neighbor are sent to the LUT and transfered into the second rank (Rank 2). This procedure continues until all 40 bits (8 bits from the Center cell and each of the four neighbors) are processed and the last rank (Rank 13), together with the last bit of the last neighbor (Left) to be looked at, return the new cell state Center-new (Center New).

Since every re-ranking uses one CAM-8 machine cycle, the number of rank-tables should be minimized to reduce the total updating time. This is

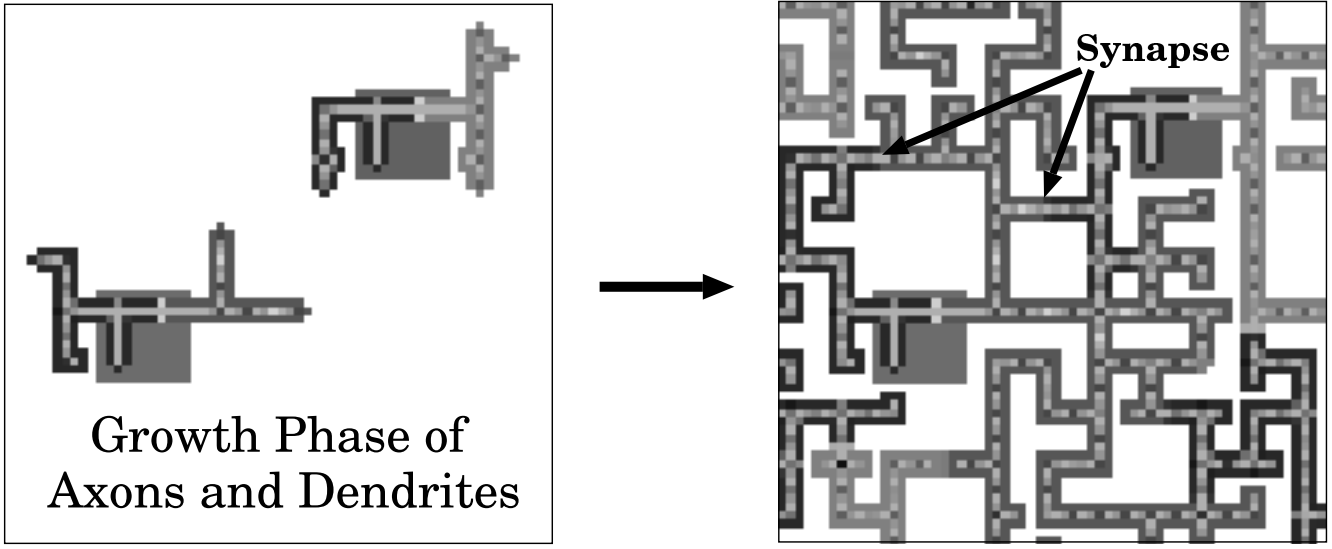


Figure 4: (Left) The growth phase of the network continues until all CA trails are connected through synapses or are blocked. (Right) A fully grown network.

equivalent to the demand that the average number of unused bits after each re-ranking-step be maximized in order to load the maximum number of new bits from the 40 bit string. This average number and hence the number of rankings depend on

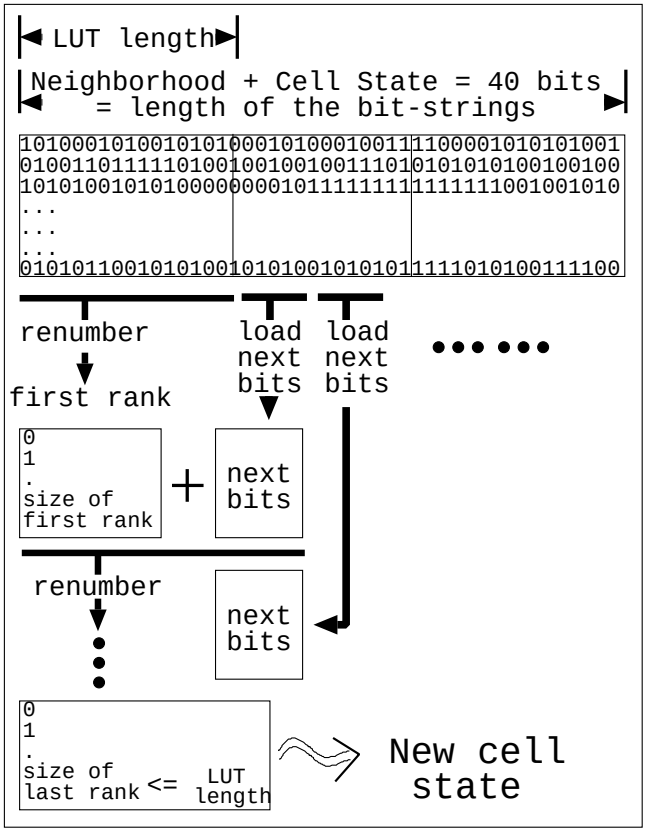


Figure 6: Scheme of the Ranking Algorithm.

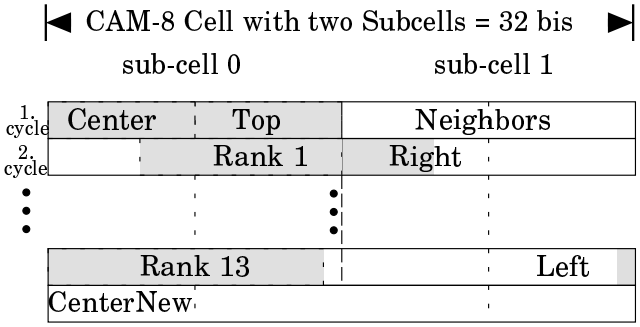


Figure 7: Simplified scheme of the of the CAM-Brain program using the LUT-size reduction algorithm on the CAM-8.

the bit-order in the bit-string. Finding the right order in which to load the bits is not a trivial problem because the defined rules have to be regarded as arbitrary data. One way is to systematically test all the bit combinations. The number of possible combinations also depend on the data and can roughly be estimated with:

$$\frac{B!}{L! \cdot (B-L)!} < N_c < \frac{B!}{L!} \qquad (1)$$

$N_c$ : Number of distinct bit-orders
$B$ : Length of the bit-string
$L$ : Length of the LUT in bits

The number of distinct combinations lies between these two limits because the order of the bits loaded together into one rank is arbitrary so that

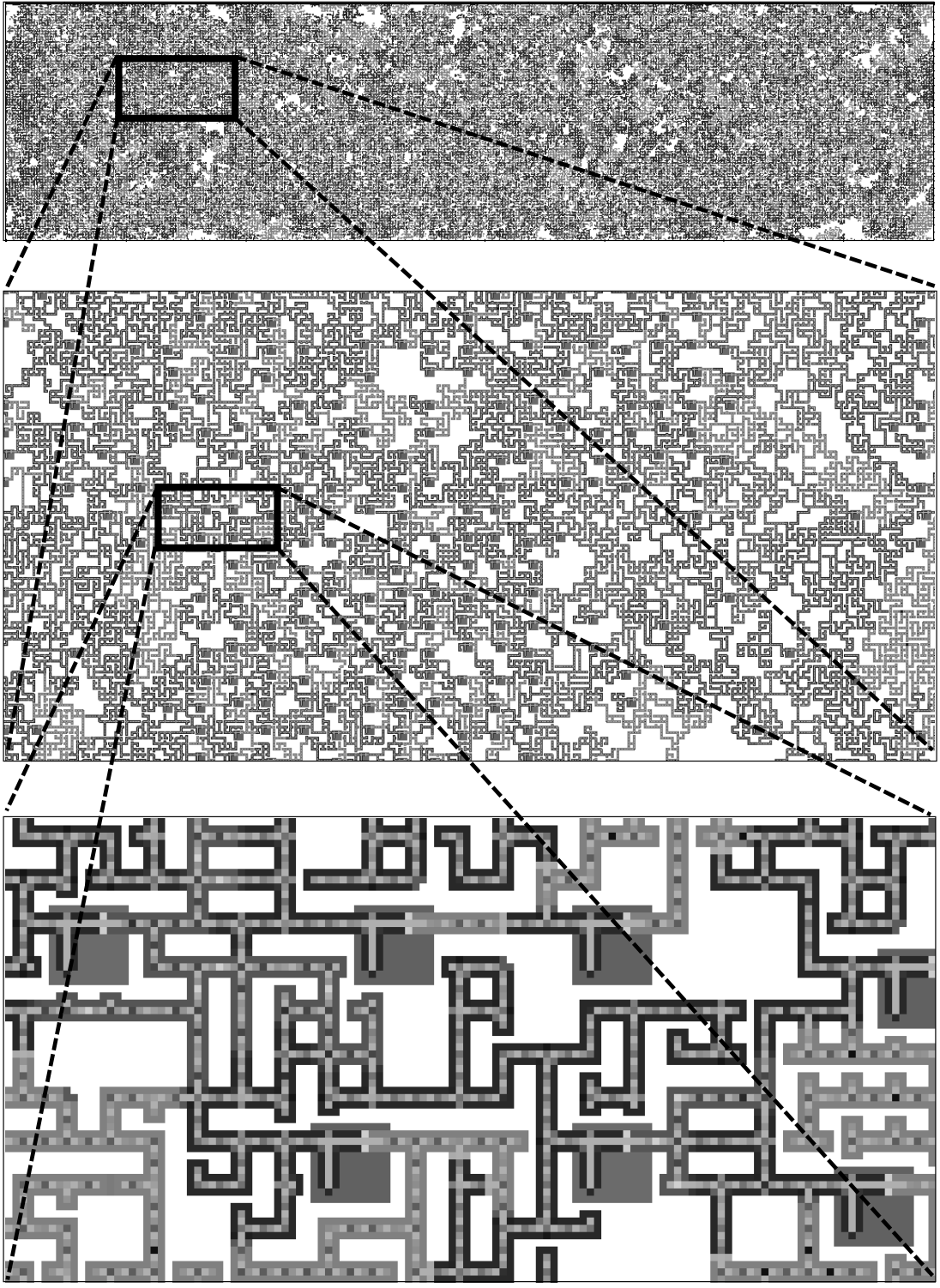

Figure 5: (Upper) A 2D CAM-8 space with 16 million 32-bit cells and more than 10,000 fully grown neurons. (Middle) A zoom (x32) showing neuron bodies, the two kinds of axon (xcitatory and inhibitory) and the darker dendrites. (Bottom) A zoom (x50) showing seven neurons with local circuitry.

the right hand fraction still has to be divided by the factorial of all these rank sizes. They also depend on the concrete ordering of the bit-string, and in general are different for each distinguished order. But, as the product of all bit-block sizes is always smaller then (B-L)!, the left side of (1) can serve as a lower limit to $N_c$. In the case of the CAM-brain implementation, $B = 40$ and $L = 16$, so that $6.3 \cdot 10^{10} < N_c < 3.9 \cdot 10^{34}$.

To find a good ordering and hence a number of

essary to go through all possible combinations $N_c$, which is almost impractical anyway, as a Sun Sparc 5 needs several minutes processing time for each. The following procedure will find a satisfactory order by testing $\sum_{i=1}^{B} i = \frac{B}{2} \cdot (B+1)$ bit combinations, if no equal orderings occur. So for our case, with $B = 40$, $N'_C \approx 820$ runs will be enough, using the following procedure:

Take out one bit, renumber the remaining $B-1$ bits. Store the size of the resulting rank and repeat the process with each of the $B$ bits. The bit that leads to the smallest size of the corresponding rank, is the bit number $B$, i.e. the last bit to be loaded. After this first step, the remaining bit-string of $B-1$ bits is processed in the same way, until all the bits are ordered. If sorted bits score the same in a given step, they are both stored, and the result of the next step decides which bit is removed and which bit remains in the bit-string. In this way, a tree-like structure of equal scoring bits is held in the memory. If the are no equal scoring bits in a step, the tree is immediately reduced to one "leaf". If equal scoring bits remain after the $B$'s and the last step, one of them can be arbitrarily chosen, because they all lead to the same number of ranks.

The algorithm explained in this section solves the problem of implementing a CA whose number of state bits multiplied by the number of neighbors, is larger than the number of bits that can be loaded into the LUT which maps to the new state.

One remaining limitation is that the number of defined CA rules must not be larger than the total size of the LUT, i.e. for $L = 16$ the number of rules must be less than 64 K. If these criteria are satisfied, the above algorithm can be used when a CA "neighborhood" is too large to be implemented on a given CA simulation hardware, such as the CAM-8.

## Conclusions

This paper shows how we ported the CA rules for the CAM-Brain from a work station to the CAM-8 machine, a difficult task, whose solution is vital to the whole CAM-Brain project. We also presented our ideas concerning CA as a computational model for parallel hardware. We showed how "semi" parallel CAM-8 hardware was used to grow and evolve complex networks, based on CA. These CA based networks were shown to be evolvable in earlier work by de Garis [7]. We expect that the CA based computational model will allow us not only to build artificial brains, but will become essential when ing times) emerges from molecular scale massively parallel 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] Drexler, K.E. *'Nanosystems: Molecular Machinery, Manufactoring and Computation'*, Wiley, NY , 1992.

[4] Lloyd, S. *'A Potentially Realizable Quantum Computer'*, Science **261**, 1569-1571 1993.

[5] Koza, J.R. *'Genetic Programming: On the Programming of Computers by Means of Natural Selection'*, Cambrige, MA, MIT Press , 1992.

[6] 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.

[7] 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.

[8] Codd, E.F. *'Cellular Automata'*, Academic Press, NY 1968.

[9] Sipper, M. *'Co-evolving Non-Uniform Cellular Automata to Perform Computations'*, Physica D **92**, 193-208 1996.

[10] Carter, F. L. *'Molecular Electronic Devices'*, North-Holland, Amsterdam, NY, Oxford, Tokyo , 1986.