Parallel Architecturing
Starting from Natural Computational Models[1]

Gheorghe Stefan[2]

1. Introduction

The development of efficient parallel computation is now limited because we do not have yet *true parallel architectures.* Nowadays parallel computation involves a bad mixture between parallel structures and parallel algorithms. The effect of a good boundary between the machines and the algorithms lacks. The main reason for this lack is due to the sequential models of computation that ground our architectural approach. *In order to have true parallel architectures we must start from parallel models of computation*. Many natural facts suggest us computational mechanisms. The current computer machines mimic only the external manifestation of the mind computational activity. The computation is an attribute of almost any physical process. Only the suggestions that come from computational aspects of physics can ground efficient parallel structures and real parallel architectures. **Context-driven architectures** are proposed starting from natural models of computation and based on actual parallel structures. Three case studies, about parallel architectures based on molecular and membrane computing, are presented.

2. The Architecturing Process

Structuring, i.e., making of structures is the main goal of current research in computation. **Architecturing**, i.e., defining architectures will be the main way to improve the performance in computation.

2.1 The Definition of Architecture

An architecture has two users because it is an *interface* which implies the ``two faces". The *first user* is the machine designer and the *second user* is the machine user. Therefore, two definitions for the same concept appear.

Definition 1 *An architecture offers of the machine designer (the first user) a set of functions to be implemented without any care about how these functions will be used.*

Definition 2 *An architecture offers of the machine user (the second user) a set of functions and makes transparent the organization and the dimension of the computational structure which performs that set of functions.*

The second definition is important for our approach. John von Neumann architecture and all related architectures are all sequential [von Neumann '45]. Parallel computation is made now without the "protection" offered by a true architecture. The user of a parallel machine must know too many things about the organization and the size of the physical structure. We cannot now hide the physical details to the user, because he uses these details in order to try to improve the efficiency of computation.

The motivation for this impossibility consists in the *sequential models of computation* which ground our computational approach.

2.2 Sequential Models of Computation

The year 1936 was very important for the history of computing. A strange synchronization triggered the occurrence of all basic models of computation:

- Allan Turing proposed his famous machine [Turing '36]
- Alonzo Church proposed the lambda calculus [Church '36]
- Stephan Kleene proposed the use of recursive functions [Kleene '36]
- Emil Post proposed independently a similar model of Turing [Post '36].

All these models are equivalent and have a common feature: they are sequential. The sequentially of the basic models are obvious. The Turing machine is controlled by a finite automaton, lambda calculus requires a progressive evaluation of lambda expressions and the recursive functions are based on the composition rule which requires an ordered evaluation of functions.

The main concept dominating these initial models is the **algorithm**. The computation is a sequentially *controlled* process. The algorithm allows us to construct the result. This common feature and the previously emphasized synchronization must have an explanation.

---

[1] Communication held in 23rd of October 2000 in the 14th Section of the Romanian Academy.

[2] Politehnica University of Bucharest, Dept. of Electronics, Bd. Iuliu Maniu 1-3, Bucharest 6, Romania, **stefan@agni.arh.pub.ro**

2.3 Gödelian Constructive Paradigm

We believe that the triggering signal for the computational events from 1936 was the paradigmatic incompleteness theorem [Gödel '31] proved by Kurt Gödel in 1931. The kernel of the Gödel's proof is based on the mechanism that allows **building** of a correct undecidable form. In this approach originates the idea of computing seen as reaching a result using an **algorithmic construct**.

All the five computational models above mentioned define the computation as an algorithmic process. The result is constructed sequentially under an algorithmic control. Thus, Gödel imposed a **constructive paradigm** with the main consequence of sequentially.

2.4 Mind and Computation

Mind offered the main suggestion for computation. More precisely, the external behavior of mind operating with numbers is now modeled by computation and simulated in computing machines. Peano's arithmetic is the system in which Gödel developed his famous theorem. Indeed, computation can be reduced to the arithmetic of positive integers, but in the same time the computation is limited to remain what can be sequentially constructed using positive integers.

In the same time, we make another enormous mistake believing that the mind is a computational process. We get many advantages from this point of view. The simplicity of the model allows us to obtain a tremendous technological growing in the last half of century. But now we must change the paradigm in order to have the chance to overpass obvious limits.

"A simplified view of the history of computing shows that computing was thought of mainly as mental processes in the 19th century; it is thought of mainly as machine processes in the 20th century, and will be thought of mainly as Nature processes in the 21st century." [Gruska '99]

Computation seems to be the natural process of computation filtered by mind. But there is the chance that the computation is more than that, remaining in the limits of the formal. We believe that the formal was restricted to computational by mind. The extended formal is suggested by the structural diversity of nature which can not be captured in computational forms. Natural forms strict include computational forms, thus offering a chance for an extended paradigm of computation.

2.5 Physics and Computation

Maybe the computation is more general and not limited as a specific mind process. Maybe, the real computation is ***hidden*** behind the superficial behaviors of our mind. The computation as mental process is overloaded with too many "side effects" induced by the communication ``interface" used by the human being in community.

Our opinion is that computation is a natural process distributed on each physical level of reality, including the mental level, but not exclusively at this level. More, the natural process of computation is inefficiently and, *maybe partially*, modeled by mental computation. Many scientists believe that any real process contains a computational aspect, some of them say (exaggerating): "physics is computation". Physics is more than computation, but in its structural part is mainly computation.

Until now we use the theory of computation according with the Church-Turing thesis. We do not exclude the possibility of computation beyond this thesis maintaining the computation as a formal-structural process. Physics can expand the formal computation, removing limits introduced by the obstinate approach performed starting from mind as exclusive site of computation.

3 "Parallel Architectures"

The term ``parallel architecture" is not consistent in the world of computing machinery. In order to be efficient in parallel computing we must renounce to the transparency of the physical structure. Thus, architecture disappears.

A parallel machine consist in a structure having the size in $O(n)$ with constant complexity, $C(n)\ \hat{I}\ O(1)$, and execution time in $O(f(n))$. Engineers imagined many and sophisticated parallel machines, but they didn't find yet an efficient way to use these wonder-working buildings. The reason for this helplessness: the languages used for these machines are too close to the assembly languages.

The main limit comes from the computational models. Let be, for example, only the Kleene's model. The composition rule allows us to understand the limit of parallelism in a computational paradigm that uses this model or the related models. Indeed, for

$$f(x_1, \dots, x_n) = g(h_1(x_1, \dots, x_n), \dots, h_m(x_1, \dots, x_n))$$

the parallelism is limited by the value of m in different stages of computation.

The standard models can be reformulated to ground the parallel approach. For example, there is a **probabilistic variant of Turing machine**, PTM. Each state of the machine can be quitted on many ways, each way having its own probability. The number of states, in which such a machine can be, grows exponentially.

There are two kinds of interpretation for PTM. In the first, a single PTM evolve randomly, according with the probabilities associated with the transitions from each state. In the second, in each cycle the machine is multiplied according to the number of transitions defined from the current state, simulating a non-deterministic computation.

A possible ``real'' structure consists in putting together *n* PTM working in the first variant. The speed of computation increases *O(n)* times, because the control is minimal, but the architecture is not defined at the system level. In this case we have many ``local'' architectures working almost efficient because do not interact.

We maintain in this point the idea of less control which helps us to improve the performance.

4. Natural Models of Computing

The mind is not the single possible suggestion for computation. Gradually, starting from the '60th years, scientists start to understand the power of others **natural mechanisms** in performing computation. The main distinction of these models in comparison with the standard models (in the meantime appears the model of Markov's algorithms [Markov '54]) is the possibility to really ground the parallel computation.

My hope is that starting from these new, natural models, the computation gets supplementary features, remaining in the limit of formal-structural approach. My opinion is that the formal is extended over the Church-Turing computability based on *control algorithm* and *sequentiality*. Hypothesis: the **super-Turing compatibility** can be expected from a paradigm of computation in which the *context allows* many applying of rules belonging to a finite set of rules.

The first problem rising related with the previous hypothesis is how much is expanded the domain of formal over the domain of Turing-computable. A very well known, but very little disseminated fact is that the space of the formal is huge in comparison with the space of Turing-computable (see **Appendix**). Thus, there is space for expanding the computable! One way in this respect is to understand the lesson of Nature regarding the parallelism.

4.1 Lindenmayer Grammars

A Lindenmayer grammar consists mainly in a set of productions parallel applied starting from an initial symbol. In each stage all the rules which can be applied are applied. This approach is one of the first in which the parallel actions are not restricted. The degree of parallelism is similar with the natural process of cellular growing, because these grammars are designed to model this natural process. Applying the rule is *driven by the context*. The *control is minimal* because all can be done is done. At each step important is the stage reached by the symbolic structure in the evolution of the generating process. The actual content of the symbolic structure triggers the rules to be applied. Thus, *the control is almost completely substituted by a context driven process*.

4.2 Genetic Algorithms

Genetic algorithms represent a style of designing algorithms when the space in which we must find the solution is too large. Instead of a systematic, very controlled, process of searching, there is proposed an apparent chaotic way of running through the space of solutions. "Crossing" conveniently possible solutions and adding an appropriate number of random mutations the searching process avoids the exhaustive crossing of all possible solutions, offering a better solution, eventually the best.

The genetic approach is very suitable for parallel computation, because many *independent* computations can be started simultaneously. The searching process can be developed *independently* in many points in the same time and the chance to find a good solution grows almost proportionally. Because there is no necessary a global control of the process, genetic algorithms offer a very interesting suggestion for parallel architecturing.

4.3 Molecular Computing

After the Adleman experiment [Adleman '94], the molecular computing, a domain initiated starting from '70 years, offers a very promising basic theoretical result: a new computational model was established [Paun '95] based on the *splicing mechanism* [Head '94]. The splicing mechanism consists in identifying in the elements of a set of strings substrings that define cutting points, in cutting the strings in identified points and after that in recombining the obtained substrings in order to form new strings.

Because in a "soup" of DNA molecules we can have a huge number of ``strings'', a big number of splicing can *independently* performed in a short time interval. The parallelism of this approach is obvious.

This model can be used also in defining "silicon splicing machine" [Stefan '97], [Stefan '98].

4.4 Membrane Computation

Recently, a new proposal holds the attention of many researchers: the *membrane computation* [Paun '99]. Starting also from a natural suggestion this kind of computation offers the possibility to involve in the

computational process some chaotic aspects starting from the ``swimming effects" of objects inside a membrane. The set of rules defined over a *super-set* are applied also depending on a evolving context until no rule can be applied. The lack of any direct control offers maybe super-Turing features of this kind of computation.

4.5 Quantum Computing

The quantum computation is only a theoretical approach, but it is very useful in the discussion about the relation between the naturally founded computation and the mind founded computation. Mind filtered the natural computation offering the Church-Turing based computation. The quantum computation offers a more complex environment for computing. The difference can be easy proved using a PTM and a *quantum Turing machine* (QTM).

The maximum power of a PTM is reached when it is used for modeling a *non-deterministic* computation. A QTM achieve the same degree of parallelism but has in addition possibilities offered by the play between the *positive interferences* and the *negative interferences* of the complex aptitudes, $a_1, \ldots, a_k$, at each level of computation.

Because "the nature does not make jumps" we must avoid to referring to two types of worlds, a classic world and/or a quantum world. All we can do is to use two type of languages. One is restricted to the Church-Turing computability and another, more actual, is the language of quantum computing. The first leads us to the actual sequential *architecture* and is seducing us with parallel *structure*. The second offers us the possibility of a parallel architecture.

5 Context-Driven Architectures

A possible way of architecturing in the context offered by natural models of computation is to use the context to drive the computational process. The conventional computation uses the more restricted mechanism of control. There are proposal of true parallel architectures starting from the conventional models of computation (reduction architecture and data-flow architecture), but they are strictly Turing compatible. We hope that starting from natural models of computing to be able to find *super-Turing architectures.*

One of the main features, common to almost all natural models of computation is the *context-driven mechanism* used to "control" the computation. This mechanism can be found also in sequential models of computation. The *Markov algorithms* use many *sequential* applications of a finite set of rules, depending on the content of the string to be processed. But in each step only one production can be applied.

The Lindenmayer grammars allow any number of rules to be applied in each step in many places of the previously generated structure. In molecular computing many splicings are performed simultaneously the soup of DNA molecules. Each super-set offers the context of many productions to be applied in the model of membrane computation. In all these cases, and maybe in others, the context selects the actions to be performed, so as a part of a finite set of rules are applied in all the place where the application is possible. Thus, the parallelism is extended in many places according to local decisions being not imposed by a centralized control.

There are mainly two way to use these new Nature inspired models. One is to find specific technologies for each of them and another is to try to use now a day Silicon technology and appropriate intermediate models. The second will be illustrated in the next section.

6. Case Studies

Three case studies will be shortly presented. All the three are experiments on the way to define parallel architectures. The architecturing process intends also to disclose *super-Turing* features of computation in the true parallel variants.

6.1 Molecular computing on Connex Memory

The molecular computing is a challenge also for the Silicon based machines. An interesting experiment was done using as physical support for the Connex memory (CM) [Stefan '98a]. The facility to find a string in time related with the size of string, offered by the CM, allowed the performing a *splicing operation* in constant time, independently of the number of strings involved in computation. It was a good result, because in the actual DNA computation a splicing operation is performed in time $O(n^{1/3})$ due to the spatial distribution of molecules.

6.2 The Eco-Chip and Molecular Computing

In the actual ``in vitro" systems are performed many splicings in the same time. In order to catch this aspect, the functions of the CM were spread over the area of a *bi-dimensional cellular automaton*. Results the, so called, **Eco-chip** (EC) [Stefan '98]. Using as physical support the EC, many splicings can be done in the same time, each being performed in time $O(n^{1/2})$. Using a simulator an "in info" experiment was done. Also, the Adleman's experiment was re-made [Stefan '98] using the same simulator.

Because the size of EC is in $O(n)$ and the complexity is in $O(1)$, the circuit is obtainable in current technologies. In this case, based on computational

model and a realizable structure we can start to define a parallel architecture.

6.3 Chaotic Membrane Computing on Cellular Automata

The promising model of *membrane computation* can also ground a parallel architecture, based on the structural support offered by the concept of *cellular automaton* and on the framework of the *chaotic* understanding of the *self-organizing processes* [Stefan '00].

A super-cell system can be represented on the "surface" of a cellular automaton. A membrane is a closed string of cells, having the same value, that closes an area of cells and, many of the enclosed cells have values belonging to a super-set. The dimension and the position of the membrane fluctuate randomly. Also, the elements of a super-set inside of a membrane, have a chaotic motion. If the elements of the super-set meet each other in this chaotic motion and if there is an appropriate rule, then the two elements are replaced according to the rule. Thus, this approach catches also the chaotic "swimming effect" of the process in a certain membrane of an actual cell. The simulation of this mechanism is under development.

Based on the chaotic interaction between the membrane and the elements it contains are expected "super-Turing" formal effects.

7 Final Comments}

This communication is a preliminary presentation of the intended future research activity plan of the Center for New Electronics Architectures. The main idea is that parallel computation can be, maybe, more than computation established by Church-Turing thesis.
In this case, a parallel architecture is more than a better way to make computation. A parallel architecture offers also a way to deeply understand physics and reality. It is another improved way to make science, to enlarge the domain of the formal in order to reach the boundary with the phenomenon and with the phenomenological approach.

Bibliography

[Adleman '94] Adleman, L. M.: "Molecular Computation of Solutions to Combinatorial Problems", *Science*, 226 (Nov. 1994).

[Church '36] Alonzo Church: "An Unsolvable Problem of Elementary Number Theory", in *American Journal of Mathematics*, vol. 58 345-363, 1936.

[Gödels '31] Kurt Gödel: "On Formally Decidable Propositions of \ Principia Mathematica and Related Systems I", in S. Fefermann et all.: *Collected Works I: Publications 1929 - 1936*, Oxford Univ. Press, New York, 1986.

[Gruska '99] Jozef Gruska: *Quantum Computing*, McGraw-Hill, 1999.

[Head '92] Head, T.: "Splicing Schemes and DNA", *Lindenmayer Systems: Impacts on Theoretical Computer Science and Developmental Biology* (G. Rozenberg, A. Salomaa, eds.), Springer-Verlag, Berlin, 1992, pp. 371 - 383.

[Kleene '36] Stephen C. Kleene: "General Recursive Functions of Natural Numbers", in *Math. Ann.*, 112, 1936.

[Lindenmayer '68] Lindenmayer, A.: "Mathematical Models of Cellular Interactions in Development I, II", *Journal of Theor. Biology*, 18, 1968.

[Markov '54] Markov, A. A.: "The Theory of Algorithms", *Trudy Matem. Instituta im V. A. Steklova*}, vol. 42, 1954. (Translated from Russian by J. J. Schorr-kon, U.S. Dept. of Commerce, Office of Technical Services, no. OTS 60-51085, 1954)

**[P`un '95]** P`un, G. : "On the Power of the Splicing Operation", in *Intern. J. Computer Math.*, Vol. 59, pp 27-35, 1995.

[Paun '99] Gh. Paun: "Computing with membranes. An introduction", *Bulletin of EATCS*, 68 Feb. 1999, pp. 139-152.

[Post '36] Emil Post: "Finite Combinatory Processes. Formulation I", in *The Journal of Symbolic Logic*, vol.1, p. 103 -105, 1936.

[Stefan '96] Stefan, G., Malita, M. : "Chaitin's Toy-Lisp on Connex Memory Machine", *Journal of Universal Computer Science*, vol. 2, no. 5, 1996, pp. 410-426.

[Stefan '97] Stefan, G., Malita, M. : "The Splicing Mechanism and the Connex Memory", *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, Indianapolis, April 13 - 16, 1997. p. 225-229.

[Stefan '98] Stefan, G., Benea, R.: "Connex Memories & Rewriting Systems", in *MELECON '98*, Tel-Aviv, May 18 -20, 1998.

[Stefan '98a] Stefan, G.: "The Connex Memory: A Physical Support for Tree / List Processing" in *The Romanian Journal of Information Science and Technology*, Vol., Number 1, 1989, p. 85 - 104.

[Stefan '98] Stefan, G.: "Silicon or Molecules? What's the Best for Splicing", in Gh. P`un (ed.): *Computing with Bio-Molecules. Theory and Experiments*, Springer, 1998. p. 158-181.

[Stefan '00] Stefan, G.: "Chaotic Membrane Computation with Cellular Automata", in progress.

[Turing '36] Alan M. Turing: ``On Computable Numbers with an Application to the Eintscheidungsproblem", in *Proc. London Mathematical Society*, 42 (1936), 43 (1937).

[von Neumann '45] John von Neumann: "First Draft of a Report on the EDVAC", reprinted *in IEEE Annals of the History of Computing*, Vol. 5, No. 4, 1993.

Appendix: Formal / Turing-computable

Any function g: $\mathbf{N}^n \to \mathbf{N}$ having the form

$$g(x_1, \dots, x_n)$$

with $x_i \in \mathbf{N}$ for $i = 1, \dots, p$, can be expressed as

$$f(x_1, \dots, x_n)$$

where: $f:\{0,1\}^n \to \{0,1\}$, with $x_i \in \{0,1\}$ for $i = 1, \dots, n$. Thus any function g, has a Boolean form f.

Definition 3 *A function $f(x_1, \dots, x_n)$ is a formal function if the* **form** *of f can be defined using a string of symbols $d_1, \dots, d_q$.*

Theorem 1 *If the definition of any formal function $f(x_1, \dots, x_n)$ is given by the binary string $d_1, \dots, d_q$ with with $x_i \in \{0,1\}$ for $i = 1, \dots, m$, then the length of definition, m, has the value: $m = 2^n$.*

Proof The circuit for the Boolean function f can be described by:

$$f(x_1, \dots, x_n) = x_n f_1(x_1, \dots, x_{n-1}) + x_n' f_0(x_1, \dots, x_{n-1})$$

the resulting n-input circuit containing an elementary multiplexer, EMUX, and two circuits with $n-1$ inputs, one for $f_1(x_1, \dots, x_{n-1})$ and one for $f_0(x_1, \dots, x_{n-1})$.
EMUX is an elementary selector circuit described by the following equation:

$$out = sel\ input\_1 + sel'\ input\_0$$

the output *out* takes the value of the selected input (*input_1* or *input_0*) selected by the value of the selection input *sel*.
In the next step, the two $(n-1)$-input circuits are each similarly described with an EMUX and two $(n-2)$-input circuits. The same is the procedure applied to the four resulted circuits and so on until the ending step in which occur 2^{n-1} EMUXs, each having connected on the selected inputs 2^n only two kind of functions, the functions of zero variables: the values 0 and 1.
The final circuit is a binary tree of n levels, containing 2^n-1 EMUXs and having connected to the 2^n inputs of the last level of EMUXs a 2^n-bit binary configuration. Shortly, we have a simple, uniform circuit (the tree of EMUXs, the same for any Boolean function of n variable) and a binary configuration of 2^n bits that personalizes the circuit for a certain f.
It is obvious that for any $n \in \mathbf{N}$, the function f has associated a *family of circuits.*

Attention: *families of circuits do more than Turing machines!* Of course, because they are random (defined) parallel structure. But, unfortunately they are, in the same time, too complex. The solution to the problem of how to deal with this kind of complexity could be the **actual** self-organizing processes.

Theorem 2 *The algorithmic complexity of a formal function $f(x_1, \dots, x_n)$, is in $O(m)$.*

Proof The program which describes the string $d_1 \dots d_m$ has the form

$$p\ d_1 \dots d_m$$

where p is a constant length preamble syntactically imposed. Therefore, the *maximum length* of program for a certain machine is in $O(m)$.

Theorem 3 *If the function $f(x_1, \dots, x_n)$ is Turing-computable, then it has the algorithmic complexity in $O(1)$.*

Proof A Turing machine accepts only functions having constant descriptions, in $O(1)$ (independent of n), because the control automaton is a finite automaton.

The main question is: what is the weight of *Turing-computable functions* in the set of *formal functions*?

Theorem 4 *The weight, w, of Turing-computable functions, of n binary variables, in the set of formal functions decreases exponentially with n.*

Proof Let be a given n. The number of formal n-input function is $N = 2^p$, with $p = 2^m$, because the definition are expressed with 2^n bits. Some of these functions are Turing-computable. These functions can be defined by compressed m-bit strings. The value of m depends on the actual function, but is realized the condition that $max(m) < n$ and m does not depends by n. Each compressed form of m bits corresponds only to one n-bit uncompressed form. Thus, the ratio between the Turing-computable function of and the formal function, both of n variables, is smaller than

$$max(w) = 2^{-(n-max(m))}.$$

And, because $max(m)$ does not depends by n, the ratio has the same form for no matter how big becomes n. Results:

$$max(w) = const/2^n.$$

If we say that almost all the formal is not computable we are wrong in a very small extent. It is a miracle that the computable is useful in this condition. A strange, to be studied, "mechanism" maybe leads the human being to be efficient using only the effective, Turing-computable functions.

What is the reason for which we do not use family of circuits for computing functions? Because this approach is *complex* and *oversized*. Most (tremendous most) of strings used for defining functions are algorithmic random, i.e., useless for building universal machines. Thus, we remain still limited to the *simple* functions performed by Turing machine under the control of finite automata. The option for the simplicity is awarded by the spectacular development of computers but is punished by the simplicity of problems resolved by these, too promising, machines.