From Kleene's Model to the Parallel Abstract Machine

Mihaela Maliţa and Gheorghe M. Ştefan

Abstract Parallel computation is mathematically defined by Kleene's model of partial recursive functions. The paper proves that only the composition rule is independent. Therefore, it is used to define the Kleene Machine, KM, and the Universal Kleene Machine, UKM, as the mathematical model for parallel computation. The Map-Reduce parallel abstract machine model is defined starting from UKM.

Key words and phrases : mathematical model of computation, abstract parallel machine, parallel computation, Kleene's recursive functions

Mathematics Subject Classification (2000) **:** 03D75, 03D80.

1 Introduction

The historical evolution of the mono-core computation teaches us about the right way to be followed in developing multi- and many-core computation. The main steps in the history of mono-cores are:

1936: four equivalent ***mathematical models*** are published [10] [4] [7] [8] (all seem to be triggered by Gödel's paper on non-decidability) out of which the *Turing Machine* offered the most appropriate technologically suggestion for future developments leading eventually to mono-core computing

1944-45: MARK 1 computer, built by IBM for Harvard University, consecrated the *Harvard* ***abstract model***, while von Neumann's report [11] introduced the *von Neumann* ***abstract model***

1953: IBM launched *IBM 701*, the first electronic computer ***manufactured in quantity***

1964: the concept of ***computer architecture*** is introduced [3] to allow the independent development for the two different aspects of computer design with different rates of evolution: software and hardware.

Thus, there are now on the market few stable and successful mono-core architectures, such as x86, ARM. The emergence of multi- and many-core computing, under the market criteria pressure, had its logical evolution distorted (in 1962, Burroughs started the manufacturing in quantity; only in 1965, Edsger W. Dijkstra raised architectural issues [6]; in 1974, the first abstract machine model is proposed [9], confused with the mathematical model). In this paper, we are able now to follow for multi/many-core (parallel) computation an improved path which consists of the following four steps: *mathematical model – abstract model – architectural model – manufacturing in quantity.*

In section 3, Kleene's mathematical model [7] is used to define the Kleene Machine, KM, and the Universal Kleene Machine, UKM, as a genuine *mathematical model* for parallel computation. Then, in section 3, the Map-Reduce abstract machine model is proposed as the second step, in defining an *abstract machine* model for parallel computation.

2 Kleene's Model for Parallel Computing

2.1 Only the Composition Rule is Independent

From [7] the following definition for partial recursive functions is extracted:

Definition 2.1 *Any partial recursive function* $f : \mathbb{N}^n \rightarrow \mathbb{N}$ *can be computed using the* ***initial functions****:*

- $ZERO(x) = 0$*: the variable* x *takes the value zero*
- $INC(x) = x + 1$*: increments the variable* $x \in \mathbb{N}$
- $SEL(i, x_1, \ldots, x_n) = x_i$*:* i *selects the value of* x_i *from the sequence* $X =< x_1, \ldots, x_n >$ *(called* identity function *in Kleene's paper)*

and the application of the following ***rules****:*

- Composition:
  $f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_p(x_1, \ldots, x_n))$ *where:* f *is a total function if* $g : \mathbb{N}^p \rightarrow \mathbb{N}$ *and* $h_i : \mathbb{N}^n \rightarrow \mathbb{N}$*, for* $i = 1, \ldots p$*, are total functions*
- Primitive recursion:
  $f(x_1, \ldots, x_n, y) = g(x_1, \ldots, x_n, f(x_1, \ldots, x_n, y - 1))$ *while*
  $f(x_1, \ldots, x_n, 0) = h(x_1, \ldots, x_n)$*, where:* f *is a total function if* g *and* h *are total functions (called* ordinary recursion *in Kleene's paper)*
- Minimization (least-search):
  $f(x, y) = \mu y[g(x, y) = 0]$*, i.e., the rule computes the value of function* $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ *as the smallest* y *for which the function* $g(x, y) = 0$*, if any.*

2.1.1 Preliminary Definitions

Definition 2.2 *The* reduction-less composition*, or* map composition*, MC, is the composition:* $f(x_1, \ldots, x_n) =< h_1(x_1, \ldots, x_n), \ldots h_p(x_1, \ldots, x_n) >$ *because:* $g(y_1, \ldots, y_p) =< y_1, \ldots, y_p >$*.*

Definition 2.3 *The* reduction composition*, RC, is the composition:* $f(y_1, \ldots y_p) = g(y_1, \ldots, y_p)$ *with* $h_i(x_1, \ldots, x_n) = SEL(i, x_1, \ldots, x_n) = x_i$*, for* $i = 1, \ldots, n$ *and* $n = p$*.*

According to the previous two definitions, the composition rule could be considered as having a ***map-reduce*** structure.

Definition 2.4 *Let be the reduction-less composition* $C_i : \mathbb{N}^i \rightarrow \mathbb{N}^{i+1}$ *with:* $h_{i+1}(x_1, \ldots, x_i) = P_i(SEL(i, x_1, \ldots, x_i)) = P_i(x_i)$ *while* $h_k(x_1, \ldots, x_i) = SEL(k, x_1, \ldots, x_i) = x_k$ *for* $k = 1, 2, \ldots, i$*. The actual form of* C_i *results:*

$$C_i(x_1, \ldots, x_i) =< x_1, x_2, \ldots, x_i, P_i(x_i) >$$

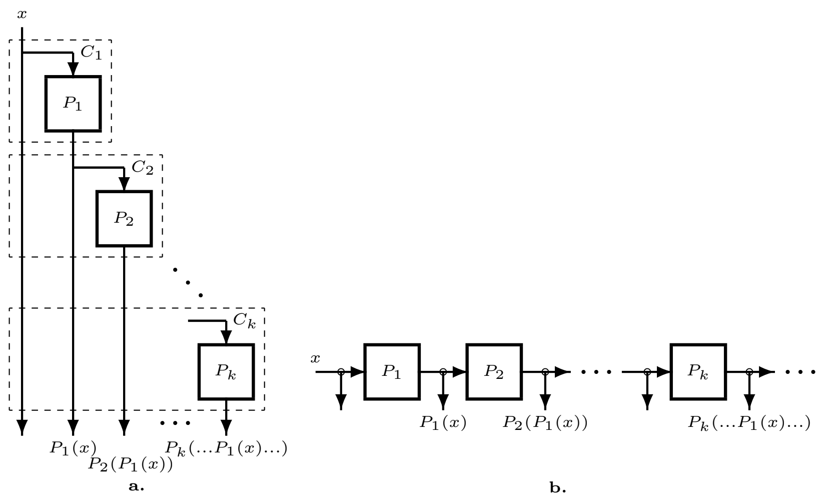
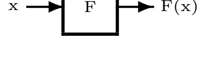


Figure 1: The multi-output pipeline structure, MOP, as a repeated application of the composition C_i, for $i = 1, 2, \ldots, k, \ldots$. **a**. The explicit application of C_i. **b**. The resulting MOP circuit structure.

Definition 2.5 *The repeated application of* C_i *(see Figure 1a*[1]*), starting from* $i = 1$ *with* $x_1 = x$ *defines the* multi-output pipelined*, MOP, function*

[1]The graphic representation of functions uses blocks of form



which stand for the computation $F(x)$.

$MOP : \mathbb{N} \to \mathbb{N}^n$, as

$$MOP(x) = (x, P_1(x), P_2(P_1(x)), \ldots, P_k(P_{k-1}(\ldots, (P_1(x)\ldots)), \ldots)$$

The function $MOP(x)$ is a total function if the functions P_i are total functions, since it is computed using only the repeated application of the composition C_i. For the theoretical model, n is not limited to a specific value.

Definition 2.6 *The function* predicated selection*, $PS : \{\{0,1\} \times \mathbb{N}\}^n \to \mathbb{N}$ is*

$$PS(S) = ADD(MULT(SEL(0,S)), \ldots)$$

where: $S =< (p_0, s_0), (p_1, s_1), \ldots >$ takes a sequence of pairs $(predicate, scalar)$, and returns a scalar. The functions ADD and $MULT$ add and multiply, correspondingly, the components of the sequence received as argument.

MULT is the function *mapped* on the first level of composition, while ADD is the *reduction* function. The function PS is used when one and only one p_i takes the value 1, i.e., for $i = k$ $p_i = 1$ and for $i \neq k$ $p_i = 0$:

$$PS(S) = ADD(\ldots, MULT(0, s_{k-1}), MULT(1, s_k), MULT(0, s_{k+1}), \ldots)$$

$$PS(S) = ADD(0, 0, \ldots, 0, s_k, 0, \ldots) = s_k$$

Definition 2.7 *Let be the Boolean sequence $X =< x_0, x_1, \ldots >$. The function $FIRST(X) : \{0,1\}^n \to \{0,1\}^n$ is computed, by definition, using a two stage structure as follows:*

1. *the function $MOP_{OR}(x, X) =< y_0, y_1, \ldots >$, for $x = SEL(0, X)$, computes the* OR prefix *function, $ORPX(X) : \{0,1\}^n \to \{0,1\}^n$, with the following computation performed in each P_i:*

   $$P_i(x, X) =< y_i, X >=< OR(x, SEL(i, X)), X >$$

   for $i = 1, 2, \ldots$, which allows to select from its multi-output the sequence of OR prefixes from the binary sequence X:

   $$ORPX(X) = MOP_{OR}(SEL(0, X), X) =< y_0, y_1, \ldots >=$$

   $$< x_0, (x_0 + x_1), (x_0 + x_1 + x_2), \ldots >$$

   where "+" stands for the logic OR function

2. *a reduction-less composition with $h_i(ORPX(X)) = first_i$, where*

   $$first_i = AND(SEL(i, ORPX(X)), NOT(SEL((i-1), ORPX(X))))$$

   for $i = 0, 1, \ldots$, where $SEL((i-1), ORPX(X)) = 0$ for $i = 0$,

such that

$$FIRST(X) =< first_0, first_1, \ldots >$$

is a sequence of Booleans with a single 1, **if any***.*

2.1.2 Theorems

Theorem 2.1 *The primitive recursive rule is reducible to repeated applications of specific compositions.*

Proof. The primitive recursion rule (see Definition 2.1) could be applied in its iterative form using the following expression:

$$f(x,y) = \underbrace{g(x, g(x, g(x, \ldots g}_{y\,times}(x, h(x))\ldots)))$$

Let be the specific instantiation of the MOP function (see Definition 2.5), with the predicate $p_i = (y == i)$, where the functions P_i compute a quintuple, as follows:

$$P_1 = H(i,x,y) =< INC(i), x, y, f(x,i), p_i >=< z_1, z_2, z_3, z_4, z_5 >$$

where: because, $i = 0$, $f(x,i) = h(x)$

$$P_k = G(i,x,y,f(x,i{-}1)) =< INC(i), x, y, f(x,i), p_i >=< z_1, z_2, z_3, z_4, z_5 >$$

for $k = 2, 3, \ldots$ and $i = 1, 2, \ldots$, where each P_k, function is computed for the quadruple $< z_1, z_2, z_3, z_4 >$ generated by P_{k-1}. From the outputs of the MOP function we select the sequences of pairs $< z_4, z_5 >$ in order to obtain:

$$S =<< f(x,0), (y == 0) >, < f(x,1), (y == 1) >, \ldots >$$

S is used as input for the function PS which provides the result $f(x,y)$. $\square$

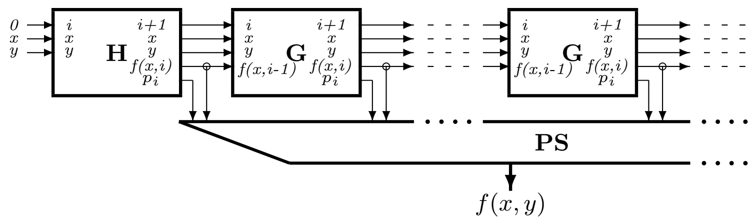


Figure 2: The specific MOP structure for partial recursive computation.

Figure 2 presents the function (see Definition 2.6) obtained by composing a specific MOP function with PS function. The two stage computation just described, as a structure indefinitely extensible to the right, is a theoretical model, because the index i takes values no matter how large, similar with the indefinitely extensible ("infinite") tape of Turing's machine. But, it is very important that the algorithmic complexity of the description is in $O(1)$, because the functions H, G, MOP and PS have constant size descriptions.

Theorem 2.2 *The minimization (least-search) rule is reducible to repeated applications of specific compositions.*

Proof. The minimization (least-search) rule computes the value of $f(x)$ as the smallest y for which $g(x, y) = 0$. The three steps used in the evaluation of the function are $f(x) = F_3(F_2(F_1(x)))$, three specific compositions, as follows :

1. $F_1 : \mathbb{N} \rightarrow \{0, 1\}^n$, is a "speculative" reduction-less composition, which returns the "infinite" sequence of predicates, X_1:

   $$F_1(x) =< h_0(x), h_1(x), \ldots >=< p_0, p_1, \ldots >= X_1$$

   with $h_i(x) = (g(x, i) == 0) = p_i$; in this step the function g is computed for x and "all" positive integers starting with 0.

2. $F_2 : \{0, 1\}^n \rightarrow \{0, 1\}^n$ is a reduction-less composition with:

   $$F_2(X_1) = FIRST(X_1) = X_2$$

   It provides a sequence of predicates with no more than one 1, ***if any***.

3. $F_3 : \{0, 1\}^n \rightarrow \mathbb{N}$, is a composition with:

   $$h_i(X_2) = ADD(SEL(i, X_2), (SEL(i, X_2) \ ? \ i : 0))$$

   $$g = ADD(h_0(X_2), h_1(X_2), \ldots) = result$$

   If $result = 0$, **then** the function $f(x)$ is not defined for the input variable x, **else**, if $result > 0$, the function is defined for x and

   $$f(x) = result - 1$$

   It selects as solution, if it exists, the incremented index corresponding to the occurrence of 1 in X_2, ***if any***.

□

The computation just described is only a theoretical model, because the index i has an indefinitely large value. But, the size of algorithmic description remains $O(1)$.

Corollary 2.1 *Any computation defined in Definition 2.1 can be done, according to Theorem 2.1 and Theorem 2.2, using the initial functions and the repeated application of the composition rule.*

Kleene's approach defines the other two rules, ordinary (primitive) recursion and minimization (least-search), for helping in the taxonomy of the recursive functions. For defining the computation the Corollary 2.2 makes the necessary delimitation.

2.2 *Kleene Machine*: a Parallel Model of Computation

According to Theorem 2.1 and Theorem 2.2 only the composition rule must be considered in defining what means computation.

Definition 2.8 ***Kleene Machine****, KM, consists of a two-layer construct, associated to the composition rule, see Figure 3, with:*

1. ***map*** *level: the independent functions h_i, for $i = 0, 1, \dots$*
2. ***reduction*** *level: the function g*

where h_i, for $i = 0, 1, \dots$ and g are initial functions or compositions.

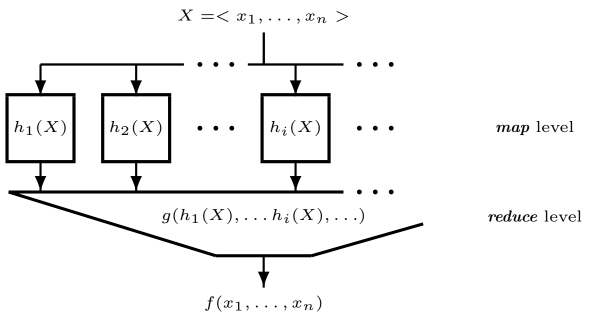


Figure 3: Kleene Machine.

Because Kleene's model is equivalent with the Turing Machine model the next corollary is true.

Corollary 2.2 ***Kleene Machine*** *represents a mathematical model for parallel computation.*

2.3 *Universal Kleene Machine*

For each function f there is a KM. As Turing defined its Universal Turing Machine, UTM, the concept of KM must be accompanied by the concept of ***Universal Kleene Machine***, UKM. An UKM must provide the possibility to define any KM on the same structure and to compose KMs.

Definition 2.9 ***Universal Kleene Machine*** *is a finite KM, with p cells on the map-level, loop connected with a Finite State Machine, FSM, with access to a non-finite Memory. The map-level of KM contains p identical cells, $C_1, \dots, C_p$, each having the function:*

$$C(h_i, X) = SEL(h_i, H_0(X), H_1(X), \dots, H_{q-1}(X))$$

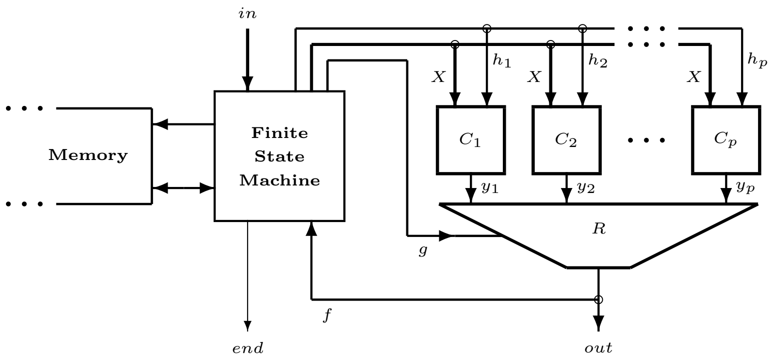


Figure 4: Universal Kleene Machine.

while the reduction-level has the function:

$$R(g, Y) = SEL(g, G_0(Y), G_1(Y), \ldots, G_{r-1}(Y))$$

where: Y is the p-component sequence $< y_1, y_2, \ldots, y_p >$, and the two finite sets of functions

$$\mathbb{H} = \{H_0(X), H_1(X), \ldots, H_{q-1}(X)\}$$

$$\mathbb{G} = \{G_0(X), G_1(X), \ldots, G_{r-1}(X)\}$$

represent the characteristic set of functions, $\mathbb{F} = \mathbb{H} \cup \mathbb{G}$, used to compose any computable function, while $\mathbb{A} = \mathbb{H} \cup \mathbb{G} \cup \mathbb{N}$ represents the alphabet of UKM. Formally:

$$UKM = (S, \mathbb{A}, S_0, f)$$

where:

- *S is the finite states set of the FSM*
- *$\mathbb{A}$ is the alphabet of the UKM*
- *$S_0 \in S$ is the initial state of the Finite State Machine*
- *f is the transition function*

  $$f : S \times \mathbb{A} \times \mathbb{N} \rightarrow S \times \mathbb{H}^p \times \mathbb{N}^v \times \mathbb{G} \times \mathbb{A}$$

  which, in each cycle, according to the current state of FSM, the element of $\mathbb{A}$ read from Memory and the output provided by the reduction R, generate the next state of FSM, a stream of p functions for the map-level, a stream of v input values, the function for the reduce-level and the element from $\mathbb{A}$ to be written in Memory.

The UKM is initialized in the state S_0, and after a finite number of cycles, if the computation is possible, the output "end" is activated indicating the end of computation with the result on the output "out".

Theorem 2.3 *The minimal UKM is defined for $p = 3$ with each cells having only one function, as follows:*

$$C_1(X) = ZERO,\ C_2 = INC(SEL(k, X)),\ C_3 = SEL(i, X)$$

while $R = SEL(g, y_0, y_1, y_2)$.

Proof. The FSM is used to build the sequence X from the string of outputs (see Figure 4). The minimal UKM becomes an UTM. □

3 ***Map-Reduce Abstract Machine*** Model for Parallel Computing

From the UKM, as a mathematical model for parallel computation, to an abstract model for parallel computation able to support an actual implementation, few simplifying steps are needed. They are not formally sustained by rigorous proofs. The purpose of this transition is motivated by the transition from a *competent* model to a model which is also able to attain high *performance*.

Definition 3.1 *A computation model is* ***competent*** *if the computation it supports ends in a finite number of steps.*

Definition 3.2 *A computation model is* ***performant*** *if the computation it supports ends in a minimal number of steps.*

The road from competence to performance requires engineering work. The result is validated by the evaluation of the resulting performance.

3.1 Forms of Parallelism

Five forms of simplified parallelism are emphasized as the meaningful set of particular compositions able to provide the transition from a competent model to a performant one.

Definition 3.3 Data-parallel *computation is defined for MC computation (see Definition 2.2) when $n = m$ with $h_i(x_1, \ldots, x_n) = h(x_i)$, for $i = 1, \ldots, n$.*

The same function, h, is applied in parallel to each component, x_i, of the input vector.

Definition 3.4 Reduction-parallel *computation is defined for RC computation (see Definition 2.3) when $n = m$ with $h_i(x_1, \ldots, x_n) = h(x_i) = x_i$, for $i = 1, \ldots, n$.*

On the first level of the composition, the map level, all the functions of one variable, x_i, perform the identity function.

Definition 3.5 Speculative-parallel *computation is defined for MC computation when $n = 1$.*

Each function h_i has the same input variable x_1.

Definition 3.6 Thread-parallel *computation is defined for MC computation when $n = m$ with $h_i(x_1, \ldots, x_n) = h_i(x_i)$, for $i = 1, \ldots, n$.*

Each cell performs a specific function on different data.

Definition 3.7 Time-parallel *computation is defined for repeated application of the composition rule with $m = n = 1$.*

The repeated application of time-parallel computation provides the following pipe of functions:

$$f(x) = f_p(f_{p-1}(f_{p-2}(\ldots f_1(x)\ldots)))$$

3.2 Integral Parallelism

We claim that the previous five forms cover efficiently the most frequent parallel computation patterns. Integrating them on a single engine provides the ***parallel abstract model*** for computation. In Figure 5, the MapReduce recursive parallel abstract model for parallel computation is presented. It consists of:

- pairs *eng-mem* in the MAP section; they correspond to the cells C_i from UKM, and consist of:
  - *eng*, the engine, which is an execution unit or a processing unit
  - *mem*, the local memory to store data (when *eng* are execution units) or data and programs (when *eng* are processing units)
- REDUCE unit; it corresponds to the R function in UKM
- CONTR, a controller used as sequencer; performs the function of FSM from UKM
- MEMORY, a memory resource for data and programs.

The entire structure from Figure 5 can be seen as a two-part entity:

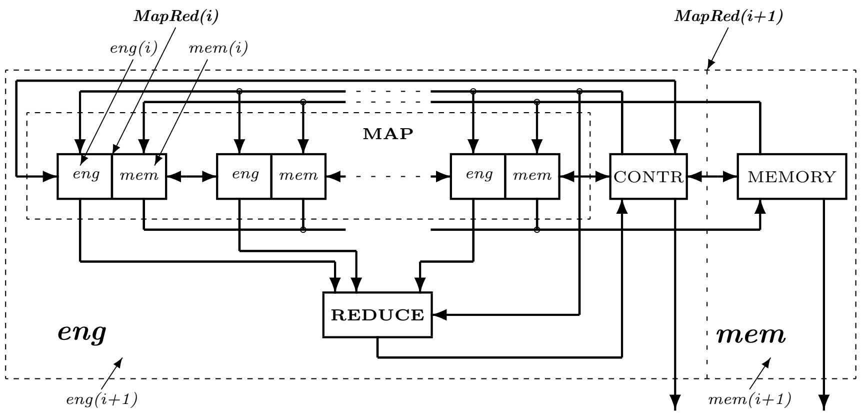


Figure 5: ***MapReduce recursive abstract model*** for parallel computation.

- ***eng***: MAP + REDUCE + CONTR
- ***mem***: MEMORY

which behaves as a cell in a ***recursive hierarchy*** of a map-reduce organization of many-core coomputation.

4 Concluding Remarks

Composition is the only independent rule in Kleene's model. Kleene defined the primitive recursive rule and the minimization (least-search) rule to support the taxonomy of the recursive functions.

Composition defines *Kleene Machine* as the mathematical model for parallel computation with two levels: the map level, which provides a *synchronous parallelism*, and the reduction level, which provides the *diachronic parallelism.*

Universal Kleene Machine represents the support for the parallel abstract machine model. It moves the not-finite resources at the level of a simpler structure – the "infinite" memory – similar with Turing's approach. Its simplest form is UTM.

The Map-Reduce parallel abstract machine model adds to the competence of UKM the possibilities to *actually* develop an efficient computing engine with high performances. The five forms of parallelism – *data, reduction, speculative, thread, time* – are simplified forms with an efficiency to be proved by use, not by formal proofs.

The generic parallel architecture and organization of an actual machine will be also *Map-Reduce* as a consequence of the mathematical and abstract model. From the chip level until de cloud level the *recursive abstract model* is able to provide the same organization and programming style.

Future work: the architectural definition using Backus' Functional Forms [2], and validation using Berkeley's dwarfs [1].

References

[1] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams and Katherine A. Yelick: *The landscape of parallel computing research: A view from Berkeley*, Technical Report No. UCB/EECS-2006-183, December 18, 2006.

[2] John Backus: "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs", *Com. of the ACM* 21, 8 1978. 613-641.

[3] Gerrit Blaauw, and Fred P. Brooks, The structure of System/360, part I - Outline of the logical structure. IBM Systems Journal 3, 2, 1964. 119135.

[4] Alonzo Church, An unsolvable problem of elementary number theory. *The American Journal of Mathematics* 58, 1936. 345363. Republished in [5].

[5] Martin Davis, *The Undecidable. Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions.* Dover Publications, New-York, 2004.

[6] Edsger W. Dijkstra, Co-operating sequential processes. *Programming Languages* Academic Press, New York, 43112. Reprinted from: Technical Report EWD-123, Technological University, Eindhoven, the Netherlands, 1965.

[7] Stephen Kleene, General recursive functions of natural numbers. *Mathematische Annalen* 112, 5, 1936. 727-742. Republished in [5].

[8] Emil Post, Finite combinatory processes. Formulation 1. *The Journal of Symbolic Logic* 1, 1936. 103105. Republished in [5].

[9] Vaughan R. Pratt, Michael O. Rabin, and Larry J. Stockmeyer, A characterization of the power of vector machines. *Proceedings of STOC1974*, 1974. 122134.

[10] Alan M. Turing, On computable numbers with an application to the Eintscheidungsproblem. *Proceedings of the London Mathematical Society* 42, 1936. Republished in [5].

[11] John von Neumann, First draft of a report on the EDVAC. *IEEE Annals of the History of Computing* 5, 4, 1993.

Mihaela Maliţa
Saint Anselm College
Post address: 100 Saint Anselm Drive, Manchester, NH 03102
E-mail: `mmalita@anselm.edu`

Gheorghe M. Ştefan
Politehnica University of Bucharest
Post address: 313 Splaiul Independenţei, Bucureşti, RO 060042
E-mail: `gstefan@arh.pub.ro`