An Evolvable Micro-controller or what's new about mutations?

Uwe Tangen
FhG – Fraunhofer Society
Biomolecular Information Processing
Schloss Birlinghoven
53754 St. Augustin, Germany

Abstract

Un-supervised learning of complex software projects is still an issue. Additionally, the objective to create open-ended evolutionary systems is lacking a confident realization. Long-term evolutionary behavior results in noisy regimes or stable fixed points including limit cycles. The evolving organizations are more or less simple or directly reflecting the programmer's point of view of a complex organization. One possible way to overcome these limitations is to drastically increase population sizes and look at longer timescales. The work reported serves two purposes: the introduction of an evolving micro-controller inclusive evolving assembler code on the new massively parallel configurable hardware MereGen^{TM}and it provides new insights on how mutation probabilities may influence the evolutionary outcome of evolving populations.

With 648 MBytes of fast SRAM and about a Gigabyte of SDRAM, millions of programs and thousands of micro-controllers can be evolved concurrently in hardware. Typical speedups are about five orders of magnitude compared with current day high-end PCs. Thus, longterm evolution of large populations is no longer the bottleneck and limitations in our understanding and the observability of evolving systems immediately grabs the focus of our attention in evolutionary experiments.

1 Introduction

The study of evolving hardware aims at creating a new power level of man-made machines. Evolving hardware is expected to be able to adapt to unknown environments and to find solutions hardly found by humans. Evolution, at least in the biological context, is intimately bound to information processing with strings typically considered as information carriers. Machines working on their own description are especially well suited to study the scope of evolutionary concepts.

The idea of information processing machines operating on themselves was first published in 1842 [1] by Menabrea in a translation from Ada Augusta, Countess of Lovelace:

> ...
> Considered under the most general point of view, the essential object of the machine being to calculate, according to the laws dictated to it, the values of numerical coefficients which it is then to distribute appropriately on the columns which represent the variables, it follows that the interpretation of formula and of results is beyond its province, unless indeed this very interpretation be itself susceptible of expression by means of the symbols which the machine employs. Thus, although it is not itself the being that reflects, it may yet be considered as the being which executes the conceptions of intelligence. ...

Though they didn't really attempt to build a machine capable of understanding itself – the idea of such an endeavor was not completely out of the question. They had at that time established the principles of a programmable computer (e.g. conditional execution and loops have been explicitly mentioned). Today we have the hardware to really delve into this domain where electronic hardware is able to self-organize and to evolve.

Despite the famous book of von Neumann [2] and his

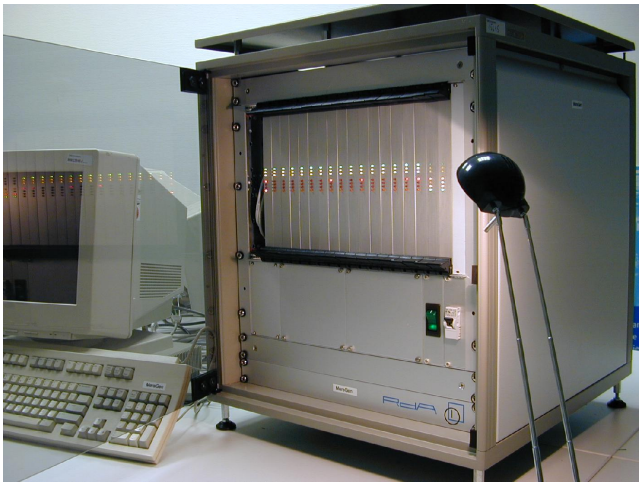

Figure 1: MereGen^{TM}
The 19 inch rack of MereGen^{TM} is shown with 18 boards.

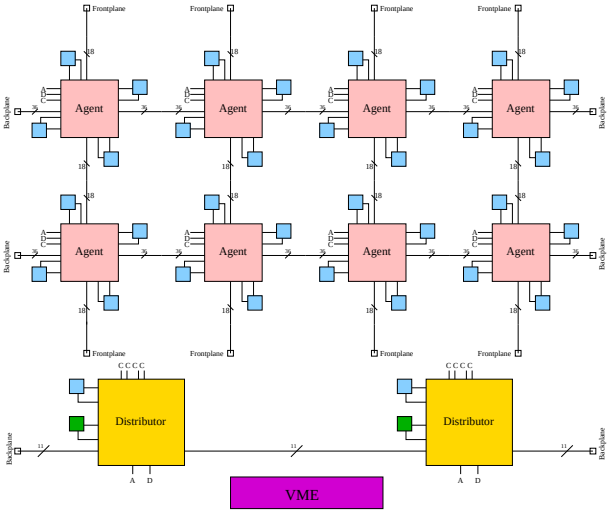

Figure 2: Sketch of the board layout
A board in MereGen^{TM} is organized hierarchically. At the bottom the VME-bus interface is drawn (Xilinx xc4013xl). The two distributors (Xilinx xcv300e) each control a block of four agents (Xilinx xcv600e). The evolving microcontrollers (μCs) are configured into the agents. A further μC, configured into each of the distributors, is dedicated to the sequential part of controlling the experiments. All agents are working continuously and simultaneously with a 64 MHz clock. Attached to the agents four fast 7.5ns 9 MBit SRAMs are used to store the evolving program data.

self-reproducing automata – it was not until 1984 that the extension of the original idea of Charles Babagge and Ada Augusta was revived – probably not known to the author then – by Dewdney in his article in Scientific American [3] about Core Wars. Dewdney had been inspired by the first worms and viruses being observed in the computer labs of Xerox. In the aftermath, several people tried to evolve programs [4, 5, 6] and seriously attempted to realize self-organized software systems. There, non evolvable register machines determine the artificial physics evolving programs are subjected to. A special minimalistic evolving system has been investigated in [7] with only two instructions available for evolution – replicate the genome or do nothing.

In a more general context, these register machines might be approximated by simpler cellular automata, see e.g. Codd [8] or Langton [9]. The comprehensive field of genetic programming introduced by Koza also uses evolving programs but not in a self-organizing context – they solve optimization problems though some work has been undertaken to realize these optimizations problems in hardware [10].

The work reported in this paper solely focuses on the self-organizing context without any optimization problems in mind. These are to be tackled in a subsequent step when self-organizing systems are better understood. The rational behind this work is not only to try to understand the transition from the non-living to the living world but more importantly to lay the foundation for sparsely controlled nano-systems. Massively parallel dynamical systems in the nanometer range can no longer be controlled in a detailed manner. They cannot even be observed in a detailed manner. In order to cope with these types of systems we must allow them to organize themselves – otherwise we will not be able to use dynamical nano-systems.

Independent of the description level used, programmable machines being physically realized have to obey certain constraints. Most often conservation laws are mentioned. Taking into account dissipative structures, conservation of energy can be abstracted relatively easily. The results of this work show the great importance of the following aspects: 1) physical properties like limited space – physical entities require space to be situated, 2) timing delays – information traversing from point A to point B needs a certain amount of time and 3) routing – why does point X know that point C is in need of exactly this data and of nothing else? Of course these physical properties might be simulated easily on current day PCs or workstations – paying the price of slower simulations. What

is not easily simulated, is the inherent explicit parallelism of chemical or biological systems. The overhead for these to simulate is considerable. Fine grained massively parallel machines have a great advantage in this respect. The more the simulations become hardware dependent, the less effort has to be made to model physical properties – they are inherent. Of course, the price is now a restriction in the number of possible physical model systems.

The work described here has been done on MereGen^{TM} [11], fig. 1, a massively parallel configurable hardware using up to 144 high performance field programmable gate arrays from Xilinx [12] dedicated solely to user designs. With MereGen speedups of up to 200.000 (five orders of magnitude, two to three orders due to parallelism and another two to three orders due to exploitation of hardware instantiated physics) have been realized.

MereGen^{TM} uses field-programmable-gate-array (FPGA) and VME64x-bus technology and was especially designed for the simulation of biological model systems and for doing research on evolving hardware. With an approximated four tera instructions per second and over 600 MByte of high speed static memory, MereGen^{TM} allows research which has not been possible with computer hardware in the quarter million dollar regime. The eighteen boards provide more than 8.5 million gates equivalents for user designs. In fig. 2, the organization of a board is shown. A Linux operated 400 MHz PowerPC connects via the VME-bus standard to the bus-interface at each board and controls the chip configuration as well as the readout and experiment scheduling.

2 The hardware model

Having studied evolving Boolean hardware [13, 14] the apparent mismatch between the needed length of description and the computational power of the resulting Boolean apparatus made us use micro-controller (μC) type machines. It was hoped that the brittleness [15] observed in the evolving Boolean hardware could be managed more easily in a micro-controlling context.

The idea of the evolvable μC is derived from [16] a minimalistic form of a RISC machine with only one instruction (MOV) left. Manipulating data is realized here through side effects when moving data from the input port to the output port or a register. Several thousand μCs (the maximum achieved were 3456 μCs in MereGen^{TM}) are chained up in such a way that each μC has access to the upper element of the stack of the neighboring μCs and to one of their registers. This chain actually is a closed loop like in the work of Rasmussen [4]. In the experiments reported, 16 μCs per chip were used. These 16 μCs share the four SRAMs available at each agent. Each μC gets access – in a round robin fashion – to the attached SRAMs for a period of 1024 clock cycles and writes its programs or stack values in the SRAMs which are operated as shift-registers. The SRAM data on the other hand is available to all μCs on the chip at the same time.

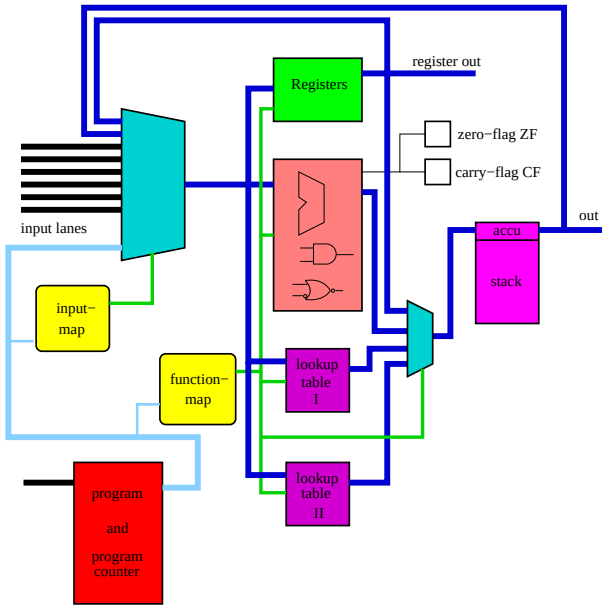

Figure 3: The evolving micro-controller (μC)
Only one instruction (MOV) is available for programs using this 8-bit μC. With just one instruction left, only the parameters are coded in one byte of the program code. The upper four bits of the instruction stream determine the location from where data is moved towards the location determined by the lower four bits of an instruction. Whether flags are tested or functions are executed depend only on the particular location addressed. With an additional mapping of the possible 16 input (input map) and 16 output (function map) locations to different input ports and different functions, extremely dense code can be realized. This mapping is defined at runtime and can therefore be subjected to evolution. In addition, two arbitrary function generators (lookup-tables I and II) which are able to realize any unary function with 8 bit wide I/O can be used to be changed by the evolving system at runtime. Thus a plastic μC is available for the hardware evolution experiments. The program size can be varied between 128 memory partitions at 8 instructions each and two partitions at 512 instructions each. Two special side-effects SWI_SRC and SWI_DEST are available to change the active partitions.

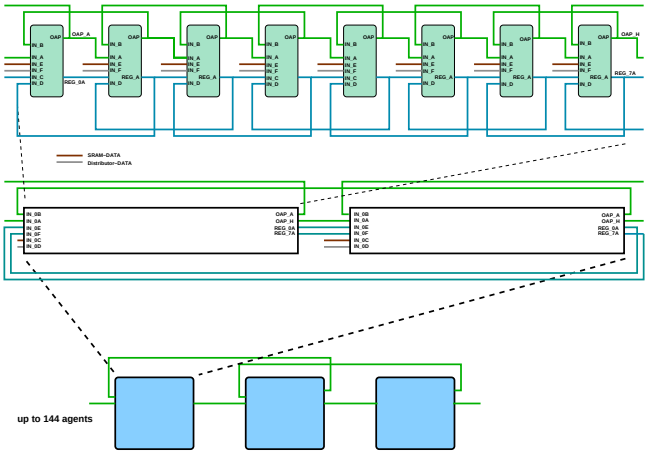

Figure 4: Chain of up to 3456 μCs

Up to 24 μCs configured into a chip give a chain of up to 3456 μCs working in parallel and at a clock rate of 64 MHz with a minimum number of one instruction per clock cycles. The design for 16 μCs is shown which has been used in most experiments. The four SRAMs attached to each agent are operated as 64 Bit memory and each word containing 8 instructions which are serially fed into the μCs at a special input port.

This means that a program in principle could be easily replicated by a factor of 16 if and only if the μCs on the chip are listening.

As already mentioned, routing of information is a central issue for evolving spatially extended information processing systems. To test the capabilities of the evolving μCs and to find a measure for evolutionary abilities [17] the topology of the system is made scalable in the range of more than two orders of magnitude. This means that by pure parameterization of the system at runtime the chain of μCs can be partitioned in 144, 36 and 18 pieces or used with full length. When designs are used with only eight μCs per chip even smaller evolving chain partitions can be investigated. According to [17] a shadow model is defined as an evolutionary model with interactions between individuals in the evolving population disabled. Here, isolating μCs serves the same purpose. Comparing the different topologies with each other, the experiment with the larger number of partitions can be seen as a shadow model. Experiments with special disabled functions allow to grab the semantics in the system as it is a typical procedure in (gene-) knock-out experiments in micro-biology.

3 Observing the system – complexity measures

Observing a self-organized evolution system is not at all trivial. In contrast to Ray [5] and others, the system is not seeded with a known self-replicator. The reason for this is that functional cells are NOT thought to exist *a priori* like the evolving systems of the Tierran type. Thus, it is not known what to expect in the system. The next problem connected with observation is the sheer amount of data being processed by the machine – several tera-bytes per second! Both problems ask for an independent area of research.

> A preliminary attempt to observe the system follows the idea that what is important will be existent in several copies in the evolving system!

An algorithm has been designed to extract all patterns in the genomic data which occur at least $k = 14$ times [18] and $min\_len = 30$ instructions. The patterns themselves are arbitrary. This algorithm is part of a compression algorithm and approximates Kolmogorov entropy [19] to a certain extent. The maximum length of the pattern searched for is restricted to 270 instructions – the reason being a limited amount of main-memory in the host computer.

All complexity measures used in this work are taking this pattern analysis data as their base and not the original genomic data. Of course, tera-bytes per second cannot even be written in the main-memory of the computer. Only a small fraction of programs is read out of the agent chips via the distributors in a regular fashion. These programs are then picked up by the host and are used for the pattern analysis.

The complexity measures used are as follows:

- EXPENSE: The pattern searching algorithm searches for repeating pairs of instructions in all picked up programs. From these pairs of instructions repeating quadruple instructions are searched until no further repeats with a minimum occurrence are found. In a last step it is attempted to elongate the longest patterns found so far. Every attempted combination of new patterns is recorded as a unit effort and counted. The total number of unit efforts divided by the total number of instructions gives the complexity measure EXPENSE.
- DIVERSITY: This is the most simple complexity measure used here. The frequency of all instruction pairs is counted. There are 2^{16} different

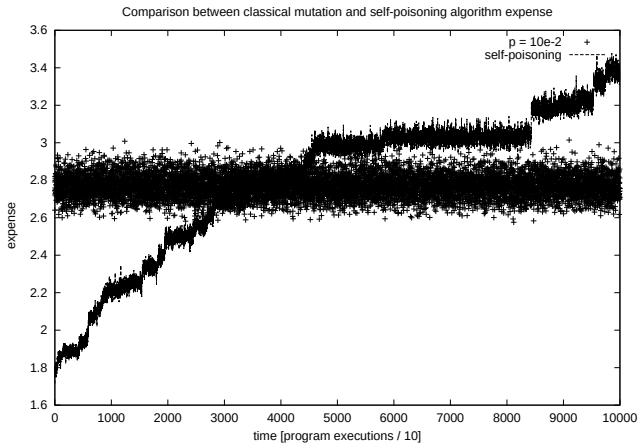

Figure 5: Two different variation systems
Two different mutation processes are shown. The classical mutation procedure with a fixed mutation probability, in this case $p = 10^{-2}$ is used, is shown with the curve '+'. With the second mutation procedure the frequency of each instruction in the genomic data is determined. In the next program iteration the most frequent instruction is replaced by a random value. Not only can different longterm evolutionary behavior be observed but also a higher value of the complexity measure EXPENSE is achieved. This means that the pattern search algorithm has increasingly more difficulties to determine repeats in the genomic data.

instructions pairs possible. The diversity is calculated as:

$$diversity = \frac{(max - min) * nr\_pair}{nr\_inst},$$

with min the count of the instruction pair with minimum frequency, max the count of the instruction pair with maximum frequency and nr_pair the number of all different instruction pairs found in the genomic data of the system.

- PATCOMP: The number of pattern levels (pair, quadruple ...) in the sequence data times the number of patterns of $min\_len = 30$ instructions found in the first iteration of the pattern search algorithm.

4 Mutations – what's new?

So far, in almost all evolutionary models published, e.g. [20] and followers, mutation is thought to be a random event, like a cosmic ray changing the information content of a sequence of letters in a uniformly distributed manner. Of course, in molecular biology it is well known that mutational events are only in rare cases evenly distributed over the genomic sequences which is thought to be a consequence of the selection pressure subjected to the system investigated. Mutation in this work always means exchange of an instruction by a random instruction if a mutational event occurs. It is futile to try and carry out bit-mutations because the fitness consequences per bit-change cannot be expected to be smooth!

Realizing a first C-simulation to test different μC-variants, it soon turned out that the evolving system more or less immediately got trapped in a stationary behavior, fig. 5 (curve at $p = 10^{-2}$). Considering the Gaya-hypothesis and the role oxygen might have played at that time, the idea of mutation as a **self-poisoning** process emerged. The idea is simple: the most successful sequence or species produces so much waste (waste is useless or even toxic per definition) that it destroys its own base of existence.

In the μC-model this could easily be established, e.g. by counting the number of instructions used in all programs. After determining the instruction with the highest frequency this instruction is destined to be completely randomized in the following round of program execution. Whenever this instruction is encountered in the μC, a uniformly distributed random number is taken instead. Other mutational events have been completely abandoned. The resulting behavior of the system changed dramatically, fig. 5 (increasing curve), non-trivial longterm evolution could immediately be observed. The reaction of the evolving system is to develop increasingly complex strategies to avoid dominance of only one program part in the system because this program would almost certainly be destroyed.

5 Experimental setup and results

A typical experiment lasts about seven minutes. At the beginning all the chips (distributors and agents) are configured and the parameters are written to local registers of the designs. Then the agents are activated and for 400 seconds a 64MHz clock continuously drives all chips in parallel. Micro-controllers in the distributors read out the programs evolving in the agents and when necessary count the frequency of instructions in the programs. Instructions, according to the parameters chosen, with the highest frequency are then written back into the agents in a certain register. If the system is working in a self-poisoning mode these instructions are exchanged by random instructions. The total design is constructed such that the evolving agents are not directly coupled with the ob-

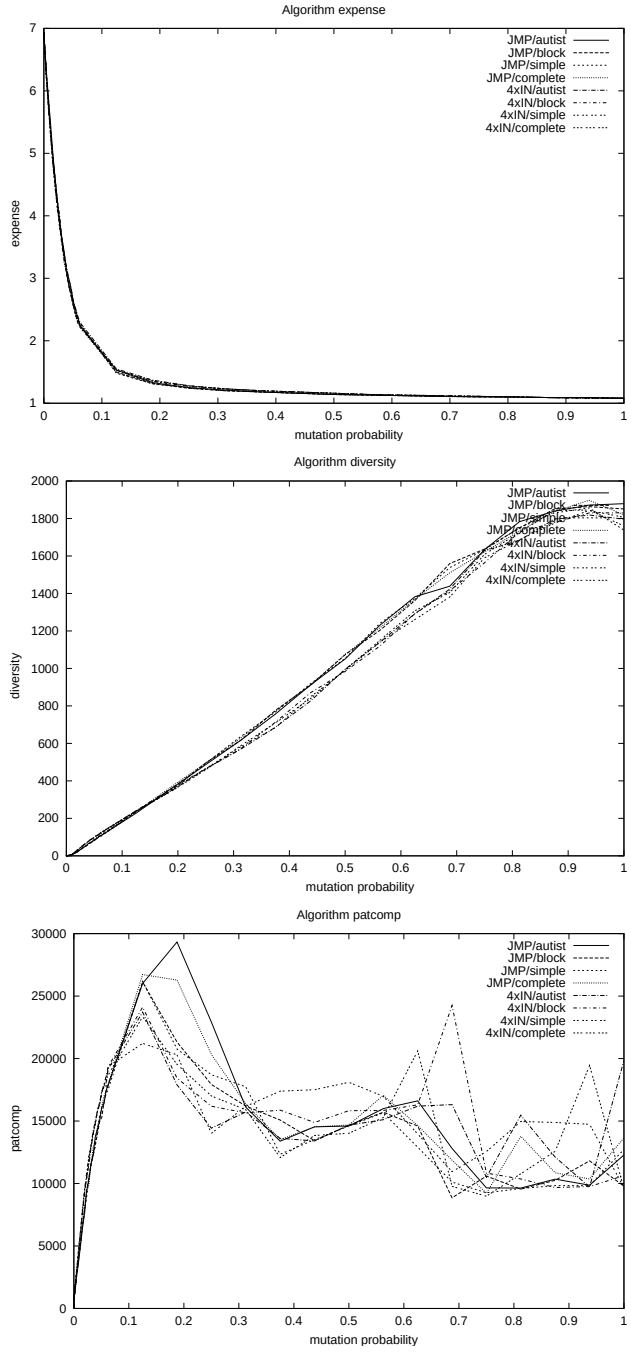

Figure 6: Different mutation probabilities

Each plot comprises eight series. Each two series have different connection partitions (autist = 128 partitions, block = 36 partitions, simple = 18 partitions and complete = 1 partition). JMP curves have a JMP-operator in the mapped function space and '4xIN' curves have four input lanes for reading the SRAM data. The top-down sequence is EXPENSE, DIVERSITY and PATCOMP at the bottom.

serving host. Because of the continuous behavior and the lack of naturally defined generations (what are the species and how long do they live?) in the system the time-base chosen is real-time.

In fig. 6, a bunch of experiments on MereGen^{TM} is shown. Each plot needed about one and a half day of operation time on MereGen^{TM}. The values plotted are average values of the last twenty complexity measure evaluations per run. A variation of the mutation rate has been investigated. As expected, it has turned out that the system depends on the mutation rate which is changed from rare mutations $p = 0, \frac{1}{2^{16}}, ..$ towards high mutation rates $p = 1$. Taking e.g. the upper plot in fig. 6, the effort needed to find all repeats is highest at low mutation rates. This is quite natural because if all instructions are identical, many different levels of repeats exist. Below two evolved program are shown. The first one is taken with a mutation rate of $p \sim 10^{-5}$ the second with a mutation rate of $p \sim 10^{-1}$ when PATCOMP has its maximum value. The listing only reflects one brief glimpse on an evolved program. Though it could be observed that programs might dominate the system for several seconds, trivial stationary behavior is not apparent.

```
# p = 1.5e-5 pattern length = 270 inst. (maximum)
   input section (4 bits)    function section (4 bits)
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _IN_B_ -----        -> _SWI_SRC_A_
 #    _COM_ -----         -> _WRI_REGA_A_
 #    _COM_ -----         -> _WRI_REGA_A_
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _OAP[7:0]_ -----    -> _IF_CF_PUSH_
 #    _IN_E_ -----        -> _SWI_SRC_A_
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _IN_B_ -----        -> _SWI_SRC_A_
e.g. take value of register _COM_ and store this value
                    in register _REGA_
 #    _COM_ -----         -> _WRI_REGA_A_
                 .
                 .
                 .
```

In the second listing two commands are sometimes depicted on the input section. This is a consequence of the possibility to execute more than one instruction per clock cycle. The notation _WRI_REGA_A_ means that the value taken from the input section is written into register A with command mapping A. In addition, most of the commands can be executed via the command mapping B which is activated if the command _EXE_COM_ is encountered. This allows, at the price of lower execution probabilities, the increase of the possible command set.

```
# p = 1.25e-1 pattern length = 210 instructions
                 .
                 .
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _COM_ -----         -> _IF_CF_ZF_PUSH_
 #    _IN_B_ -----        -> _WRI_COM_A_
 #    _OAP[7:0]_ _IF_ZF_POP_     -> _WRI_COM_A_
```

```
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _OAP[7:0]_ _IF_ZF_POP_     -> _WRI_REGC_A_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _REGA_ -----       -> _WRI_REGB_A_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _OAP[7:0]_ _POP_   -> _ADD_VAL_A_
#    _REGC_ -----       -> _IF_CF_PUSH_
#    _COM_ -----        -> _PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
              .
              .
```

6 Discussion

In the research on self-organized dynamical systems there are two basic questions: firstly the question on the transition from the non-living to the living world and secondly the need to abandon detailed control of complex nano-systems. Though the first question might sound academic, the opposite is the case. If we know how artificial life can be created we probably will know much more about creativity as such and other higher mental processes. The second question springs to mind when e.g. in DNA-computing [21] the I/O of data becomes an extraordinary issue.

The use of dedicated hardware in the research of biological model systems has both benefits and drawbacks. The benefits are obvious – the systems to be investigated are closer to real biological systems – in scale and because physics are inherently present. On the other hand, the large amounts of data processed by the hardware and its speed forbid the detailed analysis of all sequences in the machine. Now, as it is the case with molecular biology, filtering the essential data, realizing experimental controls and being content with much less powerful tools is the prize which, inevitably, has to be paid. The work reported truly lies on the verge between computer science, electronics and molecular biology.

A short discussion of the results:

- It could be shown that it is indeed possible to realize pure computer science models of self-organizing systems in hardware. It can be argued what self-replication really means. Looking at the sequences sometimes gives the impression that these are selected because of their bit-properties and not of their functional behavior. Nevertheless, these repeats occur in high numbers and are not random at all. There are a lot of intermediate pattern forming processes between the repeating structures found in crystals and e.g. bacterial cells.

- A highly parameterizable and evolvable μC has been presented with a minimum number of one instruction per clock-tic and thus a power of 64 MIPS. Optimizing the design should allow for μCs in the 80MHz range within MereGen^{TM}. With e.g. 3456 processors, a total power of 221 GIPS of customized μCs have been realized.

- The view on mutational events has broadened in the sense that self-poisoning promises to open up a new class of evolutionary power. Certainly, it is different from fitness sharing [22] or resource constrained evolutionary systems, see the appendix below.

- The task to build artificial living systems still remains. Longterm experiments done so far did not exhibit an escape from stationary behavior. The scaling experiments, fig. 6, did not show any dependency on the connection topology. All different partitions resulted in more or less exactly the same complexity measures. It seems that the selection pressure towards robustness of the evolving programs is so strong that inputs from other processors are deferred as much as possible. Looking at the sequences of the evolved programs (data not shown), no essential differences between the partition sizes can be observed. Another reason might be that no substantial organization developed and the resulting inhomogeneous sequences are a pure consequence of the physical constraints being felt by the system.

  Though this is bad news, the good news are that if one finds the means to let the evolving system display a particular scaling behavior then something important has been learned. Up to now, a statement often made that only the system size has to be sufficiently large to realize the emergent behavior desired cannot be validated.

- With this hardware realization, even non-hardware designers are able to investigate complex hardware models because the flexible μCs are suited to many different applications. With the additional broadband interconnect available on each board, even external apparati might be attached.

7 Conclusions

With MereGen^{TM}, several absolute measures become feasible: Firstly, simulation time is no longer the bottleneck – the speedup due to the hardware is sufficient to analyze the systems in the asymptotic limit. Evolutionary time scales – comparable to the evolution on earth – can be established. Secondly, due to the hardware resources scaling behavior can be measured. If a system does not scale appropriately it can not be expected to show something different if the system size is doubled (the combinatorics involved in these systems are far beyond any hope to achieve exhaustive search), thus a strong limit on possible biological models is established. Thirdly, physical properties like routing delays and routing of information as such can now be studied *in extenso* and and might help – besides biological applications – in optimizing network traffic or other systems where routing is the limiting constraint.

We succeeded in the construction of an information processing machine with the "interpretation be itself susceptible of expression by means of the symbols which the machine employs. Thus, although it is not itself the being that reflects," [1] we now might hope to build and understand self-reflecting machines.

Appendix: Fitness sharing[1]

Fitness sharing [22] is probably the concept most similar to self-poisoning proposed in this article. Though a detailed comparison is out of scope of this work, some distinguishing properties should be mentioned:

- Fitness sharing is a phenotypic concept. Whether an individual in the population shares the available fitness for its sequence with other individuals of the same or similar type or is only in an indirect manner (via e.g. mutation or crossover) determined by the genotype of this individual.

- With self-poisoning the frequency of instructions or genes throughout the population is counted. The parameter σ_{sh}, see tab. 1, should thus be maximized. This would disrupt the ability of the genetic algorithm to adapt to multi-modal fitness landscapes.

- With self-poisoning only dominant genes are affected. Individuals not using these dominant genes or using them as a genetic pool do not suffer from the consequences of randomization or even better, are perhaps able to utilize the newly generated genetic information. Thus, self-poisoning is a genotypic concept.

[1] Thanks to the two referees who made me aware of this research

Fitness sharing	Self-poisoning
$f_i \equiv f(i)$	$f_i \equiv f(i)$
$sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{sh}}\right)^{\alpha_{sh}} & \text{if } d_{ij} < \sigma_{sh} \\ 0 & \text{otherwise} \end{cases}$	n_{Ij} = number of instructions or genes of type j N = number of instruction or gene types
$m_i = \sum_{j=1}^{N} sh(d_{ij})$	if $(n_{Ii} = \max(n_{Ij}),$ $0 < j < N)$ then {
$f_{sh,i} = \frac{f_i}{m_i}$	$f_i \simeq 0$ in most cases }

Table 1: Fitness sharing [22] versus self-poisoning
The formulas on fitness sharing are taken from [23]. With f_i the original fitness of an individual, d_{ij} the Hamming-distance between two genomes and $f_{sh,i}$ the effective shared fitness of an individual. The two parameters σ_{sh} and α_{sh} determine the behavior of the genetic algorithm applied to a usually multi-modal problem. With $0 < \sigma_{sh} < 1$ and $\alpha_{sh} \neq 0$ only identical sequences are reduced linearly in fitness. To compare with self-poisoning σ_{sh} should be chosen as $\sigma_{sh} = \max(d_{ij})$ with the consequence that only unimodal fitness landscapes can be adapted properly. The main difference between both concepts is the target of fitness reduction: in fitness sharing the total fitness receivable by a sequence (phenotype) is constrained, but with self-poisoning the genotype is affected through randomization of the most abundant instructions or genes in the population of sequences.

Whether the problem of using explicit fitness as in genetic algorithms or implicit fitness used here as in many other artificial life models is of importance has to be evaluated further.

Acknowledgments

The help and support of J. S. M^{c}Caskill, T. Maeke, E. Rambow and the German Ministry of Science (BMBF) is greatly acknowledged. Also many thanks to the referees for their thorough study of this paper.

References

[1] Menabrea L. F. The Analytical Engine: Invented by Charles Babbage *Bibliothèque Universelle de Genève*, No. 82, 1842.

[2] von Neumann J. Theory of Self-Reproducing Automata Burks, A. W. University of Illinois Press, Urbana, 1966.

[3] Dewdney A. K. Computer-Kurzweil *Spektrum der Wissenschaft*, 5:8–12, 1985.

[4] Rasmussen S., Knudsen C., Feldberg R., Hinsholm M. The Coreworld: Emergence and Evolution of Cooperative Structures in a Computational Chemistry *Physica D*, 42:111–134, 1990.

[5] Ray T. S. An Approach to the Synthesis of Life In Langton C. G., Taylor C., Farmer J. D., Rasmussen S., editors, *Artificial Life II*, pages 371–408 Addison-Wesley, New York, 1991.

[6] Adami C., Brown C. T. Evolutionary Learning in the 2D Artificial Life System "Avida" In Brooks R., Maes P., editors, *Artificial Life IV*, pages 377–381 MIT Press, Cambridge, MA, 1994.

[7] Tangen U. The Extension of the Quasi-Species to Functional Evolution PhD Thesis, Jena, 1994.

[8] Codd E. F. Cellular Automata "Academic Press", New York, 1968.

[9] Langton C. G. Artificial Life Addison-Wesley, New York, 1989.

[10] Koza J. R., Benett III F. H., Hutchings J. L., Bade S. L., Keane M. A. "Evolving Computer Programs Using Rapidly Reconfigurable Field-Programmable Gate Arrays and Genetic Programming" In *"Proc. of the 1998 ACM/SIGDA seventh international symposium on Field Programmable Gate Arrays"*, pages 209–219 "ACM", 1998.

[11] Tangen U., Maeke T., M^{c}Caskill J. S. Advanced Simulation in the Configurable Massively Parallel Hardware MereGen In *Caesarium 2000, LNCS* Springer, 2001 In press.

[12] Xilinx Virtex^{TM}-E 1.8V Field Programmable Gate Arrays, DS022 Xilinx Corporation, San Jose, USA, 2000.

[13] Tangen U., Schulte L., M^{c}Caskill J. S. A Parallel Hardware Evolvable Computer POLYP: Extended Abstract *Proceedings of the FCCM'97 in IEEE Symposium*, 1:238–239, 1997.

[14] Tangen U. Self-Organisation in Micro-Configurable Hardware In Bedau M., M^{c}Caskill J. S., Packard N. H., Rasmussen S., editors, *Artificial Life VII*, pages 31–38 The MIT Press, Cambridge, Massachusetts, 2000.

[15] Holland J. H. Escaping Brittleness: the Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems In Michalski R. S., Carbonell J. G., Mitchell T. M., editors, *Machine Learning II*, pages 593–623 Morgan Kaufman, Los Altos, CA, 1986.

[16] Donlin A. Self Modifying Circuitry - A Platform for Tractable Virtual Circuitry *Lect. Not. Comp. Sci.*, 1482:199–208, 1998.

[17] Bedau M. A., Snyder E., Packard N. H. A Classification of Long-Term Evolutionary Dynamics In Adami C., Belew R. K., Kitano H., E. Taylor C., editors, *Artificial Life VI*, pages 228–237 MIT Press, Cambridge, Massachusetts, 1998.

[18] Karlin S., Morris M., Ghandour G., Leung M.-Y. Efficient Algorithms for Molecular Sequence Analysis *Proc. Natl. Acad. Sci. USA*, 85:841–845, 1988.

[19] Kolmogorov A. N. Three Approaches to the Quantitative Definition of Information *Int. J. Comp. Math.*, 2:157–168, 1968.

[20] Eigen M. Selforganization of Matter and the Evolution of Biological Macromolecules *Z. Naturwissenschaften*, 58:465–523, 1971.

[21] McCaskill J. S., Wagler P. From Reconfigurability to Evolution in Construction Systems: Spanning the Electronic, Microfluidic and Biomolecular Domains. In Hartenstein R. W., Grünbacher H., editors, *"FPL 2000, LNCS"*, Vol. 1896, pages 286–299 "Springer, Berlin Heidelberg", 2000.

[22] Goldberg D. E., Richardson J. "Genetic Algorithms with Sharing for Multimodal Function Optimization" In Grefenstette J., editor, *"Proc. of the Second Intern. Conf. on Genetic Algo."*, pages 41–49 "Lawrence Erlbaum Associates, Hillsdale NJ", 1987.

[23] Horn J. The Nature of Niching: Genetic Algorithms and the Evolution of Optimal, Cooperative Populations "Diss., Cornell Uni.", Urbana, Illinois, 1997.