Let's consider Moore's law in its entirety

Gheorghe M. Ștefan
Politehnica University of Bucharest
E-mail: gheorghe.stefan@upb.ro

Abstract—Gordon Moore's predictions refer, in addition to the resolution at which we can make the devices, also to the size of the chips and to the "circuit cleverness". Considering Moore's law in all its aspects offers a more complex perspective on the current evolution of the field of integrated circuits. Computers, as a General-Purpose Technology, tend to be replaced by various types of accelerators. A possible solution that we present is based on parallelism supported by the recursive abstract model that we propose starting from the model of partially recursive functions of Stephen Kleene.

Keywords — Moore's Law; intense computation; accelerators; parallelism; mathematical model of parallel computing; abstract model for parallelism.

1. Introduction

Gordon Moore talks twice about the evolution of integrated circuit technology. In addition to the internal report from 1965 [1], the paper from 1975 [2] provides important new insights. If the first intervention is mainly about the density of components (see Fig. 2) and the efficiency with which the integrated circuits were produced (see Fig. 1), in the 1975 paper are explicitly added two other dimensions for evaluating the evolution of the domain of integrated circuits.

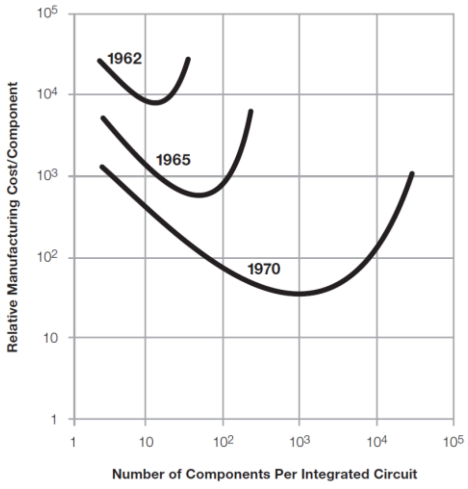

Fig. 1 Cost vs. number of components per IC [1].

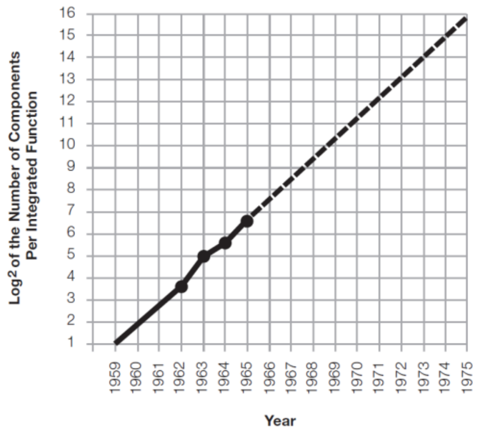

Fig.2 Number of components per IC [1].

In Figure 3 all the three factors considered:

Factor1: density (device size)
Factor2: chip size
Factor3: "circuit cleverness"

are represented cumulatively. To the factor given by density, are added the size of the chip and "circuit cleverness".

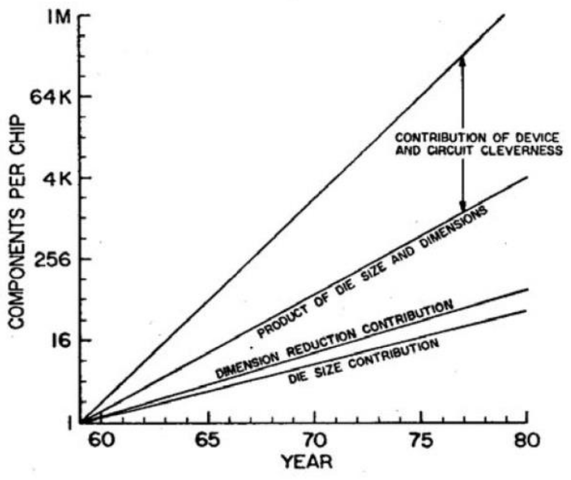

Fig. 3 Components per chip depending on the three factors: device size, chip size and "circuit cleverness" [2].

"Circuit cleverness" translates, in what follows in this paper, by organization and architecture appropriate to functions that integrated circuit must perform. Besides, density & size given by technology, and

organization & architecture we can add few other factors involved, such as algorithms, languages, compilers and, why not, hybrid approaches involving analog, bio- and quantum-computing.
In the following we focus only on how organizational and architectural aspects influence mainly **Factor3** *– "**circuit cleverness**" – and subsidiary* **Factor2** – chip ***sizing***.

2. Distinctions Imposed by the Exponential Growth

In theoretical computer science, the complexity of circuits is treated using terminology that creates confusion that we can no longer accept when the size of the circuits exceeds a certain limit. Terms such as *circuit-size complexity* are unacceptable when notable differences between ***size*** and ***complexity*** are required. We are obliged to explicitly nuance the two concepts when we use them to speak today distinctly of the size and complexity of circuits. The same for the intensity and complexity of computing. The dimensions of the circuits and the intensity of the computation reach very high values today, so high that their identification with the complexity become confusing.
The conceptual framework for these distinctions is given by the *algorithmic complexity* introduced independently in the 1960s years by Gregory Chaitin [3], Andrei N. Kolmogoroff [4], Ray Solomonoff [5].

A. Size vs. Complexity of Circuits

In the case of circuits, we will have to distinguish between their size and their complexity.

Definition 1: The size of a *n*-input circuit, *S(n)*, is given by the number of components or the area it occupies on silicon.

Definition 2: The complexity of a *n*-input circuit, *C(n)*, is proportional to the size of the shortest description, expressed in number of characters, that defines it.

For example, the complexity of a circuit can be characterized by the length of the shortest program written in the Hardware Description Language that describes it.

Definition 3: A circuit is complex if its size and complexity are in the same complexity class: $S(n) \sim C(n)$.

Conjecture: As the size of a circuit increases exponentially, its complexity is limited to a logarithmic increase.

Corollary: If the size of a digital system increases exponentially and its complexity is limited to increase logarithmically, then the system must be programmable to justify increasing its size.

Because the dynamics of complexity cannot keep pace with the dynamics of size, we will have to consider the functional dynamics offered by the programmability of the structure deployed on an integrated circuit. Functional complexity will not be able to increase significantly unless the circuit function can be specified by a program running on a much simpler circuit structure (memories tightly interleaved with execution units). Thus, the incompressible bit structure of programs will give the functional complexity.

B. Intensity vs. Complexity of Programs

In the case of programs, we will have to distinguish between their intensity and their complexity.

Definition 4: The intensity of a program is given by the number of clock cycles required for its execution.

Definition 5: The complexity of a program is proportional to the size, expressed in number of characters, of the simplest description that defines it.

Definition 6: A program is complex if its intensity and complexity are in the same complexity class.

Definition 7: A program is intense if its complexity is much smaller than its intensity.

3. Organization & Architecture on Big Sized Dies

The way in which the "circuit cleverness" and die size evolves is dictated equally by the organization and architecture of the system that grows exponentially in size. Both organization and architecture must allow the acquisition of functional facilities to justify the increase of both chip size and component density. How can the exponential increase of the number of components per chip be capitalized in the conditions in which the accepted increase of the complexity of the structure is much slower (according to our point of view it is logarithmic)? Only a programmable cellular structure in which the data is interleaved with the execution elements represents the solution.

A. Organizations

The evolutions from the last years on the computer products market highlight more and more the distinction between complex and intense computing, in conditions in which the consumed energy is an essential parameter. Under these conditions, the ***organizations*** of the computing systems migrate toward ***heterogeneous computing*** as a solution that is required in the form of a **host** system, responsible for complex computing, and an associated **accelerator** that runs intense computing (see Figure 4).

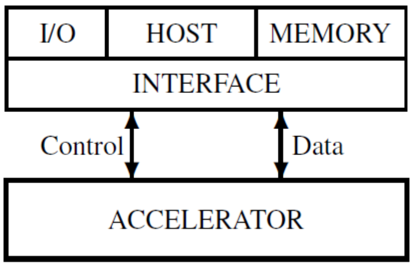

Fig. 4 Heterogenous computing system, where HOST runs the complex computation while ACCELERATOR runs the intense computation.

The same trend is supported by the decline of computer technology as General-Purpose Technology (GPT) [6]. The accelerator has a specialized rather than a general structure, and its share in the size of the computer system tends to be dominant. Why wouldn't it be possible for us to move to a stage where accelerator technology is becoming a GPT?

The host is implemented using a mono or multi (2÷8) core system, while for accelerators we can consider several parallel solutions:

1. FPGA-based reconfigurable circuits (the genuine, natural parallel approach) able to take over the intense parts of the program
2. FPGA-based pseudo-reconfigurable systems implemented as parametrizable & configurable programmable systems
3. Parallel programmable systems (“artificial parallelism” provided by many-core (>64) computing systems)

Each solution comes with its advantages and disadvantages.

The first offers specific solutions that can be very efficient (in terms of execution time and energy), but it is difficult to generalize, due to the fact that it requires high-performance digital designers and the number of circuit designers is much smaller than that of programmers. On the other hand, the solution that involves using compilers that translate from high-level languages into Hardware Description Languages is far from providing efficient translations in all cases.

The second solution is general in nature, but the limitations are due to the use of FPGA technology which limits the speed of execution and the possibility of reducing energy consumption. The good news is that more and more frequently used modules (such as DSPs, BlockRAMs and UltraRAMs) are being implemented in this technology as ASICs. When such modules are usable, the performance gap compared to ASIC implementations is significantly reduced.

The third solution should provide the most efficient solution (in terms of speed performance and power consumption), but it does so very rarely. Why? Because, as we will show, parallelism has entered computer science through the back door.

Can we coherently answer the following question: *what does parallel calculation mean, at least at the level of a silicon die*? Partially yes: a Boolean circuit represents the

natural parallel implementation of a function defined in $\{0,1\}^n$ with values in $\{0,1\}^m$. Much more difficult is the case of the ***artificial implementation of parallelism*** in the form of programmable cellular computational structures.

B. Architectures

From an architectural point of view, the two components of a heterogeneous computing system present to the user completely different images. We discuss the second and third solutions proposed for the accelerator, solutions that involve programmable structures.

While the host is programmed in a high-level language that provides the compiler with a conventional set of instructions, the accelerator is seen as a hardware-implemented ***kernel library*** of functions. Kernel, because the accelerator is a finite structure that allows the acceleration of functions defined on data structures limited by its size. Thus, the kernel library is used to accelerate, through programs written in a high-level language, a function ***library*** for extended data structures to the level required by real applications.

4. "Cleverness of Organization" and Big Sized Chip Management

Let us consider, in the following, Moore's "circuit cleverness " under the more general formulation of "organization cleverness". And, if we consider the issue of chip size, we will converge to similar requirements.

Thus, to get noticeable accelerations, a "clever organization" requests a ***cellular*** structure of a programmable accelerator that performs parallel computation. Cellular, to keep the complexity under control and programmable, to get functional complexity.

In the same time, the curse of larger die size and poor yields can be addressed by self-repair replaces/bypasses techniques which are facilitated by repetitive ***cellular*** structures. It should be noted that these techniques are most easily applied currently to memory structures. This brings to our attention the same *programmable & cellular* structures.

But what means parallel computation (at least) at the level of a silicon die? The histories of mono-core and many-core computing paradigms are radically different. While the first is rigorously substantiated, the second is presented with an evolution distorted by an unfortunate interaction between academic research and developments imposed by the corporate space.

A. Mono-Core vs. Many-Core Evolution

Let us briefly present a comparison between the history of mono-core computing and that of the parallel computing. While for the mono-core computing the following steps have been completed:

1. 1931: Kurt Gödel [7] proved that the decision problem, formulated by David Hilbert, does not have a logical solution in Peano's arithmetic
2. 1935: Alonzo Church [8], Stephen Kleene [9], Emil Post [10], Alan M. Turing [11] published independently their mathematical version of Gödel's incompleteness theorem, thus providing four independent, but equivalent ***mathematical models*** for computing as a mechanism based on logic decision
3. 1937: Claude E. Shannon [12] defended at MIT his master thesis which became the theoretical foundation for designing and implementing practical digital circuits.

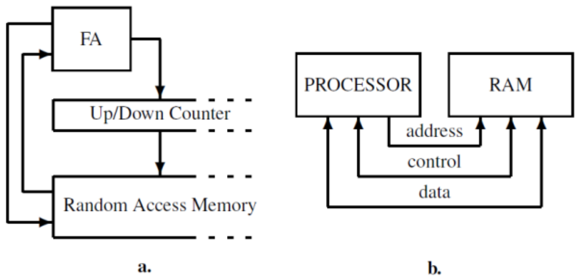

Fig. 5 From the mathematical model of mono-cell computation to the abstract model. **a.** Turing Machine. **b.** John von Neumann abstract model.

4. 1946: John von Neumann [13], based on Turing's approach and on the design and

implementation made by John Mauchly and J. Presper Eckert for ENIAC, provided the ***abstract model*** (see Figure 5) for the mono-core computing machine (his approach is paralleled by the *Harvard* version)

5. 1953: IBM announced the first mass-produced computer – IBM 650
6. 1954: John Backus made the draft specification for the first high level programming language – FORTRAN
7. 1964: the term of ***computer architecture*** is used "*to describe the attributes of a computing system as seen by the programmer"* [14], is introduced when specifying the IBM 360 computer

for the multi- or many-core parallel computing, the sequence of meaningful events had the following distorted and incoherent evolution:

1. 1962: ***manufacturing in quantity***; the first symmetrical MIMD parallel engine is introduced on the computer market by Burroughs Corporation
2. 1965-75: ***architectural issues***; Edsger W. Dijkstra formulates, starting with [15], the first concerns about specific parallel programming issues (such as, the critical regions problem, semaphores, the dining philosophers problem)
3. 1974-82: ***abstract machine models***; proposals of the first abstract models (bit vector models in [16] and PRAM models (see Figure 6) in [17,18]) start to appear after almost two decades of non-systematic experiments (started in the late 1950s) and the too early market production

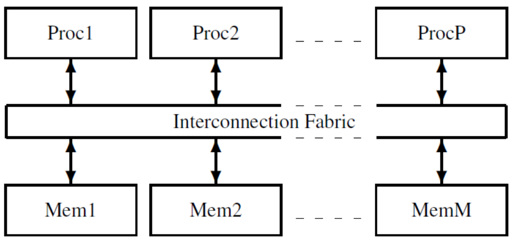

Fig. 6 Parallel Random-Access Machine (PRAM) is an abstract machine model for parallel computing currently confused with a mathematical model for parallel computing. It is not theoretically supported by any mathematical model.

4. ?: ***mathematical parallel computation model***; no one yet really considered it as mandatory, regrettably confusing it with abstract machine models, although it is there and waits to be considered (see the Kleene's mathematical model for computation [9]).

Now, in the 3rd decade of the 3rd millennium, after more than half century of chaotic development, it is obvious that *the history of parallel computing is distorted by missing stages and uncorrelated evolutions* [19].

What is the consequence of the distorted history of parallel computing? *Ad hoc* organizations of multi- or many-core structures fail to provide accelerations close to the peak performance they could theoretically achieve. For example, the oxymoronic GPGPU (General Purpose Graphic Processing Units), proves to be particularly efficient in graphic applications, but very rarely manages to perform close to its very high peak performance when used as general-purpose accelerator.

Table 1. Comparing GPU with CPU

	Intel Xeon E5-2690v4	**Nvidia P100**
Technology	14 nm	16 nm
Power	135 Watt	250 Watt
Clock	~2.8 GHz	1.1 GHz
Threads	28	3584
Bandwidth	76.8 GB/s	732 GB/s
Access to L1	5-12 cycles	80 cycles
Price	$2090	$2500

Let's compare the performance offered by an Intel processor (Xeon E5-2690v4) and an Nvidia graphics accelerator (P100). For the engines exemplified in Table 1, the expected acceleration provided by the GPU for a code running on the CPU, considering the ratio between the threads executed in each engine and the ratio of the running frequencies, is:

$$acc = (3584/28)\times(1.1/2.8) = 50.28$$

Why for most applications the actual acceleration is <10? Because there are some inconsistencies in the way the organization and architecture of the Nvidia chip *as general-purpose accelerator* are defined. For example:

1. while the computational power of GPU is ~50× the computational power of CPU, the bandwidth is only ~10× larger, thus increasing the effect of the *von Neumann Bottleneck*
2. the access to Level 1 Cache requests ~10× more clock cycles for GPU
3. although the data flow in an accelerator is very predictable, GPUs use cache memories instead of program-controlled buffers
4. the structural organization of a GPU considers first the criteria of geometric organization of the space on silicon and only secondly the theoretical aspects (if any) of computations
5. there is no good balance in GPUs between computing resources, which are oversized, and the possibilities of communication between cells.

How should the structure (organization) of an efficient many-cell programmable accelerator look like? To answer this question, we must start, as in the case of mono-cell computing, from a mathematical model, which this time will have to be dedicated for parallel computation.

B. A Recursive Abstract Model for Parallel Computation

Even if the literature does not consider it as such, there is a mathematical model for parallel computation that is waiting to be considered. This is the Partially Recursive Function Model [9] proposed by Stephen Kleene independently and in the same year as the Turing Model, with which it is mathematically equivalent.

Because the only independent rule in Kleene's model is the *composition rule* [20], the conception of an abstract model for parallelism starting from the definition of partial recursion, just as von Neumann's abstract model started from Turing's mathematical model, considers only the composition rule:

$$f(X) = g(h_1(X), \ldots h_p(X))$$

whose representation in the form of a circuit is given in Figure 7. Starting from this representation, we highlight two theoretical forms of parallelism:

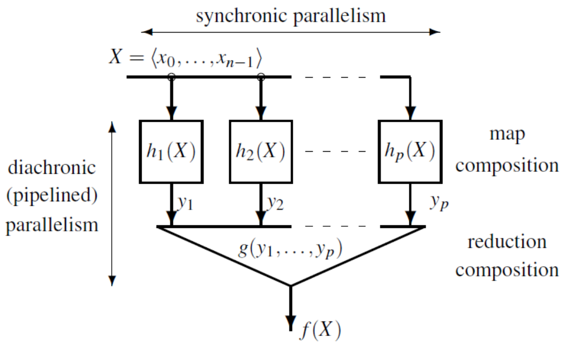

Fig. 7 The circuit version of the composition rule.

- *synchronic parallelism*, in the form of MAP composition, a linear array of cells
- *diachronic (pipeline) parallelism*, due to the pipeline connection between MAP array and a *log*-dept REDUCE network.

Kleene's mathematical model leads to the abstract model for parallel computing [19], just as Turing's mathematical model led to the von Neumann's abstract model (or Harvard model).

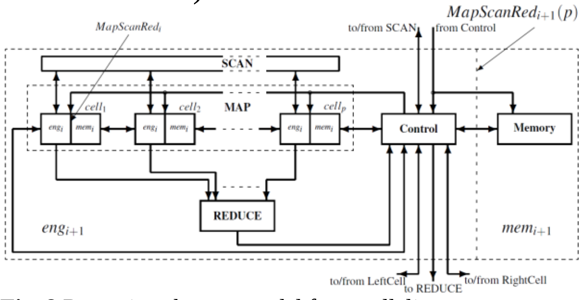

Fig. 8 Recursive abstract model for parallelism.

Figure 8 shows the abstract model resulting from Kleene's mathematical model. This model is a recursive one. Indeed, the *(i+1)*-th level in the recursive organization

$$MapScanRed_{i+1}(p)$$

is divided in two parts: *eng_{i+1}* and *mem_{i+1}* and contains in the MAP array *p* linearly connected cells each containing an *eng_i* and a *mem_i*. The *p* cells of the MAP array are thought of as having the same organization as *$MapScanRed_{i+1}(p)$*: an *eng* module, with a MapScanRed system loop connected with a Control unit, and a *mem* section. Thus, we obtain a hierarchically organized cellular structure where memory resources are tightly interleaved with execution elements. Only level 1 in the hierarchy – *$MapScanRed_1(p)$* – is build from *elementary cells*, for which:

- *p=64÷4096,* in currently used technological nodes or in FPGA implementations
- *eng_0*: *8÷64*-bit execution unit
- *mem_0*: *4÷16* KB SRAM

- REDUCE: is a *log*-depth tree network performing reduction for functions as *add, min, max*, …
- SCAN: is a *log*-depth network performing functions such as *permute, prefix*, …
- Control: is a standard processing mono-core with *eng+mem+progMem*
- *progMem*: provides in each cycle an instruction for Control and another to be executed, with a *O(log p)* latency, in each active cell of the MAP array.

For example, in Figure 9 is represented a two-level MapScanRed structure with 4 cells, where each cell is a MapScanRed structure with 256 elementary cells. Thus, the structure contains 1024 hierarchically organized execution elements on two levels.

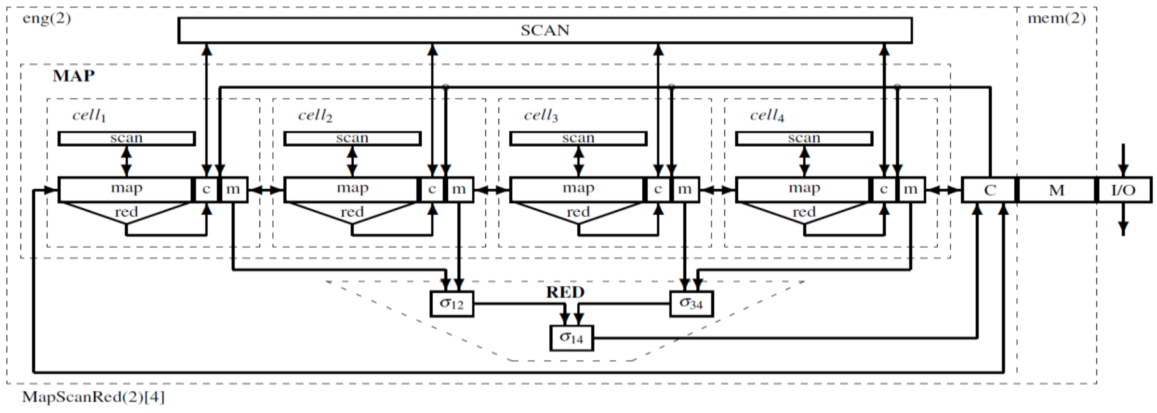

Fig. 9 $MapScanRed_2(4)$ – a two-level hierarchy with 4 cells – with $MapScanRed_1(256)$ cells – an one-level system in the recursive hierarchy with 256 elementary cells.

5. Concluding Remarks

In the last half century, **Factor1** has received almost exclusively the attention of the community involved in the development of integrated circuit technology. Now, when the effect of this factor seems to be exhausted, I think we need to pay more attention to the other two factors. Their effect was not considered with sufficient attention, especially the effect of the third, "circuit cleverness". *Ad hoc* structural developments accompanied by uninspired architectural approaches have led to inefficient computational accelerators both in terms of the use of the computing capabilities deployed on silicon and in terms of energy consumption.

While **Factor2** is addressed quite well by the modularity of large circuits (mainly memories), for **Factor3** we believe that major reformulations need to be considered. Because big-sized circuits cannot achieve complexities comparable to their size, programmable cellular structures are required in which the execution elements are interspersed with memory blocks. Thus, large accelerators naturally become parallel computing structures.

Because *off-the-shelf* accelerators are general *ad hoc* structures or dedicated structures, they cannot be used efficiently for parallel computing. Consequently, we propose an abstract model for one-chip parallel computing based on the mathematical model of partial recursive functions proposed in 1936 by Stephen Kleene.

The proposed model is a recursive one that allows a hierarchical cellular organization that attenuates, by distributing the execution elements to the appropriately dimensioned memory modules, the *von Neumann Bottleneck* effect.

Thus, even if the density of the devices will not increase significantly, we have the hope offered by a good use of them in designing "clever organizations" to which we associate "clever architectures" to allow the action of **Factor3** designated by Gordon Moore with "circuit cleverness".

References

[1] G. Moore: "The Future of Integrated Electronics." *Fairchild Semiconductor* internal publication (1964).

[2] G. Moore: "Progress in Digital Integrated *Electronics*", *Technical Digest 1975. International Electron Devices Meeting, IEEE*, pp. 11-13, 1975.

[3] G. Chaitin: "On the Length of Programs for Computing Binary Sequences: Statistical Considerations", *J. of the ACM*, **16**(1):145-159, 1966.

[4] A.A. Kolmogorov: "Three Approaches to the Definition of the Concept "Quantity of Information", in *Probl. Peredachi Inform.*, **1**(1):3-11, 1965.

[5] R. J. Solomonoff: "A Formal Theory of Inductive Inference", in *Information and Control*, **7**(1):1-22, and **7**(2):224-254, 1964.

[6] N. C. Thompson, S. Spanuth: "The Decline of Computers as a General Purpose Technology", in *Communication of the ACM*, **64(**3):64-72, 2021.

[7] K. Gödel, "Uber formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme", (On Formally Undecidable Propositions of Principia Mathematica and Related Systems), *Monatshefte für Mathematik und Physik* **38**:173-198, 1931.

[8] A. Church, "An unsolvable problem of elementary number theory", *American Journal of Mathematics* **58**(2):345-363, 1936.

[9] S. Kleene, "General recursive functions of natural numbers", *Mathematische Annalen* **112**(5):727-742, 1936.

[10] E. Post, "Finite Combinatory Processes - Formulation 1", *Journal of Symbolic Logic* **1**(3):103-105, 1936.

[11] A. Turing, "On computable numbers with an application to the Eintscheidungsproblem", *Proceedings of the London Mathematical Society*, **42**(1):230-256, 1936 and a correction in **43**(6):544-546, 1937.

[12] C. Shannon, *A Symbolic Analysis of Relay and Switching Circuits*, Master Thesis at MIT, 1937. Republished in *Transactions of the American Institute of Electrical Engineers*, **57**(12):713-723, 1938.

[13] J. von Neumann, *First Draft of a Report on the EDVAC*, Moore School of Electrical Engineering, University of Pennsylvania, June 30, 1945. Republished in *IEEE Annals of the History of Computing*, **15**(4):27-75, 1993

[14] G. M. Amdahl, G. A. Blaauw, F. P. Brooks, Jr., "Architecture of the IBM System/360", *IBM Journal of Research and Development*, **8**(2):87-101, 1964.

[15] E. W. Dijkstra, *Cooperating sequential processes*. Technical Report EWD-123, Eindhoven University of Technology, Netherlands, 1965.

[16] V. R. Pratt, M. O. Rabin, L. J. Stockmeyer, "A characterization of the power of vector machines". *Proceedings of STOC'1974*, pp. 122-134, 1974.

[17] S. Fortune, and J. C. Wyllie, "Parallelism in random access machines". *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing*, pp. 114-118, 1978.

[18] L. M. Goldschlage, "A universal interconnection pattern for parallel computers". *Journal of the ACM*, **29**(4):1073-1086, 1982.

[19] G. M. Ştefan, M. Maliţa: "*Can* One-Chip Parallel Computing *Be Liberated From* Ad Hoc Solutions? *A* Computation Model Based Approach *and Its* Implementation", *18th International Conference on Circuits, Systems, Communications and Computers*, Santorini Island, Greece, July 17-21, pp. 582-597, 2014.

[20] G. M. Ştefan: *Composition is the only independent rule in Kleene's model of partial recursive functions*, Available at: http://users.dcae.pub.ro/~gstefan/2ndLevel/technicalTexts/2019_Composition.pdf