Triangle: Language, Literature, Computation, n. 6, 2011
Publicacions Universitat Rovira i Virgili · ISSN: 2013-939X
https://revistes.urv.cat/index.php/triangle

On the Concepts of Parallelism in Biomolecular Computing

Remco Loos[1], Benedek Nagy[1,2]

[1] Research Group on Mathematical Linguistics
Rovira i Virgili University
Tarragona, Spain
E-mail: remcogerard.loos@estudiants.urv.es

[2] Faculty of Informatics (Computer Science and Information Technology)
University of Debrecen,
Debrecen, Hungary
E-mail: nbenedek@inf.unideb.hu

Summary. In this paper we consider DNA and membrane computing, both as theoretical models and as problem solving devices. The basic motivation behind these models of natural computing is using parallelism to make hard problems tractable. In this paper we analyze the concept of parallelism. We will show that parallelism has very different meanings in these models. We introduce the terms 'or-parallelism' and 'and-parallelism' for these two basic types of parallelism.

1 Introduction

Over the last decade, molecular computing has been a very active field of research. The great promise of performing computations at a molecular level is that the small size of the computational units potentially allows for *massive parallelism* in the computations. Thus, computations that are intractable

in sequential modes of computation can be performed (at least in theory) in polynomial or even linear time.

In this paper, we investigate the way parallelism is used in different models of molecular computation. We are interested in the role parallelism plays in theoretical models (that is in the language-generating devices) as well as in the way parallelism is employed to solve computationally hard (typically NP-complete) problems.

We focus on two branches of molecular computing, DNA computing and membrane computing. In the future, this work could be extended to other models of molecular computation, such as forbidding-enforcing systems, as well as other bio-inspired models of computation, like cellular automata and neural networks.

The field of DNA computing was instigated by Leonard Adleman's 1994 paper [1], in which he reports a molecular solution of an instance of an NP-complete problem. Since then, much work has been done in this area, covering both experimental work and the formulation of formal and computational models. These models typically represent DNA strands as strings and model biochemical operations by string rewriting rules. The reader is referred to [11] for a detailed overview of the main computational models of this type.

Membrane computing is an area of molecular computing initiated by Gheorghe Paun [8, 10]. A membrane system (also called P system) is a computing model inspired by the structure of a living cell. A membrane structure defines regions where objects evolve according to given rules. From this basic structure, many different computational devices can be defined, depending on the objects used (strings, symbols), the types of rules allowed and the way the generated language is defined. Gheorghe Paun's book [9] is a good introduction to the most important types of membrane systems.

We will first analyze the nature of parallelism (see also [7]) in the formal computational models of both areas. Next, we examine the ways parallelism is used in both cases to solve computationally hard problems.

We will try to avoid considering specific models. Instead we focus on the nature of the parallelism present in them, which we will see to be common to most models in the considered area.

2 Parallelism in DNA computing

The field of DNA computing considers molecular computing in a variety of ways, which range from purely theoretical computational models to more practical 'molecular algorithms' to actual experimental implementations of molecular computations. The theoretical models include different types of systems such as splicing systems, sticker systems and deletion-insertion systems. Details about these systems can be found in [11]. Here we do not consider Watson-Crick automata, which is not a parallel device: rather it is based on the inherent power of Watson-Crick complementarity. We claim that despite the different levels of abstraction and the different models, essentially the same type of parallelism underlies all systems of DNA computing.

In experimental DNA computing, the working assumption is that all molecules are present in such huge quantities that they can be considered infinite. All formal models considered share this assumption: We start from a (generally finite) initial language L, with all words $w \in L$ present in arbitrarily many copies. Similarly, in experiments, a series of biochemical operations is applied sequentially, but each biochemical operation applied affects all molecules present. This is reflected in the theoretical models, where each word is rewritten sequentially, but this sequential rewriting is applied to all words in parallel. In this way, one computation gives all possible solutions. In the context of language generating systems, we can say that one 'run' of the system gives the entire generated language. This is an important difference with respect to most known models of computation, such as the Turing machine or Chomsky grammars, where one run of the system accepts or generates just one word in the language.

As an example of this type of parallelism we consider a molecular algorithm presented in [4]. Although this is a theoretical algorithm, experimental implementations of this or very similar algorithms are reported in [3], [4] and [5]. In addition to exemplifying parallelism, this example also shows how this parallelism can be used to reduce the computational complexity of NP-complete problems.

The algorithm we consider solves the satisfiability problem for disjunctive clauses (SAT). Suppose the Boolean variables of the formula are p_1 to p_n and the number of clauses is m. Suppose, moreover, we have a molecule with $2n$ sites on which we can 'write' (i.e. change) the site in such a way it can later be recognised or 'read' as being written. An unwritten site

is interpreted as 1 (true) and a written one as 0 (false). We start with a single molecule which encodes $2n$ 1's. We interpret this as the values of $p_1, \neg p_1, p_2, \neg p_2, ...p_n, \neg p_n$.

Now, the algorithm for solving SAT is the following:

1. For each variable y, divide the solution into two parts. In one part, write the site for y. In the other part, write the site for $\neg y$. This yields all consistent assignments of variables.
2. For each clause, divide the content of the test tube. If for instance the clause is $(p_i \vee \neg p_j)$, we divide it into two parts. In one part, remove all molecules which have $\neg p_i = 1$, in the other, those which have $p_j = 1$. Thus only molecules which satisfy this clause remain.
3. Check if a molecule remains. If so, the answer is 'yes', otherwise 'no'.

We see that all the solutions are generated and checked in parallel, and in this way we can simultaneously explore all options, and solve the instance of SAT in $O(n + m)$ biochemical operations (i.e. in linear time), assuming each operation takes constant time.

Fig. 1 shows a sketch of the DNA computing method. In this example there are three kinds of original DNA-molecules with sticky ends. Two of these kinds can make a new molecule. The result can be seen in the figure: long molecules without sticky ends (for instance, the last one). The third kind of molecule can also make new molecules. For example, the second molecule is the result of their reaction. Technically, by amplifying we can multiply the number of present; the assumption is that there are enough molecules of each kind in the sup. We assume that all possible ways of continuing the computation are present in the same time.

3 Parallelism in membrane computing

A Membrane system (also called P system) is a computing model which abstracts from the way living cells process chemical compounds in their compartmental structure. This amounts to a membrane structure that defines different regions which evolve according to given rules. The objects can be described by symbols or by strings of symbols (in the former case their multiplicity matter, (that is, we work with multisets of objects placed in the regions of the membrane structure); in the second case we can work

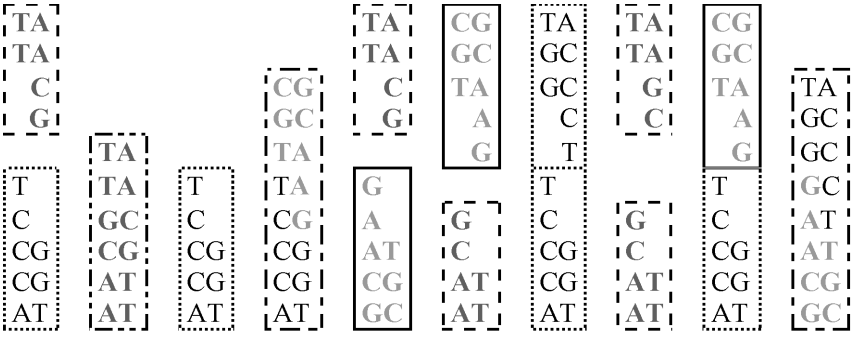

Fig. 1. Several DNA molecules of the same type and all possible ways of continuing the computations are present

with languages of strings or, again, with multisets of strings). By using the rules in a nondeterministic, maximally parallel manner, one gets transitions between the system configurations. A sequence of transitions is a computation. With a halting computation we can associate a result, in the form of the objects present in a given membrane in the halting configuration, or expelled from the system during the computation. Various ways of controlling the transfer of objects from one region to another and of applying the rules, as well as (using so-called active membranes:) possibilities to dissolve, divide or create membranes were considered.

Many of these variants lead to computationally universal systems, while several variants with enhanced parallelism are able (at least theoretically) to solve NP-complete problems in polynomial (often, linear) time, by making use of an exponential space.

Parallelism can be controlled through cooperative rules, catalysts, etc. A simple membrane computer with symbol objects can be seen in Fig. 2.

Let us see how a membrane system can generate languages. First, as the most usual case the so-called multiset languages are considered. The membrane system in Figure 2 starts with a copy of the objects a and b in membrane 1. In any subsequent configuration (except the halting one) there is also exactly 1 copy of a and b in membrane 1. If a or b is processed by the rule sending symbols into membrane 2 the other symbol should be processed by the same type of rule, for maximal parallelism. In these

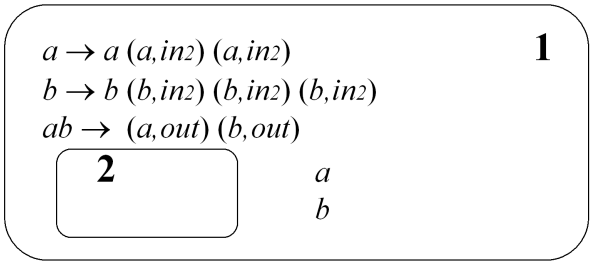

Fig. 2. A cooperating membrane system

parallel steps two copies of a and three copies of b appear in membrane 2. No rule is available in membrane 2 so all copies inside still remain with no change. The process continues until the cooperative rule is used to send both a and b out of membrane 1. The computation halts with this step. A word of the generated language is in membrane 2 (we consider it as the output membrane). This membrane contains objects $a^{2n}b^{3n}$ depending on the length of the process. So, the generated multiset language as a Parikh-set is $(2n, 3n)$.

As we have seen in the membrane system, parallelism is inside the computation of a (multiset) word. The result of a computation process is a particular word of the language, and it can be any because of non-determinism. To generate the whole language, we need to restart the computation many times (usually infinitely many times). So, the power of parallelism is used inside the computation of a word: and this means that each word can be computed quickly and effectively.

Sometimes the objects sent out of the system build the result word (traces). Parallelism has the same role here: the whole system constructs only a very particular part (a word or words whit the same Parikh-vector) of the language in a run.

The basic idea of parallelism is the same whenever we consider membrane systems using catalysts, evolution rules, priorities, cooperative rules, symport, antiport, electrical charges, dissolutions, creating and/or dividing membranes. The only difference is that by using active membranes one can

dynamically play with the structure of the system as well. With more membranes one can easily organize the derivation process because the membranes can have various rule sets. The creation and division of membranes enables independent computations to be performed in parallel way, too.

Now let us analyze how membrane systems can be used for effective problem solving. We briefly describe how membrane creating can be used to solve SAT in linear time. First we have an initial membrane with only one object. Applying the only applicable rule for this object we introduce two new objects corresponding to the possible values of the first variable, and a technical object to continue the process. Then the new objects of the logical variable create new membranes (and copy some symbols to the new membranes). Now for each new membrane two new objects are introduced for the next variable. These new objects create new ones again, etc. Finally the membrane structure forms a complete n-level binary tree. Each path from the initial to a leaf-membrane represents a possible truth-assignment. Now, each membrane in the n-th level computes objects for satisfied clauses of the SAT formula. (This can be done easily by a comparing the literals of the clauses and the given truth-assignment of the membrane.) Using a cooperative rule a special symbol is sent out if all clauses are satisfied in a membrane. In the next step the previous level membranes forward these symbols. Therefore, this special symbol moves up all n levels, and finally leaves the system and terminates the process with answer 'satisfiable'. More technical details can be found in [9].

In this process the power of parallelism builds up a complete tree by levels in linear time. In each membrane, at the deepest level there are rules for each clause, so clauses can be evaluated in a parallel way. (Here parallelism is used in the same way as in language generation.)

4 A brief comparison

Now we describe some essential differences between DNA and membrane computing.

In DNA-computing we assume that there are (theoretically) infinitely many copies of each string of the initial language. We also assume that infinitely (arbitrarily many) many copies of each string can appear in the molecular soup and, therefore, all possibilities are explored. In a mathematical model, we can use sets to describe the configuration of the system. The assumption is that the 'space' is already given.

In contrast membrane computing multisets. The numbers of the present objects are very important. Membrane computing use 'active membranes' (division, creation, etc). These allow all options to be explored, space traded for time, etc. In a way, an extra level of parallelism is needed for efficiency, the parallelism of rule application does not play any significant role.

In DNA computing we can try all possibilities (maybe in a clever order, to reject the false ones as soon as possible) to have a solution. If there is any, then a (some) try will be successful.

5 Notions of parallelism

Abstracting from the specific systems we can identify two essentially different notions of parallelism. We classify those here.

In the 'and-parallelism' the computation needs several branches. These branches provide some subresults. Typically, these involve rewriting in parallel way, but there is a dependence between parallel computations. For instance, a parallel computation generates one word. It affects both computational power and complexity. The non-deterministic massive parallel way of applying the rules in the membrane system fits this notion exactly.

The 'or-parallelism' (which we could also call 'Chinese army parallelism') is the following. The parallel branches independently try to solve the problem. Any of them can produce the solution.

Computations are independent and they are performed in parallel. One example is parallelism, which generates all words simultaneously. This type of parallelism only affects complexity. We assume that the space can be considered arbitrarily large. Parallelism in DNA computing, where each word is rewritten sequentially, but all words are rewritten in parallel is of this type. The use of active membranes in membrane computing also provides this kind of parallelism. Note that this notion is closely related to non-determinism in the usual sense. This concept of parallelism considers all the ways in which a non-deterministic algorithm can run at the same time.

6 Conclusions

In this paper we have analyzed two fields of natural computing, DNA computing and membrane computing, and studied the parallelism present in

them. Even though the parallelism is the main motivation for these fields and an important property of both computational systems and problem solving algorithms, it has never really been studied in itself. Looking at the way the systems in these areas work in terms of parallelism, we made two important observations. Firstly, in spite of the diversity of computational systems and even levels of description (experimental implementations, problem solving algorithms, language generating devices), we can still make general statements about the parallelism present in these fields, because the underlying notion of parallelism is essentially the same in all models. Secondly, we noted that the notion of parallelism used in DNA computing is very different from the parallelism present in all membrane systems. Whereas systems in DNA computing have parallel independent computations, in membrane systems the parallelism is applied in a dependent way, with all parallel computations taking place inside the rewriting of a single configuration. However, in membrane computing there exists a subclass of membrane systems called membrane systems with active membranes, which in addition to the parallelism present in all membrane systems allows for another type of parallelism, which is the same type of parallelism as in DNA computing. It is precisely this additional kind of parallelism that makes it possible to provide efficient solutions to computationally hard problems.

We have also described the concept of parallelism in computations in a more general way. There are two basic notions. In the so-called 'and-parallelism' the results of several parallel branches are needed to provide the result of the computation. We use parallelism for computation to be effective: parallel branches work somehow on the same 'state' or configuration of the system. In 'or-parallelism' (analogous to the so-called Chinese army algorithm) the branches are independent and any of them can produce the final result. We use parallelism because we do not know which branch will be successful. This concept can be imagined as a non-deterministic machine running in all possible ways at the same time.

7 Acknowledgements

A version of this paper is presented and appeared in [6]. The first author was supported by Research Grant TIC2003-09319-C03-01 of the Spanish Ministry of Education and Science. The second author was supported by the programme Öveges of NKTH, Hungary.

References

1. L.M. Adleman, *Molecular Computation of Solutions To Combinatorial Problems*, Science, 266: 1021-1024 (1994)
2. T. Head *Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors*, Bull. Math. Biology, 49, 737-759 (1987).
3. T. Head, X. Chen, M.J. Nichols, M. Yamamura and S. Gal, *Aqueous solutions of algorithmic problems: emphasizing knights on a 3X3*, in: DNA Computing - 7th International Workshop on DNA-Based Computers (N. Jonoska, N.C. Seeman eds.), Lecture Notes in Computer Science, v. 2340, Springer-Verlag, Berlin, 191-202 (2002)
4. T. Head, X. Chen, M. Yamamura and S. Gal, *Aqueous computing: a survey with an invitation to participate*, J. Computer Sci. & Tech. 17, 672-681 (2002)
5. T. Head, G. Rozenberg, R. Bladergroen, C.K.D. Breek, P.H.M. Lommerse and H. Spaink, *Computing with DNA by Operating on Plasmids*, Bio Systems 57, 87-93 (2000)
6. R. Loos and B. Nagy, *Parallelism in DNA and Membrane Computing*, CiE2007, Computability in Europe 2007: Computation and Logic in the Real World, Siena, Italy, (2007), 283-287.
7. B. Nagy, *On the Notion of Parallelism in Artificial and Computational Intelligence*, Proceedings of the 7th International Symposium of Hungarian Researchers on Computational Intelligence, Budapest, Hungary, (2006), 533-541.
8. Gh. Paun, *Computing with Membranes*, Journal of Computer and System Sciences, 61, 1, 108-143 (2000) and Turku Center for Computer Science-TUCS Report No. 208 (1998)
9. Gh. Paun, *Membrane Computing: An introduction*. Springer-Verlag, Berlin (2002)
10. Gh. Paun and G. Rozenberg, *A guide to membrane computing*, Theoretical Computer Science, 287, 73-100 (2002)
11. Gh. Paun, G. Rozenberg and A. Salomaa, *DNA Computing - New Computing Paradigms*, Springer-Verlag, Berlin (1998)