Liquid Brain: The Proof of Algorithmic Universality of Quasichemical Model of Fine-grained Parallelism

Alexander N. Gorban', Katya O. Gorbunova
Institute of Computational Modelling SB RAS
660036, Akademgorodok, Krasnoyarsk, Russia
E-mail: gorban@cc.krascience.rssi.ru

D. C. Wunsch II
Applied Computational Intelligence Laboratory
Department of Electrical Engineering, Texas Tech University,
Lubbock, TX 79409-3102,
E-mail: Dwunsch@aol.com

Keywords: parallel computations, fine-grained parallelism, distributed computing

Abstract

A new formal model of parallel computations – the Kirdin kinetic machine – is suggested in [1]. It is expected that this model will play the role for parallel computations similar to Markov normal algorithms, Kolmogorov and Turing machine or Post schemes for sequential computations. The basic ways in which computations are realized are described; correctness of the elementary programs for the Kirdin kinetic machine is investigated. It is proved that the deterministic Kirdin kinetic machine is an effective calculator. A simple application of the Kirdin kinetic machine – heap encoding – is suggested. Subprograms similar to usual programming enlarge the Kirdin kinetic machine.

Introduction

The problem of effective programming with fine-grained parallelism is far from being solved. It seems that, despite numerous efforts, we have not yet understood parallel computations, considering them mainly as result of usual algorithm parallelization. There are some promising approaches based on models of computing environments constructed from large number of elementary calculators of the same type (neural networks, cellular automata etc.). If it is possible to implement a problem in such environment (for example, by methods of neural networks training [7]), further realization with parallel computers can be easily constructed within the framework of the ideas "similar tasks for different elements". There are other perspective ideas and approaches to construction of models of fine-grained parallelism besides neural networks.

Parallel Substitution Algorithms (PSA) [9] conceptually go back to von Neumann cellular automata, but have more powerful expressive capabilities. The following fundamental concepts form the background of the Parallel Substitution Algorithm:

1) Fine-grained parallelism. Each data-item is introduced being attached to a point in the computation space represented as a countable discrete set of names. The computation is an

iterative procedure over a data array in this space. At each step certain data subarrays are replaced by other ones, these actions being done all over the whole data-array.

2) Decentralized control. No order of operation execution is defined in the model. Each substitution is performed when and where the ready conditions coincide with a data-pattern in the space. Thus, the associative mechanism of the time-spatial control of the computation process is performed, the spatial relations being explicitly represented by means of the functions defined in the computational space.

3) Synchronous mode of execution. An abstract outer clock is presumed to exist, making the computational process to obey the following rules:

- all operations ready to be executed should be executed simultaneously with the first coming clock pulse,
- no changes in the array are allowed out of the clock pulses. The asynchronous mode of execution is also of great theoretical interest. Its investigation is very important for understanding the behavioral properties of computations in the cellular spaces.

4) Interpretability by automata nets. A set of substitutions representing the cellular computation admits the direct mapping onto a net of automata. This allows one to construct methods and tools for architectural design of hardware implementation of cellular algorithms.

The direction of the development of the PSA theory is stipulated by the objective of its creation: to constitute the fundamentals for methods of synthesis of algorithm-oriented architectures of cellular processors

The chemical computer (SCAM – *Statistic Cellular Automata Machine*) is offered in [8] in development of the cellular automata theory. SCAM is based on imitation simulation by Monte-Carlo methods of a class of heterogeneous chemical reactions occurring in a thin layer of molecules adsorbed on the surface of a crystal catalyst. Algorithmic universality for SCAM has been proved.

Artificial Immune Systems [10] are highly distributed systems based on the principles of natural system. This is a subject of great research interest because of its powerful processing capabilities. In particular, it performs many complex computations in a completely parallel and distributed fashion. Like the nervous system, the immune system can learn new information, recall previously learned information and performs pattern recognition tasks in a decentralized fashion. Also its learning takes place by evolutionary processes similar to biological evolution. [10] reviews the models that have been developed based on various computational aspects of the immune system. The existing immunity-based methods emulate one or the other mechanisms of the natural immune system. Further study should integrate all the potentially useful properties in a single framework in order to develop a robust immunity-based system. There are many potential application areas in which immunity-based models appear to be very useful. They include fault detection and diagnosis, machine monitoring, signature verification, noise detection, computer and data security, image and pattern recognition, and so forth. It is to be noted that the mechanisms of the immune system are remarkably complex and poorly understood, even by immunologists. Understanding the immune system is important, both because of determining its role in handling complex diseases and because of potential applications to computational problems. Moreover, if we can understand the functionality and the inherent mechanisms of various components of the immune system from the computational viewpoint, we may gain better insights about how to engineer massively parallel adaptive computations.

In [14] E.A. Liberman suggested in 1972 an interesting model. He insists that the living cell is controlled by molecular stochastic computer of parallel-successive action. MMC operates with molecules-words (DNA, RNA, proteins) according to the program recorded in DNA and RNA. Operations are produced by molecular devices (RNA- and DNA-polimerases, ligases, proteinases and so on) contained in proteins and RNA-molecules. Molecular devices operating with molecules-words are recorded on molecules themselves, and they are read off by ribosomes. Therefore the program of the reorganization of the program itself may be recorded on the molecules-word. MCC operates with molecular words having definite addresses. The words and the operators collide by Brownian movement and combine if the molecular surfaces of an address segment are complementary and properly oriented. It is possible to reproduce not only the program but also the operators of MCC. Molecular computer operates with words-molecules according to the program , recorded in DNA, with the aim of predicting outer situation in the next time-moment and selecting macroscopic motion. Each step of direct calculation needs the consumption of minimally necessary portion of free energy and search is due to the Brownian movement without free energy loss. A search of words of RNA-alphabet is performed by means of steady-state chemical reactions catalyzed by polynucleotide phosphorylases. Price of action D_A is determined as a product of the value of free energy loss by one operation E_O and these operations time T_O: $D_A \approx E_O T_O \approx 10^{13} h$; and for a search $D_A \approx 10^3$ h, where h is Planck constant (quantium of action). The hypotheses are proposed in that paper, that MCC of neurons is adopted for the work of brain.

A new abstract model of computations – *the Kirdin kinetic machine* – is investigated in this paper. This model is expected to play the same role for parallel computations as the Turing machine and other abstract algorithmic calculators for sequential computations.

The Kirdin kinetic machine is based on chemical reactions in liquids or gases. Our optimistic expectations go back to the theorem of M.D.Korzuhin [12] on chemical reaction ability to imitate any dynamic system for finite times and to the theorem of A.N.Gorban [13] on chemical systems approximating any dynamic systems.

1. The Kirdin Kinetic Machine

A processed unit for the Kirdin kinetic machine is *an ensemble* of strings M from the alphabet L, which is identified with a function F_M taking non-negative integer values: $F_M{:}L^* \to N \cup \{0\}$. The value of $F_M(s)$ is interpreted as a number of copies of a string s in the ensemble M.

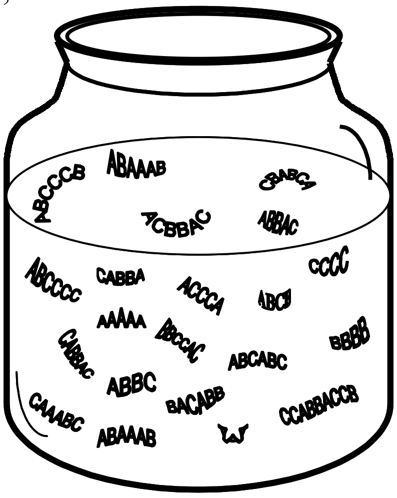

Fig.1. A jar with strings

The processing consists of an aggregate of *elementary events*, which occur non-deterministically and in parallel. An elementary event $S{:}M \mapsto M'$ means that from the ensemble M an ensemble K^- is removed (it is possible if for arbitrary s $F_K^-(s) \le F_M(s)$)and an ensemble K^+ is added. The ensembles K^- and K^+ are unambiguously set by *rules* or *commands*, which are combined in *a program*. The commands can be of only three kinds (*u, w* – arbitrary, *v, f, g, k, q, s* are fixed):

1) *Disintegration* $uvw \to uf + gw$;

2) *Synthesis* $uk + qw \to usw$;

3) *Replacement* $uvw \to usw$.

The program ***P*** *is applicable* to an ensemble *M*, if any command of ***P*** is applicable to *M* . Elementary event *S* is unambiguously determined by a rule *p* from the list of commands of the program ***P***, and by an ensemble K^{-} determined by this rule and such that $F_K^{-}(s) \leq F_M(s)$ for any s. An elementary event *S is allowable* for an ensemble *M* and a program ***P*** if there is a rule *p* in the list of commands of the program ***P*** and the values of function F_M for strings in the left part of this rule are positive.

We shall say that *N* of allowable events *are compatible*, if $F_M - \sum_{i=1}^{n} F_i^{-} \geq 0$, where F_i^{-} is a removed ensemble for the *i*-th event.

Ensemble *M* is called *a final ensemble* if no command of the program ***P*** is applicable to it. ***P*** is referred to as *a finite program* if application of commands of the program to the initial ensemble always leads to a final ensemble. If all final ensembles coincide, the program ***P*** is named *deterministic* for the ensemble *M.*

The Kirdin kinetic machine can be informally described as a jar with strings. We add rules-catalysts to this jar, some of them, colliding with the strings, promote their disintegration, others, meeting a pair of suitable strings, promote their synthesis, and the third replace some substrings in strings.

The basic ways of realization of calculations are described, *correctness* of elementary programs for the Kirdin kinetic machine is investigated.

2.Statistical method of implementation

The "statistical" method of implementation of the Kirdin kinetic machine is of particular interest. In the limit of large ensembles it gives kinetic behavior, similar to the behavior of a complex system of chemical reactions (wherefrom, in fact, the name "the Kirdin kinetic machine" appears). Some non-negative number r_i – "constant of rate" is compared with each *i*-th command in statistical realization. The realization can be presented as follows: for small Δt with the probability $r_i\Delta t$ one of commands-reactions is chosen, and with probability $1-\Sigma_i r_i \Delta t$ empty command is chosen (nothing occurs); the probability of choice of two or more commands is infinitesimal ($o(\Delta t)$). If *i*-th command-reaction is chosen, then from the ensemble strings are chosen randomly and with equal probability, in the amount necessary for fulfillment of the command. If the command is applicable to this set of strings, it is executed and we pass to the following Δt, if it is inapplicable, the ensemble does not change and we pass to the following Δt all the same. In the limit $\Delta t \to 0$ we obtain random process – statistical model of the Kirdin kinetic machine.

3. Correctness of the programs consisting of one command

3.1. Desintegration

Consider the program consisting of a single disintegration command $uvw \to uf + gw$

Applicability

This program is applicable for an ensemble *M* if there exist such strings $u,\ w \in L^*$, that $F_M(uvw) > 0$.

Finiteness

Lemma 1. The program **P** consisting of a single disintegration command is not finite, if $v \subset f$ or $v \subset g$.

Proof. Let $v \subset f$ or $v \subset g$. That is the program $uvw \to uf_1vf_2 + gw$ or $uvw \to uf + g_1vg_2w$. In both cases the program application results in a new string uf_1vf_2 or g_1vg_2w for which the program is applied and so on is not finite. Lemma 1 is proved.

Example. $uAw \to uAB + Baw$

The program is applied to an arbitrary ensemble if one string at least includes symbol A. With every disintegration of such a string, two more strings will be generated for which a program is applied, that is the reason why the program won't be finite.

Lemma 2. Program P consisting or a single disintegration command won't be finite for the ensemble M, if

1) there are two possibly similar divisions of the string $v=a_1b_1=a_2b_2$;
2) $f=b_1b_2x$, $g=ya_1a_2$, where x and y are some fixed strings;
3) there exists a string uvw $(F_M(uvw) > 0)$ that $u=u_1a_1$ or $w=b_2w_1$.

Proof. Under lemma conditions

$v=a_1b_1=a_2b_2,\ uvw \to ub_1b_2x + ya_1a_2w,$

$F_M(u_1a_1vw) > 0$ or $F_M(uvb_2w_1) > 0$.

To prove infiniteness it is enough to indicate one infinite sequence of elementary events.

We shall consider disintegration of the string u_1a_1vw:

1) $\{u_1a_1vw\} \mapsto \{u_1\underline{a_1b_1}b_2x,\ ya_1a_2w\}$;
2) the program is applied to the string $u_1\underline{a_1b_1}b_2x$ as $v=a_1b_1$: $\{u_1\underline{a_1b_1}b_2x\} \mapsto \{u_1\ b_1b_2x,\ ya_1\underline{a_2b_2}x\}$
3) the program is applied to the string $ya_1\underline{a_2b_2}x$ as $v=a_2b_2$: $\{ya_1\underline{a_2b_2}x\} \mapsto \{y\underline{a_1}\ \underline{b_1}b_2x,\ ya_1a_2x\}$
4) the string $y\underline{a_1b_1}b_2x$ includes the string $\underline{a_1b_1}b_2$ as well as the string $u_1\underline{a_1b_1}b_2x$. Disintegration of the former was discussed in 2). This string generates a string including $a_1\underline{a_2b_2}$. Disintegration of the former was discussed in 3). Every disintegration step will generate new strings including alternately changing strings $\underline{a_1b_1}b_2$ or $a_1\underline{a_2b_2}$.

We shall now discuss disintegration of the string uvb_2w_1:

1) $\{uvb_2w_1\} \mapsto \{u\ b_1b_2x,\ ya_1a_2b_2w_1\}$;
2) the string $ya_1a_2b_2w_1$ includes the string $a_1a_2b_2$, it will generate the string with $\underline{a_1b_1}b_2$ and so on.

Thus we indicate infinite sequences of events under the lemma conditions. Lemma is proved.

Criterion of disintegration finiteness

Theorem 1. Program P consisting of a single disintegration command $uvw \to uf + gw$ is finite for the ensemble M if

I. $v \not\subset f$, $v \not\subset g$

II. for two arbitrary divisions $v=a_1b_1= a_2b_2$ where $a_1=a_2$ or $b_1=b_2$ are possible, at least one of three conditions is not fulfilled

(1) $f=b_1b_2f_1$, where f_1 is an arbitrary fixed string

(2) $g=g_1a_1a_2$ where g_1 is an arbitrary fixed string

(3) there exists the string uvw, $F_M(uvw) > 0$ such $u=u_1a_1$ or $w=b_2w_1$, where u_1 and w_1 are fixed strings

Proof. Consider an arbitrary string of the type $uvw\in M$. Let us build for it a binary tree as a derivation of all possible strings. The second line of every tree vertex indicates the conditions under which the disintegration program is applicable for the given string. The third line presents the form of the given string under these conditions. A descendant inherits conditions of all its ancestors. A left edge indicates a string of the type *uf*, a right adge – *gw*. The left and right tree branches are given separately.

We shall give explanations to the figures. The very left branch has the maximal length equal to $|u|$ under the condition that substring u is of the type $a_1...a_1a_2$, otherwise the branch is shorter. With every step to the right it generates the string gf_1. The conditions of its disintegration are analogous to the branch 1.2. Branch 1.2 has the maximal length to the right equal to $|f|$ under the condition that the substring f is of the type $b_1b_2...b_2$, otherwise the branch is shorter.

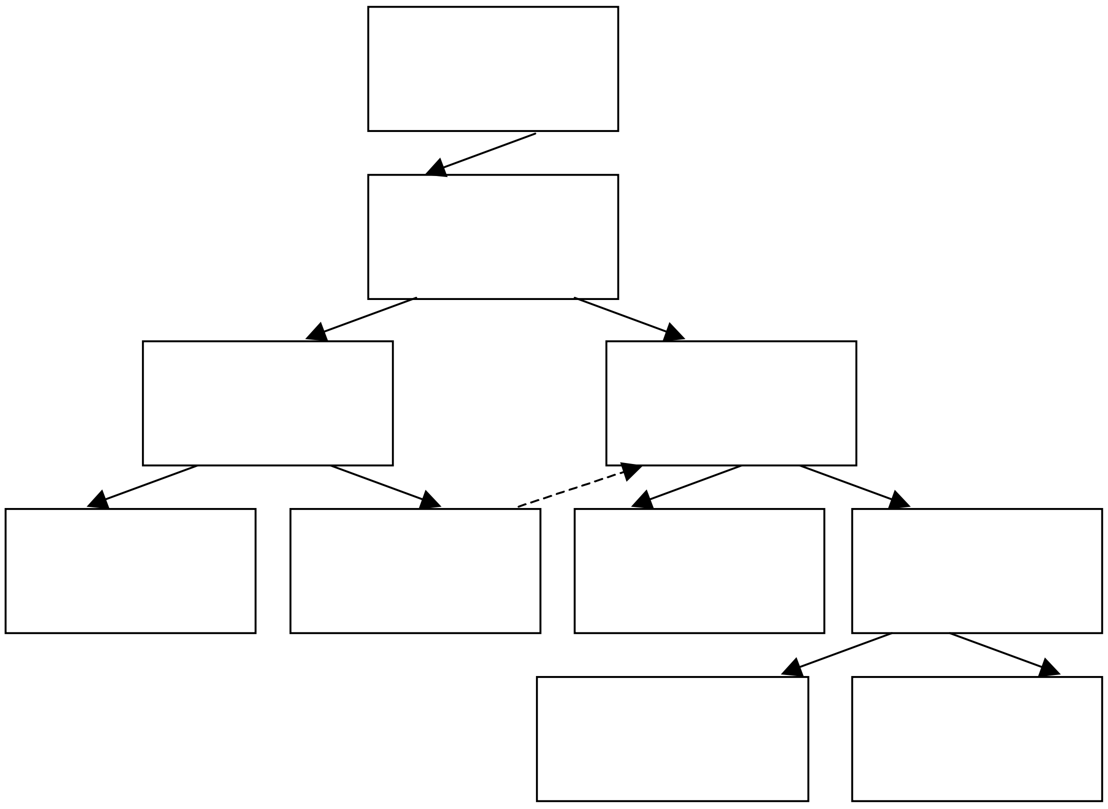

Fig.2. The left branch of disintegration tree.

With every step to the right branch 1.2 generates strings of the type $g_1b_1b_2...b_2f_2$ obtained under the condition $f=b_1b_2f_2$, while their disintegration is possible only in the case $g=g_2a_1a_2$ which contradicts the criterion conditions.

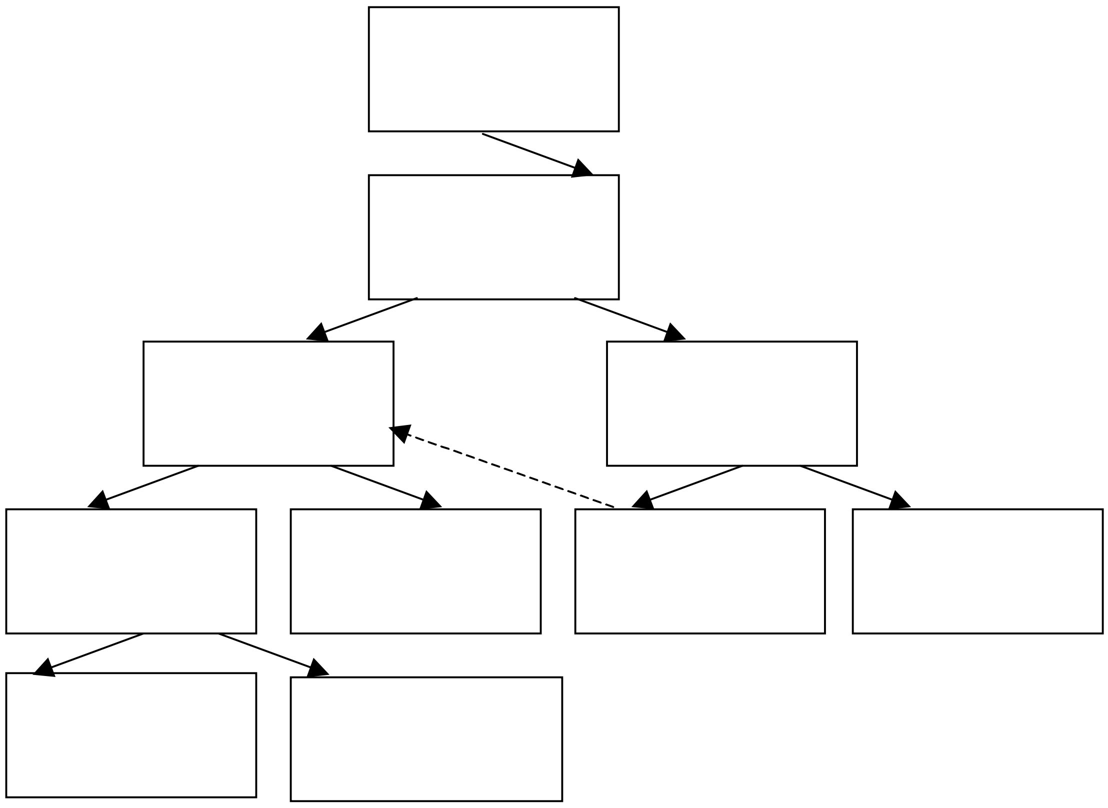

Fig. 3. The right branch of disintegration tree.

The right branch of the binary tree is symmetrical with a precision of resignations.

Thus we see that when the criterion conditions are met all sequences of elementary events will be finite for a program consisting of a single disintegration command, that was requested to prove.

Example. Infinite program: $uabw \to ubb + aaw$

Let string $aab \in M$.

$aab \to \{abb, aa\};$

$abb \to \{bb, aab\}.$

We obtained the string *aab* from which we started. The given program is applied to this string as well.

Determination

A program is not deterministic if in the initial ensemble M_0 or in one of ensembles obtained by one or several elementary events of the ensemble M_0 there is a string for which the allocation of substrings v takes place ambiguously. It may happen in cases when one or several symbols from the beginning of string v is included in its end in the same order.

Let string v be included in the string h ($F_M(h)>0$) more than once without meeting. I.e. the string h can be presented as $u_1v^{(1)}u_2v^{(2)}...u_nv^{(n)}w$, where n denotes the number of incorporations v into h. Elementary events taking place in an arbitrary order divide string h into $n+1$ strings: u_1f, gu_2f, ... , gu_nf, gw.

Criterion of disintegration determination

Theorem 2. Disintegration program is non-deterministic if there exist a division of string $v=aba$ and at least one of the following conditions is met.

1. $\exists u,w$ such that $F_M(uababaw)>0$;
2. $f=xababay$ or $g=xababay$, where x, y are fixed strings
3. $f=df_1$, $\exists\ u_1,w$ such that $F_M(u_1cvw) > 0$,where $cd=ababa$;
4. $g=g_1c$, $\exists\ u,w_1$ such that $F_M(uv\ dw_1) > 0$, where $cd=ababa$.

Otherwise disintegration is deterministic.

Proof. Under the condition $uabaw\rightarrow uf+gw$.

1. $\exists u,w$ such that $F_M(uababaw)>0$. Disintegration is applied to the string $uababaw$ with two methods: $u\underline{aba}baw\rightarrow uf+gbaw$ and $uab\underline{aba}w\rightarrow uabf+gw$ which results in two different ensembles. Hence, in this case disintegration is non deterministic.
2. $f=xababay$ or $g=xababay$, where x, y are fixed strings. The disintegration command can be presented in one of following ways: $uabaw \rightarrow uxababay + gw;$ $uabaw\rightarrow uf+gxababay;$ $uabaw \rightarrow uxababay+gx_1ababay_1$.For every case an arbitrary allowable elementary event generates the ensemble including strings with ambiguous separation out of the string aba for which point 1 of the given criterion is applicable.
3. $f=df_1$, $\exists\ u_1,w$ such that $F_M(u_1cvw) > 0$,where $cd=ababa$. Here cd is possible another division of the string $ababa$. A disintegration command is presented as follows: $uabaw\rightarrow udf_1+gw$. Applying the program to the string u_1cvw we obtain two strings u_1cdf_1 and gw. Application of disintegration to the string $u_1cdf_1= u_1ababaf_1$ is non-deterministic (see point 1).
4. $g=g_1c$, $\exists\ u,w_1$ such that $F_M(uv\ dw_1) > 0$, where $cd=ababa$. A disintegration command is presented as follows: $uabaw\rightarrow uf+g_1cw$. Applying the program to the string $uvdw_1$ we obtain two strings uf and g_1cdw. Application of disintegration to the string $g_1cdw=g_1ababaw$ is non-deterministic (see point 1).

If there exists a division of the string $v=aba$ but none of conditions 1-4 is met, then the initial ensemble and all ensembles obtained with elementary events from the initial ones do not generate the strings including the string $ababa$. Therefore such disintegration is deterministic.

Let the string v be impossible to present as the string aba, then intersected incorporation of v into each strings are impossible. Therefore non-determination is impossible for every ensemble.

The criterion of determination of disintegration is asserted.

Example. $uABABw \rightarrow uf + gw$, $F_M(ABABAB)>0$.

The string v can be presented as the division aba: $a=AB$, b is nil.The string $ABABAB$ meets the condition 1 of the criterion, therefore the string disintegration is non-deterministic. In fact, two different ensembles can be obtained from the string $ABABAB$ by applying disintegration command:

$F_{M1}(f)=1$, $F_{M1}(gAB)=1$;

$F_{M2}(Abf)=1$, $F_{M2}(g)=1$.

3.2. Synthesis.

Consider a program consisting of a single command of the type $uk+qw \rightarrow usw$.

Applicability

The program is applicable to arbitrary ensembles containing a pair of strings of the type uk and qw. $F_M(uk) > 0$, $F_M(qw) > 0$

Finiteness

It is obvious that the program is always finite. It glues all pairs of corresponding strings until they are exhausted.

Determination

In the general case the program is non-deterministic as it glues arbitrary strings ambiguously.

Synthesis determination criterion

Theorem 3. The synthesis program is deterministic if and only if at least one of the following conditions is met:

1. $\left|\{uk \mid F_M(uk) > 0\}\right| = 1$ and $\left|\{qw \mid F_M(qw) > 0\}\right| = 1$
2. $\left|\{uk \mid F_M(uk) > 0\}\right| = 1$ and $F_M(uk) \geq \sum\limits_{qw \in M} F_M(qw)$
3. $\left|\{qw \mid F_M(qw) > 0\}\right| = 1$ and $F_M(qw) \geq \sum\limits_{qw \in M} F_M(uk)$

Proof.

1. Let the initial ensemble be $F_M(uk)=n$, $F_M(qw)=m$ where u and w are some fixed strings and there are no other strings of the type uk and qw. The final ensemble for the given ensemble is always the following.

$F_{M'}(usw)= F_M(usw)+n$, $F_{M'}(uk)=0$, $F_{M'}(qw)=m-n$ if $n<m$.

$F_{M'}(usw)= F_M(usw)+m$, $F_{M'}(uk)= n-m$, $F_{M'}(qw)=0$ if $m<n$.

2. Let the initial ensemble be.

$F_M(uk)=n$, $F_M(qw_1)=m_1$, $F_M(qw_2)=m_2$,… $F_M(qw_l)=m_l$,

where u, w_1, w_2,…, w_l are some fixed strings with $\sum\limits_{i=1}^{l} m_i \leq n$, and there are no other words of the type uk and qw. The final ensemble for the given ensemble is always the following:

$F_{M'}(qw) = m - \sum\limits_{i=1}^{k} n_i$, $F_{M'}(u_1k)=0$, $F_{M'}(u_2k)=0$,…, $F_{M'}(u_lk)=0$,

$F_{M'}(u_1sw)=n_1$, $F_{M'}(u_2sw)=n_2$,…, $F_{M'}(u_lsw)=n_l$.

Notice that condition (1) is a special case for (2) or (3) with respect of whether n is more than m and vice versa. If these conditions are not met, then we can indicate at least two different ensembles, which are final for the given program. Let the initial ensemble be the following:

$F_M(uk)=n$, $F_M(qw_1)=m_1$, $F_M(qw_2)=m_2$,…, $F_M(qw_l)=m_l$, Where *u, w*$_1$*, w*$_2$*,…, w*$_l$ are some fixed strings with $\sum_{i=1}^{l} m_i \geq n$, and there are no other words of the type *uk* and *qw*. And for definiteness $\forall j=1 \ldots l$ $\sum_{i=1}^{l} m_i - m_j \leq n$. We indicate two possible final ensembles:

3. $F_{M'}(uk)=0$, $F_{M'}(qw_1)=\sum_{i=1}^{l} m_i - n$, $F_{M'}(qw_2)=0,\ldots$, $F_{M'}(qw_l)=0$, $F_{M'}(usw_1)=m_1-\sum_{i=1}^{l} m_i + n$, $F_{M'}(usw_2)=m_2,\ldots, F_{M'}(usw_l)=m_l$.

4. $F_{M'}(uk)=0$, $F_{M'}(qw_1)=0$, $F_{M'}(qw_2)=\sum_{i=1}^{l} m_i - n,\ldots$, $F_{M'}(qw_l)=0$, $F_{M'}(usw_1)=m_1$, $F_{M'}(usw_2)=m_2-\sum_{i=1}^{l} m_i + n,\ldots, F_{M'}(usw_l)=m_l$.

In the case when condition (3) is not met, non-determination is proved in a similar way. Hence, the determination is proved in a similar way. Therefore, the determination criterion for the program consisting of a single synthesis command is proved.

Example (non-deterministic synthesis)

$uK+Qw \rightarrow uSw$. $F_M(AK)=1$, $F_M(QB)=1$, $F_M(QA)=1$

Ensemble *M* has one string of the type *Qw*. Therefore the ensemble does not meet any of the criterion conditions. Hence, that synthesis is non-deterministic. Truly, here there are two allowable incompatible events. Two different ensembles can be final:

$F_{M1}(ASA)=1$, $F_{M1}(QB)=1$, and

$F_{M2}(ASB)=1$, $F_{M2}(QA)=1$.

Example (deterministic synthesis)

$uK+Qw \rightarrow uSw$. $F_M(AK)=3$, $F_M(QB)=1$, $F_M(QA)=1$

The given ensemble meets condition (2) of the criterion hence synthesis is deterministic. Final ensemble is ensemble *M* such that

$F_M(AK)=1$, $F_M(ASB)=1$, $F_M(ASA)=1$

3.3. Replacement

Consider a program consisting of a single command of the type $uvw \rightarrow usw$.

Applicability

The program is applicable to ensemble *M* if there exist such strings $u, w \in L^*$ that $F_M(uvw) > 0$.

Finiteness

Replacement does not change the number of strings in an ensemble, it is finite if $v \not\subset s$.

Determination

The conditions of replacement determination are similar to those of disintegration determination. Also we are to check if the strings *v* can be decomposed into substrings of the type *aba* and if strings of the type *ababa* are included into strings of the initial ensemble or an ensemble obtained from the initial one.

Replacement determination criterion

Theorem 4. Replacement program is non-deterministic if there exists a division of string *v=aba* and at least one of the following conditions is met.

1. $\exists u,w$ such that $F_M(uababaw)>0$;
2. $s=xababay$, where x, y are fixed strings
3. $s=ds_1$, $\exists\ u_1,w$ such that $F_M(u_1cvw) > 0$,where $cd=ababa$;
4. $s=s_1c$, $\exists\ u,w_1$ such that $F_M(uvdw_1) > 0$, where $cd=ababa$;
5. $s=d$, $\exists u_1,w_1$ such that $F_M(u_1c\ v\ ew_1) > 0$, where $cde=ababa$.

Otherwise, replacement is deterministic.

3.4. Summary.

The commands of disintegration and replacement are non-deterministic, if conditions 1&2 or 1&3 of the following list are fulfilled.

1. The string *v* to be replaced can be decomposed as *aba*, i.e. its beginning and end coincide.
2. In strings, to which these commands are applicable, there are the strings of the kind *ababa*, i.e. the string *v* can be chosen in two ways.
3. In the strings obtained after application of these commands there are strings of the kind *ababa*, i.e. the string *v* can be chosen in two ways.

For the programs consisting of any number of commands of disintegration and replacement, these criteria are easily generalized.

A program consisting of commands of synthesis is always finite, and in general cases non-deterministic.

4. Algorithmic universality of the Kirdin kinetic machine

We assume that the Kirdin kinetic machine will be of a universal character. Thus a question arises: how the Kirdin kinetic machine correlates with consecutive standard algorithmic formal models.

Theorem. *The deterministic Kirdin kinetic machine is equivalent to any consecutive standard algorithmic formal model, such as the Turing machine or Markov normal algorithms. Hence, it is an effective calculator.*

Proof. Let us build Kirdin kinetic machine modelling an arbitrary Turing machine. Since the Turing machine is deterministic and sequential, Kirdin kinetic machine will be deterministic. So for every moment of time there will exist only one allowable event. A Kirdin kinetic machine alphabet is $L=A\cup Q$. An initial ensemble consists of a single string corresponding to the standard initial configuration of the Turing machine $M_0=\{\ q_0\alpha\ \}$, where $\alpha\in A^*$. Turing mating command modelling can be described with a table with a command of the Turing machine in its first column and the corresponding system of Kirdin kinetic machine commands in its second column.

The TM command	KKM command
$q_i a_j \to q_i' a_j' R$	$uq_i a_j w \to ua_j' q_i' w$ $ua_j' q_i' \to u\, a_j' q_i' \lambda$
$q_i a_j \to q_i' a_j' L$	$ua_1 q_i a_j w \to uq_i' a_1 a_j' w$ … $ua_m q_i a_j w \to uq_i' a_m a_j' w$ $q_i a_j w \to q_i' \lambda a_j' w$
$q_i a_j \to q_i' a_j' E$	$uq_i a_j w \to uq_i' a_j' w$

A program for the deterministic Kirdin kinetic machine consists of replacement commands. Consider the modelling of the deterministic Kirdin kinetic machine consisting of replacement commands by the Markov normal algorithms. All commands *uvw→usw* of the Kirdin kinetic machine are replaced by rules *v→s* of the Markov algorithm. In the end of algorithm we add the rule $\lambda \to \cdot\lambda$, where λ is nil symbol. The theorem is proved.

The halting problem for the Turing machine is unsolvable in the general case, *hence finiteness the Kirdin kinetic machine for the programs consisting of commands of replacement is also unsolvable.*

The Kirdin kinetic machine is non-deterministic in the general case. It is natural that it cannot be completely equivalent to the deterministic calculator. Nevertheless, we have seen that it completely includes all deterministic universal calculators. What can be said about the Kirdin kinetic machine in the non-deterministic case? We shall distinguish another class of the Kirdin kinetic machine.

We will call the Kirdin kinetic machine *partially deterministic*, if its program consists of deterministic commands of replacement and disintegration and any commands of synthesis.

A partially deterministic Kirdin kinetic machine can be modeled by a specially arranged system of algorithmic calculators, for instance, let them be the Turing machines.

Consider a partially deterministic Kirdin kinetic machine, $|M|$ is the number of strings in its ensemble. From the beginning $|M|$ Turing machines are initialized, each of them processes its own string. The program for each machine consists of commands of replacement and disintegration. The application of a command of disintegration means initiation of a new Turing machine, and the configuration of the first of them corresponds to the string *uf*, and the configuration of the second – to the string *gw*.

At the same time, an *over-calculator* is functioning, with a program consisting of all commands of synthesis of the initial program. It compares configurations of Turing machines with strings *uk* and *qw*. If both are present in the configurations, it “switches off” one of these machines, and the configuration of the another turns into *usw*.

5. Unstructured memory

Unstructured memory is an organic elementary application of the Kirdin kinetic machine. Its basic idea is to store the information about a long text by means of a special dictionary, consisting of strings which length is much shorter than the length of the initial text. The list of all strings of length *q*, included in the given text, referred to as *q*-carrier of the given text. Strings starting from any place in the text are considered. For a text of the length *N* there are *N-q+1* of such strings. If each string of the *q*-carrier is put into correspondence to the frequency of its occurrence in the text, we obtain the frequency dictionary of length *q*.

Transition from the text to its frequency dictionary is a useful technique, which allows comparing texts of different lengths and performing their information analysis, which was successfully made for genetic texts in [11]. Besides, the frequency dictionary fixes the information about the text in a set of small objects – strings with their frequencies, which can be stored separately, "in a heap". There exist probabilistic estimations of the length of the dictionary, sufficient for the text unambiguous reconstruction.

If a dictionary of the length *d* contains strings, which occur uniquely, then for the dictionary of the length *d+1* and larger the text is restored unambiguously. This very case will be considered. The following program for the Kirdin kinetic machine constructs the dictionary of the length *k* from an initial (long) text. This program is finite and deterministic. If the final ensemble will consist of an unique string – !, then the obtained dictionary does not contain complete information about the initial text. It means, that such length of the dictionary is insufficient for the unambiguous reconstruction of the initial text. Hence, the given procedure should be started anew for the initial text, but with *k* increased by 1. And so on, until we obtain a dictionary of length *k* as final ensemble, and now it is necessary to start the program for the last time, to construct the dictionary of length *k+1*, from which the initial text can be reconstructed unambiguously.

Introduce a new designation: v^k in the left part of a rule denotes an arbitrary string of the length *k* in the initial alphabet. All entries of v^k in one rule denote the same string. Entries of the symbol v^k in different rules are not connected.

$uv^1v^{k-1}v_1^1w \to uv^1v^{k-1} + v^{k-1}v_1^1w$

$v^k + v^k \to !$

$! + v^k \to !$

Now, storing and, probably, transferring the initial text through communication channels as the dictionary, we can always unambiguously reconstruct it from this dictionary. The following program of the Kirdin kinetic machine is intended for this purpose.

$uv^k + v^kw \to uv^kw$

6. Structural programming in structureless parallelism

A natural necessity arose from programming for the Kirdin kinetic machine to create an analogue of subprograms in usual programming. The subprograms can be useful for conveyer programming and for encapsulation of a piece of an ensemble with its own program.

A processed unit for an enlarged Kirdin kinetic machine is at this time an ensemble of strings from a certain alphabet L. At the moment of initialization a number of employ cells for subprograms appears. For every cell an identificator is created. It consists of symbols from the alphabet I (with $L \cap I = \varnothing$) and positive integer number – i.e. this cell arity. For example, Cell(3) – a cell with the Cell identificator and 3 arity. Arity corresponds to the number of position for input string – parameter of a cell.

The processing consists of an aggregate of elementary events, which occur non-deterministically, in parallel and are regulated by rules or commands. The commands of disintegration, synthesis and replacement for the basic program effect the ensemble without cells. To exchange strings between a subprogram (cell) and the ensemble there are special commands in a program. They are the commands of conditional adsorption and desorption.

Adsorption: $Cell()_i+uvw\rightarrow Cell(uvw)_i$

$Cell()_i$ is the *i*-th vacant position in the *Cell.* Expression $Cell(uvw)_i$ means the *i*-th position is taken by the string *uvw*.

When the program starts working the strings of the $Cell()_i$ type are initiated for all input strings of all subprogram cells. One of commands of the type $Cell()_i+uvw\rightarrow Cell(uvw)_i$ having been applied, the *i*-th input position of the *Cell* is taken until it is released from the subprogram *Cell* as demonstrated below. Then the string $Cell()_i$ again appears in the ensemble.

Every cell is associated with a subprogram consisting of disintegration, synthesis and replacement commands. The subprogram effects only the strings occurring in this cell. Input cell parameters can be used with the release of an input parameter position or without it. For the use of an input parameter without a position release a special command appears similar to the synthesis command of the type

$In(uk)_i+qw\rightarrow usw+ In(uk)_i$ or $uk + In(qw)_i \rightarrow usw+ In(qw)_i$.

For the use of an input parameter with a position release the command $In(uvw)_i \rightarrow uvw+ In()_i$ is applied. Application of the command formally means the appearance of the string $Cell()_i$ in the basic ensemble of the Kirdin kinetic machine.

Desorption is initiated out of a cell/ The right part of any subprogram command can carry *Out(usw)* string for synthesis command or *Out(uf)* or *Out(gw)* strings for disintegration commands. The application of such a command means that the string *s* from the *Out(s)* ia taken from a cell and goes to the basic ensemble of the Kirdin kinetic machine.

The following types of information processing in cell are possible:

1.Sequential processing:

- A necessary set of strings goes into a cell
- Processing takes place inside the cell
- The modified set of strings goes out of the cell
- The cell is ready to accept a new set of strings

2.Parallel processing:

- A cell is constantly ready to accept new strings which are included into the process occuring inside the cell
- From time to time the cell discharges strings-results of its perfomance into the basic ensemble.

3.Conveyer processing:

- New strings constantly enter a cell
- Strings are processed without any interaction between themselves

- After processing the strings goes out of the cell independently

The example of a program with subprograms

*Problem: an ensemble consists of a set of patterns-strings from the alphabet L. A detected string is added to it and indicated with the symbol * in the beginning (with* $* \notin L$*). The problem is to find out if the string coincides with a pattern.*

Program:

Main: $Reverse_1()+*w \to Reverse_1(*w)$

$Compare_1()+u* \to Compare_1(u*)$

$Compare_2()+u^{(L)} \to Compare_2(u^{(L)})$

Reverse(1): $In_1(*w) \to In_1(*w)+*w$

$u*v^1w \to uv^1 \times *w$

$uv_1^1v^1 \times w \to uv^1 \times v_1^1w$

$v^1 \times w \to v^1w$

$v^{(L)}* \to Out(v^{(L)}*)$

Compare(2): $In_2(u^{(L)}) \to In_2()+u^{(L)}$

$In_1(u*)+v^{(L)} \to u*v^{(L)}+ In_1(u*)$

$uv^1*v^1w \to u*w$

$* \to Out(*)$

$uv_1^1*v_2^1w \to u \times w$

$uv^1*w \to u \times w$

$u*v^1w \to u \times w$

$\times+w \to w$

This program is finite and deterministic. Subprogram *Reverse(1)* realize sequential processing. Subprogram *Compare(1)* realize conveyer processing. The answer to the problem is “yes” if final ensemble consists of one or several * symbols, “no” if final ensemble is empty.

Conclusions

The Kirdin kinetic machine is based on two paradigms:

- fine-grained parallelism
- structureless parallelism

These seem to be the most perspective directions of the development of computer science.

We have seen that the Kirdin kinetic machine is a universal calculator. The ways of solution of the problem of program execution correctness for the Kirdin kinetic machine are offered. Determination of finiteness for the Kirdin kinetic machine is very complicated and, in the general case, unsolvable. But the same is known about the Turing machine.

Determinacy means definiteness of the result. Most likely, for some range of problems we will not be interested in strict determinacy, but in near determinacy or even simply probabilistic distributions of the final ensemble.

According to the well-known “Minsky hypothesis” the efficiency of a parallel system increase proportionally to logarithm of processor number. To overcome this restriction the following approach often applied. Extremely parallel algorithms of solutions are built for different types of problems. The algorithms use some abstract paradigm of fine-grained parallelism, for example, structureless parallelism. For particular parallel computers means of parallel processes realization with a given abstract architecture are created. As a result an effective tool for parallel program production appears.

References

1. Kirdin A.N. Ideal ensemble model of parallel computations // Neural informatics and its applications. Abstracts of V all-Russia seminar. – Krasnoyarsk, KGTU, 1997. p.101: in Russian.
2. Gorbunova E.O. The analysis of elementary programs for ideal ensemble model of parallel computations // Abstracts of INPRIM-98. – Novosibirsk: izd-vo Instituta matematiki, 1998. – p.77: in Russian.
3. Gorbunova E.O. Finiteness and determinacy of simple programs for the Kirdin kinetic machine // Methods of neuroinformatics. / A.N.Gorban (ed.). Krasnoyarsk; KGTU. – 1998.– p. 23-40: in Russian.
4. Gorbunova E.O. To the question of algorithmic universality of the Kirdin kinetic machine// Neuroinformatics and its applications. Abstracts of VI all-Russia seminar. – Krasnoyarsk, KGTU, 1998. p.147-48: in Russian.
5. Markov A.A., Nagorny N.M. The theory of algorithms. – M.: Nauka, 1984. 432 p.: in Russian.
6. Uspensky V.A., Semenov A.L. The theory of algorithms: basic discoveries and applications. –M.: Nauka. 1987.– 288 p.: in Russian.
7. Gorban A.N., Rossiev D.A. Neural networks for personal computer. – Novosibirsk: Nauka, 1996. 276 p.: in Russian.
8. Latkin E.I. SCAM: chemical computer // The theory of computations and languages of specifications. – Novosibirsk, 1995.– V. 152: Computing System. –p.140-151: in Russian.
9. Achasova S., Bandman O., Markova V. and Piskunov S. Parallel substitution algorithm. Theory and Application.– WORD SCIENTIFIC, 1994.
10. Dasgupta D.(Ed.). Artificial immune systems and their applications.– SPRINGER, 1998. XIV, 310 p.
11. Bugaenko N.N., Gorban A.N. and Sadovsky M.G. Maximum entropy method in analysis of genetic text and measurement of its information content – Open Sys. and Information Dyn. 5, 1998. – p.265-278.
12. Jabotinskiy A.M. Concentration Selfooscillations. – Moscow: Nauka, 1974: in Russian.
13. Gorban A.N., Bykov V.I., Yablonsky G.S. Essays about chemical relaxation. – Novosibirsk: Nauka, 1986: in Russian.

14. Liberman E.A. Cell as molecular computer (MCC). – Biophysics. Volume XVII, No 5, 1972.