Active Knowledge, LuNA and Literacy for Oncoming Centuries

Victor Malyshkin[1,2(✉)]

[1] Lab of Parallel Program Synthesis, Institute of Computational Mathematics and Mathematical Geophysics Russian Academy of Sciences, Novosibirsk, Russia
malysh@ssd.sscc.ru
http://ssd.sscc.ru

[2] Chair of Parallel Computing, National Research University of Novosibirsk, Novosibirsk, Russia

Abstract. The concept of active knowledge implementation on the basis of the theory of structural program synthesis, modern technologies and their necessary developments are considered. The theory is proposed for technological description, accumulation, keeping, processing and application of active knowledge. On this basis the notion of literacy for the future is suggested. The concept was implemented in the frame of the LuNA project aimed at elimination of parallel programming from the process of large-scale numerical models development.

Keywords: Active knowledge · Program synthesis · Active knowledge base · Structural program synthesis · Program automatic construction · Literacy · LuNA fragmented programming system

LITERACY is the capability of a human being to describe, to keep, to understand and to apply the knowledge.

1 Introduction

During last years, the computing technologies have substantially changed our life. In this simple technological paper we would like to look at new progressive technology – technology of active knowledge – that can be partially implemented right now basing on the current scientific knowledge, accumulated in scientific computations, theoretical models of parallel computations [1–6], technological results and current system software that are now in use or under development [7–16].

The approach to a possible pragmatic application of the theory of structural program synthesis [6] to the development of the active knowledge technology is discussed. The sum of the current theoretical results provides understanding of the approaches to the solution of active knowledge problem. It is interesting now

C. Bodei et al. (Eds.): Degano Festschrift, LNCS 9465, pp. 1–12, 2015.
DOI: 10.1007/978-3-319-25527-9_19

to look at how the problem could be solved and technologically[1] implemented. In this technological paper the problem of the current implementability of the active knowledge technology is considered.

2 Active and Passive Knowledge

Principal idea. The problem of active knowledge technology creation was principally solved in the frame of the theories of program synthesis. Below the technologically implementable approach, based on the theory of structural program synthesis, that is ready for implementation right now, is mostly discussed.

2.1 Passive Representation of the Knowledge

There are now four main problems to utilize the knowledge represented in passive form:

1. Currently existing phonetic systems of knowledge description, keeping and processing are actually the systems of *passive* knowledge (texts, movies, etc.) representation, i.e., this knowledge description cannot be applied directly, automatically. Keeping in the hands a book(s) with the complete enough description of the technology of a bridge construction, a desired/specified bridge cannot be constructed automatically even after the magic incantations were pronounced. This is so, because a computer doesn't understand phonetic text and unable to extract the knowledge from it.
2. In order to utilize such passive knowledge, the people should adopt (read, understand and be able correctly to apply) a large body in the literature of the subject. It takes generally about 25 and even more years. Also, the technologies are now permanently under modifications and the process of learning is being permanent.
3. A big volume of passive knowledge is already accumulated. Sometimes the passive knowledge is not in (intensive) use and even can be forgotten and lost.
4. Also application programs, implementing passive knowledge, are usually developed as "black box" and cannot be automatically modified in accordance with a user demands.

2.2 Active Representation of the Knowledge

In order to overcome the above mentioned problems, the knowledge should be represented in *active* form that can be understood by a computer system and applied automatically to solution of a certain specified problem. Active knowledge representation should provide knowledge extraction from its description and the knowledge use for solution of a certain application problem [13].

[1] The term *technological* is used in order to denote that a specified object (theory, model, problem, algorithm, program, etc.) can be implemented for practically acceptable time and with necessary properties, with the use of acceptable number of resources.

Mathematical logic describes the way to solve the problem of active knowledge representation, understanding and application in four steps:

– development of complete enough axiomatic theory (AT),
– application problem formulation in terms of the theory,
– derivation of a desirable algorithm for application problem solution (APS),
– generation of a program, implementing derived APS algorithm.

2.3 Logic Program Synthesis

The method is based on the sufficiently complete AT, describing an object domain. Within the frame of this theory the necessary assertions are proved or disproved from the set of axioms.

The fact is, the method is technologically not implementable, at least because of too high complexity of APS algorithms derivation and the absence of a possibility to provide desirable pragmatic properties both derived APS algorithms and implementing programs.

2.4 Technological Requirements to Representation

Technological description of AT should provide:

1. In order to avoid high complexity of APS algorithm derivation, an AT should be described partially [14], only the necessary APS algorithms should be derivable in the frame of the AT. This de facto means, that AT (knowledge base of the partial object domain) description should be generated from a problem formulation.
2. An AT is defined partially not taking care of completeness of the AT. Practically it means, that as a rule, an AT should be defined for solution the only problem or, may be, for solution of few problems.
3. Derived APS algorithms should be represented as a set of recursively countable set of functional terms [17]. Therefore, a language of partial AT description should be a language for description of sets of all operations of functional terms with their input and output variables. This language can be understood by a computer. This is the basis for creation of new writing and literacy. Ability to operate with AT will define soon the literacy of human being.
4. An AT description should be easy extendable, in order to include into the AT the descriptions of the other problems solutions when the need arises. It should be simple technological (not scientific) work.
5. Program, implementing derived algorithm, should be generated automatically and possess by all the desirable properties (arbitrary program is not acceptable), like:
   – to be executed in parallel;
   – to be tunable (statically and dynamically) to all the available resources of a computer system;
   – able to be executed on heterogeneous distributed computer system;

– with dynamic workload balancing;
– to have the demanded level of reliability;
– to be modified on a user demand,

and many other pragmatic properties.

3 Technological Notion of Knowledge

It is necessary to define a technologically implementable method to describe, to accumulate and, what is the most important, correctly to exploit the knowledge, represented in the active form, without knowledge adaptation by a human being in all the thinkable details.

3.1 Structural Program Synthesis (SPS)

For the reason of high complexity of an APS algorithms derivation in the frame of logic programming we are forced to use in practice the methods with lesser possibilities for the knowledge description, but more suitable for computer processing. One of such methods is based on imposing a structure on the knowledge base that reflects the associative dependences between the objects of the object domain. In the process of an algorithm derivation, the explicit representation of these dependences in a computer permits to use the associative search instead of the random search in logic programming. The method of structural program synthesis [6] is based on this idea. Rephrased idea of structural program synthesis is:

– There is no any technological sense to develop the general theory for algorithm derivation from functional specification. A partially defined AT can be successfully used providing an acceptable quality both algorithms and implementing programs.
– A desirable program should be constructed out of accumulated well-developed ready-made modules each of which represents one step $((A, A \supset B) \supset B)$ of APS derivation.

In the method of structural program synthesis (program synthesis on the basis of computational models) the completeness of the theory is not considered at all. In principle, *the description of an object domain includes only that objects which were already implemented in practice with an acceptable*[2] *quality.* The method is based on a carefully designed partially defined object domain description, on such partial AT, in which only necessary problems are well solved. Often, in such partially defined AT, a solution of the only necessary problem is constructed.

The theory of structural program synthesis [6] satisfies all the above requirements. This method was applied to parallel implementation of different large-scale numerical models [10,16] and the necessary experience was gained, algorithms and programs were developed.

[2] The term *acceptable* is used in order to denote that a specified solution, quality, program, resources allocation, etc., can be used in practice because, for example, better solution doesn't exist or is not known now.

3.2 The Notion of Knowledge

Omitting unnecessary here details, the program module (procedure, subroutine, hardware block) will be considered to be *atomic unit* of an active knowledge. Such module represents one step $((A, A \supset B) \supset B)$ of APS derivation. The execution of this module will result in the *active knowledge application*. The algorithms, used into this module, implement the active knowledge. For instance, the knowledge of arithmetic is now implemented inside the module called *calculator*. From now nobody needs to know algorithms of arithmetic and mental arithmetic in order successfully to utilize the arithmetic.

A set of all the knowledge units of an object domain (library of modules) does not yet constitute the knowledge base, because on this set different relations exist, in particular, the relation of information dependencies or neighborhood relation. These relations should also be described and implemented in a generated program. Also for any module the (input) variables, to which only this module is permitted to be applied for computing another (output) variables should be described. Any module denotes the step of an APS algorithm derivation. Thus, knowledge base is actually a partially defined AT, represented by a set of steps of an APS algorithm derivation. An example is given below.

With this technological definition of the active knowledge, the main objects of knowledge processing will be algorithms and programs. Therefore, the well-known methods of APS algorithms derivation and program synthesis can be applied to the description and processing of the knowledge base [6].

4 Technological Model of Knowledge

Thus, let us consider shortly the computational model [6] as the basis for definition of the knowledge base.

4.1 The General Definition of the Program Synthesis Problem

The knowledge base (actually partially defined AT) in the method of structural program synthesis constitutes a set of ready-made modules (programs, subroutines). For each module the allowable sets of input and output variables are defined. The solution of a specified problem is assembled out of these modules if possible. If this is not possible, the AT is extended by adding new modules to the knowledge base. Also, if the solution of a specified problem does not satisfy the user, then some modules should be replaced by better modules or new modules be added.

The general formulation of the program synthesis problem is:
Given:

- class S of input problem specifications,
- class P of resulting programs,
- equivalency relation $\sim$ on P,
- quality relation $>$ on P, that satisfies the axioms of the partial order.

The algorithm $A : S \to P$ should be found such that:

– program $p = A(s), p \in P$, satisfies the specification $s \in S$,
– p is the best (in the sense $>$) program among $\{p \mid p = A(s)\}$.

In such a way, the algorithm $A : S \to P$ solves a program synthesis problem, that is, for every specification $s \in S$, the algorithm A constructs an element $p = A(s), p \in P$, which is the best program among all the programs solving a specified problem s.

4.2 Computational Model Definition

Given [6]:

– The finite set $\mathbf{X} = \{x, y, \ldots, z\}$ of variables for representation of different computed values;
– The finite set $\mathbf{F} = \{a, b, \ldots, c\}$ of functional symbols (operations, Fig. 1a), $m \geq 0$ is the number of input variables, $n \geq 0$ is the number of output variables;
– $in(a) = (x_1, \ldots, x_m)$ is a set of input variables, $out(a) = (y_1, \ldots, y_n)$ is a set of output variables (Fig. 1), if $i \neq j \to y_i \neq y_j \ \& \ x_i \neq x_j$.

Model $C = (\mathbf{X}, \mathbf{F})$ is called *simple computational model* (SCM). Operation $a \in \mathbf{F}$ describes the possibility to compute the variables $out(a)$ from the variables $in(a)$, for example, with the use of a certain procedure. The model can be graphically depicted (Fig. 1).

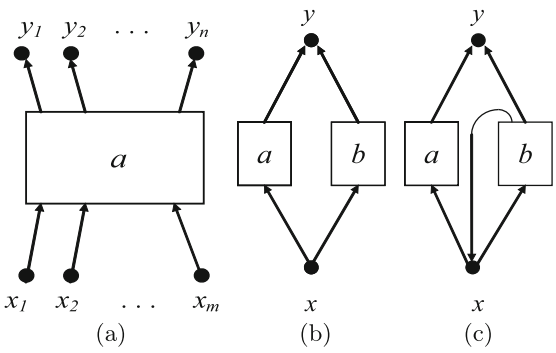


Fig. 1. Examples of operations, variables and model

Let $V \subseteq \mathbf{X}$, $F \subseteq \mathbf{F}$ be given. A set of functional terms $T(V, F)$ is defined as follows:

1. If $x \in V$, then x is a term $t, t \in T(V, F)$; $in(t) = \{x\}$; $out(t) = \{x\}$.

2. Let $\{t^1, \ldots, t^s\} \subseteq T(V, F)$ and $a \in F$, $in(a) = (x_1, \ldots, x_s)$ be given. The term $t = a(t^1, \ldots, t^s)$ is included into $T(V, F)$ if $\forall i(x_i \in out(t^i))$, $in(t) = \bigcup_{i=1}^{s} in(t^i)$, $out(t) = out(a)$. Here $t = a(t^1, \ldots, t^s)$ denotes that t is the term $a(t^1, \ldots, t^s)$.

A term is depicted as a tree that contains both operations and variables of the term.

We say that a term t computes a variable y if $y \in out(t)$. A set of terms $T(V, F)$ defines all the variables of the SCM that can be computed from V variables. A set of terms $T_V^W = \{t \in T(V, F) \mid out(t) \cap W \neq \emptyset\}$ computes all those variables from W that can be computed from V variables.

Any such subset $R \subseteq T_V^W$ that $\forall x \in W \exists t \in R(x \in out(t))$ is called (V, W)-plan. This (V, W)-plan defines an algorithm computing the variables W from the variables V. Here V and W denote the sets of input and output variables of the algorithm, respectively. Everywhere further a recursively countable set of functional terms is considered as a representation of an algorithm.

In order to satisfy all the technological requirements to AT representation, the axioms are not used, they are not formulated. Instead, AT is representation by the set of all possible in AT derivation steps. These steps (functions, modules) actually constitute the computational model.

Interpretation. Let $V \subseteq \mathbf{X}$ be given. *Interpretation* $\mathbf{I}$ in the domain D is a function that assigns:

- to every variable $x \in V$ an entry $d_x = I(x) \in D$, d_x is a value of the variable x in the interpretation $\mathbf{I}$,
- to every operation $a \in F$, $in(a) = \{x_1, x_2, \ldots, x_m\}$, $out(a) = \{y_1, y_2, \ldots, y_n\}$, a computable function $f_a : D^m \to D^n$,
- to every term $t = a(t_1, t_2, \ldots, t_m)$, a superposition of the functions is assigned in accord with the rule $\mathbf{I}(a(t_1, t_2, \ldots, t_m)) = f_a(\mathbf{I}(t_1), \mathbf{I}(t_2), \ldots, \mathbf{I}(t_m))$.

If $t = a(t_1, t_2, \ldots, t_m)$ is an arbitrary term, $in(a) = \{x_1, x_2, \ldots, x_m\}$, $out(a) = \{y_1, y_2, \ldots, y_n\}$, then $\mathbf{I}(out(a)) = val(t) = (d_1, d_2, \ldots, d_n) = f_a(val_{x_1}(t_1), val_{x_2}(t_2), \ldots, val_{x_n}(t_n))$.

Further it is assumed that for every function $f_a = \mathbf{I}(a)$ there exists a module (procedure) mod_a that can be used in a program to compute the function f_a.

Correct Interpretation. If there exist two different terms t_1 and t_2, $y \in out(t_1) \cap out(t_2)$, $in(t_1) \cup in(t_2) \subseteq V$, then $val_y(t_1) = val_y(t_2)$ in the interpretation $\mathbf{I}$, and the interpretation $\mathbf{I}$ is called *correct interpretation*. In the correct interpretation for any variable y, any pair of the terms t_1 and t_2, $y \in out(t_1) \cap out(t_2)$ yields the same value, $val_y(t_1) = val_y(t_2)$.

For definition of mass computations this model should be extended by inclusion of indexed operations and indexed variables (arrays). This technical work can be easily done. Obviously, in this extended model, a mass algorithm is represented by an infinite recursively countable set of functional terms.

A program that implements an algorithm, represented by a set of functional terms, can be constructed with the procedure calls to mod_a in the order not

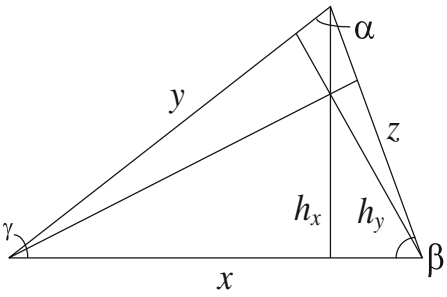
contradicting to the information dependences between the operations imposed by the terms structure. Usually, a run-time system is used to implement all the calls in a proper order.

4.3 An Example of Knowledge Base, i.e. Partially Defined AT

Below (Fig. 3) the example of the description of the application partially defined axiomatic theory TRIANGLE is given. The partially defined axiomatic theory TRIANGLE is the part of an axiomatic theory GEOMETRY. The axiomatic theory TRIANGLE does not describe the whole theory GEOMETRY, but only that its part that provides the solution of the problem, formulated in Fig. 2. Actually, the TRIANGLE is assembled out of the objects of GEOMETRY. The problem formulation defines:

- the sets of input $V = \{x, z, \gamma, \alpha\}$ and output $W = \{s\}$ variables of the problem,
- the necessary partial AT in which only this problem can be solved is assembled out of notions, valid for problem formulation and found in the active knowledge base. An algorithm of the formulated problem solution is derived on the partial AT and represented here as the set of two functional terms t_1 and t_2.

If some another problem is needed to be solved then new suitable modules and variables should be included into the AT for this problem solution.



$$V = \{x, z, \gamma, \alpha\};\ W = \{s\};$$
$$t_1 = f_1(x, d_1(z, a(\gamma, \alpha)));$$
$$t_2 = f_2(z, d_2(x, a(\gamma, \alpha)));$$

Fig. 2. Problem formulation

The capability to create similar knowledge base, to describe and to apply it constitutes the nature of the new literacy for active knowledge description.

Next time it is necessary to draw the attention, that computational model cannot contain unknown solution of a problem. It contains only well-known units of knowledge (ready-made modules). This means, that with the use of the method of structural program synthesis any user will be able to solve a problem with the quality equal to the quality of the best known solution.

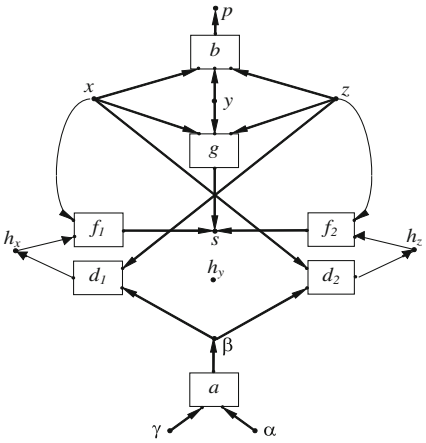


Fig. 3. Partial axiomatic theory TRIANGLE

4.4 Particle-In-Cell Method

Particle-In-Cell method (PIC) is widely used for numerical modeling in physics and chemistry [10,16]. But PIC is applied individually to the solution of an individual problem of numerical modeling. Therefore, now the PIC application cannot be formally described as a more or less complete AT. Fortunately, the methods of PIC parallel implementation are now well known and can be formally described within a partially described AT. This permits to generate a framework for solution of a certain problem of numerical modeling in which a user develops the sequential fragments of the codes whereas the whole parallel program is automatically assembled/generated out of them, for more details see [10,15].

Parallel implementation of PIC includes parallel implementation such sophisticated system algorithms as dynamic resources allocation, processes migration, dynamic load balancing, dynamic tuning of an application program to all the available resources, etc. Clearly, that solution of these problems should be formally described within a partially described AT.

4.5 LuNA Project

The above described approach was implemented within the LuNA project of fragmented programming system [10,11,13,15]. LuNA project is directed to the elimination of parallel programming from the process of large scale numerical models development. In fact LuNA is the system for implementation of AT, described as computational model.

The LuNA fragmented programming system makes:

– accept as input the knowledge description of an object domain (computational model here) and the lists of input and output variables,

– a desired numerical algorithm(s) of a problem solution is derived and represented as recursively countable set of functional terms [6,17],
– the desirable program implementing the derived algorithm is automatically generated. Both algorithm and program possess by all the desired/specified pragmatic properties (to be executed in parallel, dynamic load balancing, dynamic tuning to all the available resources, etc.).

Massivity and regularity of numerical algorithms allow generating parallel programs of acceptable quality. One of the serious problem of LuNA implementation is the replacement of the well-known in sequential programming system algorithms by distributed system algorithms with local interactions (DSALI), especially DSALI for distributed resources allocation, optimization of the generated program execution, dynamic load balancing and many others problems of dynamic optimization. LuNA is under permanent modification.

Now LuNA is able already to generate practical parallel programs implementing different application of PIC methods for large-scale simulation.

4.6 New Literacy

Now the new incipient literacy, based on the capability of a human being to create, to understand and to apply AT is arising. Technically, new writing language (AT writing) may contain the facility for sets of modules and variables description, like this is done in LuNA. New literacy (AT-literacy) includes also the ability to create a proper AT.

Earlier, the human population was divided into the following groups:

– the literate people which were able to read and to write texts,
– the semiliterate people which were able to read but not to write texts,
– the illiterate people which could neither read nor write the texts.

The current human population also begins to become stratified, dividing into the following groups:

– the illiterate people which mostly look at the screens and press the buttons, may be, they know phonetic alphabet and are able to read the text. They are usually not able to write reasonable texts.
– the semiliterate people who are able to read and to write phonetic texts. They know some systems of passive knowledge representation and are able to read and to adopt the knowledge in the passive representation,
– the literate people, who are able to operate with ATs and, certainly, will be able to create, to process, to understand and to utilize the knowledge in the active representation, in AT writing.

The current phonetic writing provides the accumulation of the knowledge in a passive form. As a rule, with the use of this literacy a human being is able to adopt and to apply the knowledge in one object domain, only.

The AT-literacy will practically infinitely extend the abilities of a human being to adaptation of new object domains. This literacy will substantially

change the representations of new scientific and industrial technologies. AT writing should provide at least a description of recursively countable sets of modules and variables.

Instead of an object domain adaptation in all its details, only the ability to formulate a problem will be needed. In this manner, for instance, 10 object domains can be adopted.

The ability to formulate the problems will be the basis of education in the future. Only a few people, good mathematicians, also experienced in parallel computing technologies, will be doing by the most interesting work, i.e., the development of the active knowledge bases.

5 Conclusion

Models of parallel computation are now successfully utilized in scientific and industrial modeling. Above consideration demonstrates that these models will be applied for modeling social phenomena too. AT writing will co-exists in parallel with phonetic writing.

It seems, that next 50 or may be 100 years and even more IT community will be doing by the creation of active knowledge bases: transformation of currently accumulated passive knowledge bases into active form and creation new one. This is not single-step process. The sphere of programming systems application will be narrowing and instead the systems, implementing AT (like LuNA), should be developed.

Finally, I would like to remark the following. The human society on its way to more progressive organization meets many problems. Current American movies often frighten us by extraterrestrials, monsters, etc. But new literacy is really far more terrible thing, bad dream, that threaten to the humanity by degradation (human population moronization) in next one or two centuries, because with the use of active knowledge technology human beings do not need to think.

It is clear, that the uncontrolled development of any scientific discipline, including such inoffensive thing as literacy, can lead to humanity self-destruction. Therefore, united nations and states must provide a global control on applications of **any** new scientific result.

Hope, I am not fully right and this prediction will not be proved.

References

1. Zadykhailo, I.B.: Sostavlenie tsiklov po parametricheskim zapisyam spetsialnogo vida. Zhurnal vychislitelnoi mathematiki i mathematicheskoi physiki **3**(2), 337–357 (1963). (in Russian)
2. Manna, Z., Waldinger, R.: Synthesis: dreams -> programs. IEEE Trans. SE **SE–5**, 294–398 (1979)
3. Ershov, Y.L., Goncharov, S.S., Sviridenko, D.I.: Semantic programming. Information Processing: Proceedings of IFIP 10th World Computer Congress Series, vol. 10, Amsterdam, pp. 1113–1120 (1986)

4. Giannesini, F., Kanoui, H., Pasero, R., Caneghem, M.: Prolog. International Computer Science Series. Addison-Wesley, Wokingham (1986)
5. Genesereth, M.R., Nilsson, N.J.: Logical Foundation of Artificial Intelligence. Morgan Kaufmann, Los Altos (1987)
6. Valkovskii, V.A., Malyshkin, V.E.: Synthesis of Parallel Programs and Systems on the Basis of Computational Models, Nauka, Novosibirsk, p. 128 (1988). (in Russian, Sintez parallelnykh programm i sistem na vychislitelnykh modelyakh)
7. Andrianov, A.N., Efimkin, K.N., Zadykhailo, I.B.: Nonprocedural language for mathematical physics. Program. Comput. Softw. **17**(2), 10–22 (1992)
8. Andrianov, A.N., Efimkin, K.N., Levashov, V.Y., Shishkova, I.N.: The NORMA language application to solution of strong nonequilibrium transfer processes problem with condensation of mixtures on the multiprocessor system. In: Dongarra, J., Alexandrov, V.N., Tan, C.J.K., Juliano, B.A., Renner, R.S. (eds.) ICCS-ComputSci 2001. LNCS, vol. 2073, pp. 502–510. Springer, Heidelberg (2001)
9. Malyshkin, V.: Assembling of parallel programs for large scale numerical modeling. In: Handbook of Research on Scalable Computing Technologies, Chap. 13, pp. 295–311. IGI Global, USA (2010)
10. Kraeva, M.A., Malyshkin, V.E.: Assembly technology for parallel realization of numerical models on MIMD-multicomputers. Future Gener. Comput. Syst. Elsevier Sci. **17**(6), 755–765 (2001)
11. Kireev, S., Malyshkin, V.: Fragmentation of numerical algorithms for parallel subroutines library. J. Supercomputing **57**(2), 161–171 (2011)
12. Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Haidar, A., Herault, T., Kurzak, J., Langou, J., Lemarinier, P., Ltaeif, H., Luszczek, P., YarKhan, A., Dongarra, J.: Flexible development of dense linear algebra algorithms on massively parallel architectures with DPLASMA. In: Proceedings of the Workshops of the 25th IEEE International Symposium on Parallel and Distributed Processing, Anchorage, Alaska, USA, pp. 1432–1441. IEEE (2011)
13. Malyshkin, V.: How to create the magic wand. Currently implementable formulation of the problem. In: The Proceedings of the 5-th SoMeT International Conference on New Trends in Software Methodologies, Tools and Techniques, vol. 147, pp. 127–132. IOS Press, Quebec, 28–30 September 2006
14. Malyshkin, V.: Two approaches to program synthesis or implementation of partially defined theories. In: Proceedings of the 11th International Conference on New Trends in Software Methodologies, Tools and Techniques, vol. 246, pp. 285–291. IOS Press, USA (2012)
15. Malyshkin, V.E., Perepelkin, V.A.: The PIC implementation in LuNA system of fragmented programming. J. Supercomputing **69**(1), 89–97 (2014). Springer
16. Kireev, S.: A parallel 3D code for simulation of self-gravitating gas-dust systems. In: Malyshkin, V. (ed.) PaCT 2009. LNCS, vol. 5698, pp. 406–413. Springer, Heidelberg (2009)
17. Kleene, S.C.: Introduction to Metamathematics. Princeton University Press, Princeton (1952)