ДОПОВІДІ
НАЦІОНАЛЬНОЇ
АКАДЕМІЇ НАУК
УКРАЇНИ

ІНФОРМАТИКА ТА КІБЕРНЕТИКА

https://doi.org/10.15407/dopovidi2020.06.015
UDC 004.4'24

A.Yu. Doroshenko, O.A. Yatsenko

Institute of Software Systems of the NAS of Ukraine, Kyiv
E-mail: doroshenkoanatoliy2@gmail.com, oayat@ukr.net

Formal methods of parallel software design automation

Presented by Academician of the NAS of Ukraine P.I. Andon

Formal methods and software tools of automated design and synthesis of parallel programs are proposed. The developed facilities use the language based on the Glushkov system of algorithmic algebras intended for a high-level and natural linguistic representation of algorithms and apply rewriting rules technique to transform programs. The tools also use the method of syntactically correct algorithm scheme design which eliminates syntax errors during the construction of algorithms and programs. The approach is illustrated on developing the parallel N-body simulation program for the executing on a graphics processing unit.

Keywords: *algebra of algorithms, automated design, formal methods, graphics processing unit, parallel computation, term rewriting.*

Current and emerging scientific and industrial applications require a significant computing power provided by parallel platforms such as multicore, clusters, cloud computing, and GPGPU (General-Purpose Computing on Graphics Processing Units). Modern software development tools for such platforms are quite complex and require the use of new programming models, as well as knowledge of software and hardware details. One of the promising areas in the modern parallel programming is the development of abstract models and formal methods of design, analysis, and implementation of algorithms and programs. Such models and methods are constructed, particularly, in the framework of the algebraic programming and algorithmics. Algebraic programming is based on the term rewriting theory [1] and describes the processes of program design, algebraic transformations, proving mathematical theorems and the development of intelligent agents. The algorithmics is the direction of computer science being developed within the Ukrainian algebraic-cybernetic school [2, 3]. It is based on the Glushkov system of algorithmic algebras (SAA) focused on solving the problems of formalization and design of sequential and parallel algorithms. The objects of research in algorithmics are models of algorithms and programs represented in the form of high-level specifications — schemes. This paper proposes formal methods and software tools for designing parallel programs based on the algebra of algorithms. The ap-

Цитування: Doroshenko A.Yu., Yatsenko O.A. Formal methods of parallel software design automation. *Допов. Нац. акад. наук Укр.* 2020. № 6. С. 15—20. https://doi.org/10.15407/dopovidi2020.06.15

plication of the tools is illustrated with an example of the development of the parallel *N*-body simulation program intended for the executing on a graphics processing unit (GPU). The proposed approach is related to works on formal methods [4] and synthesis of programs from specifications [5, 6]. The main distinctive feature of our approach consists in using algebraic specifications based on SAA and represented in a natural linguistic form simplifying the understanding of algorithms and facilitating the achievement of demanded software quality. Another advantage of our tools is the method of automated design of syntactically correct algorithm specifications [2], which eliminates syntax errors during the construction of schemes.

Formal algorithm design in the Glushkov algebra. Like algebraic specifications in general, SAA is focused on the analytical form of representation of algorithms and a formal transformation of these representations, in particular, with the purpose of optimization of algorithms by given criteria. SAA is the two-sorted algebra $GA = < \{\mathrm{Pr}, Op\}; \Omega_{GA} >$, where sorts Pr and Op are sets of logic conditions and operators defined on some information set IS; Ω_{GA} is the signature of operations. Logic conditions are predicates defined on the set IS and taking values of the three-valued logic $E_3 = \{0, 1, \mu\}$, where 0 is for false, 1 is for true, and μ is for unknown; the operators represent mappings (probably, partial) of IS to itself. The signature Ω_{GA} consists of logic operations (disjunction, conjunction, negation) and operator constructs, which can be represented in analytic, natural linguistic, and flowgraph forms. In this paper, we use the natural linguistic form based on the algorithmic language SAA/1 [2] intended for the multilevel structured designing and documenting of sequential and parallel algorithms and programs in the form of SAA schemes. The main operator constructs of SAA/1 include the following:

- serial execution of operators: "*operator* 1"; "*operator* 2";
- branching: IF '*condition*' THEN "*operator* 1" ELSE "*operator* 2" END IF;
- for loop: FOR (*counter* FROM *start* TO *fin*) "*operator*" END OF LOOP;
- asynchronous execution of n operators: PARALLEL($i = 0, \ldots, n-1$) ("*operator* i");
- synchronizer, which delays the computation until the value of the specified condition is true: WAIT '*condition*'.

In [2], additional constructs to design parallel programs for GPUs using Nvidia CUDA [7] were added. In particular, they included the operation calling the kernel function executed by parallel threads:

CALL GPU KERNEL(*blocksPerGrid*, *threadsPerBlock*) ("*operator*"),

where *blocksPerGrid* is the number of blocks in the CUDA grid; *threadsPerBlock* is the number of threads in each block; "*operator*" is executed by a separate thread.

Software tools for automated design and synthesis of algorithms and programs. The algebra-algorithmic approach is supported by tools developed within the framework of Kyiv algebraic-cybernetic school. Particularly, they include the Integrated toolkit for Design and Synthesis of programs (IDS) [2, 3] and the term rewriting system TermWare [1]. The main components of IDS are the constructor intended for the dialogue design of syntactically correct schemes of algorithms represented in SAA and the synthesis of programs in C, C++, Java languages and the database containing the description of SAA constructs, basic predicates and operators, and their program implementations. The basic idea of the constructor consists in the level-by-level top-down designing of schemes by detailing language constructs of SAA. On

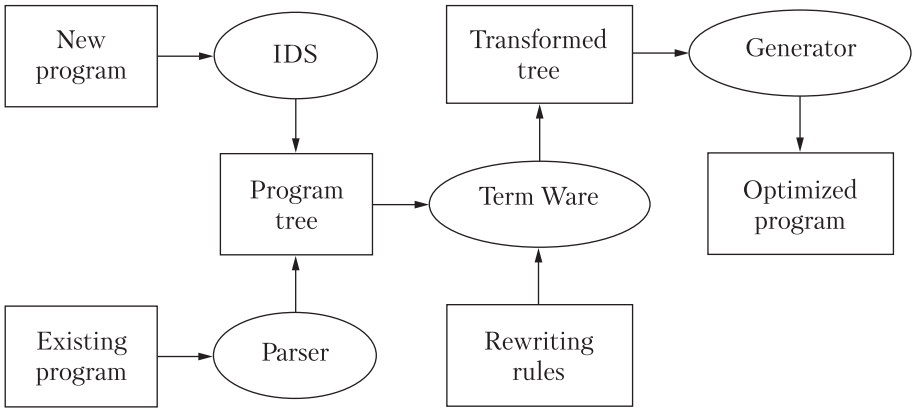


Fig. 1. The workflow of software development with IDS and TermWare

each step of the design, the system allows the user to select only those constructs, the substitution of which into a scheme does not break its syntactic correctness. The constructor uses the list of SAA constructs from the database and an algorithm design tree. During the design of an algorithm, the SAA operations chosen by the user are displayed in the tree with the further detailing of their variables. Depending on a type of the chosen variable (logic or operator), the system offers the corresponding list of SAA operations. Based on the algorithm tree, IDS generates the text of an SAA scheme and a program in a target language.

To automate the transformation (e.g., parallelization) of programs, IDS is applied together with the term rewriting system TermWare [1] (see Fig. 1).

TermWare provides a language for describing the rewriting rules that operate on data structures which are called terms, and a rule engine that interprets rules to transform terms. Informally, terms are tree-like structures of the form $f(t_1, t_2, \ldots, t_n)$, where subterms $t_1, t_2, \ldots, t_n$ are either tree nodes themselves or leaf nodes corresponding to constants or variables. The general form of a TermWare rule is $source\,[condition] \to destination\,[action]$ where $source$ is the input sample, $destination$ is a target sample, $condition$ is a term defining the applicability of the rule, $action$ is the operation executed when the rule is applied. The conditions and the actions are optional components of the rule, which can call imperative code. For this purpose, the developer should implement the "fact base" – a class providing the methods that can be called from rewriting rules. In this way, the connection between declarative rules in TermWare language and imperative code in traditional object-oriented languages (such as C, C#, Java) is established.

IDS and TermWare were applied for the automated design and generation of parallel programs for multicore CPUs and GPUs, particularly, in the subject domain of weather forecasting [2, 3, 8].

Application of the algebra-algorithmic tools for developing an example of a parallel program. We applied IDS and TermWare for designing a parallel program intended for solving the *N*-body simulation problem on a GPU. The problem consists in a simulation of a dynamical system of *N* particles with known masses m_i that interact in pairs according to the Newtonian law of gravitation [9]. The positions and velocities of the particles at the initial time moment $t = 0$ are known and are $r_i\big|_{t=0} = r_0$ and $v_i\big|_{t=0} = v_0$, respectively. It is necessary to approximately find positions and velocities at next time moments. The system of $2N$ first-order ordinary dif-

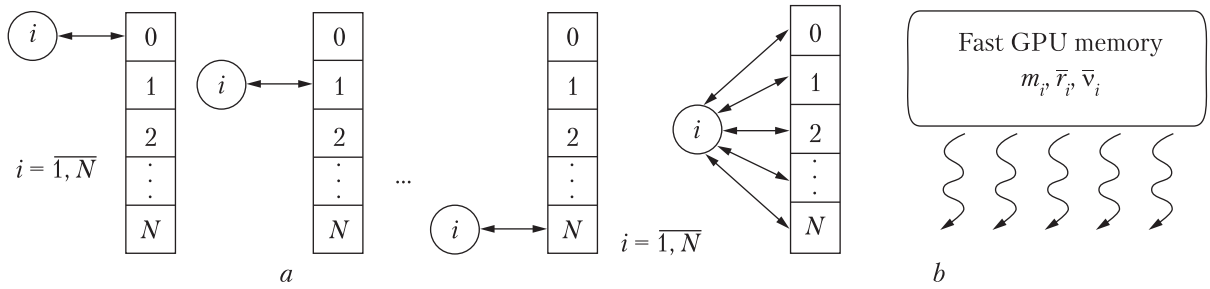


Fig. 2. The general computation scheme in sequential (a) and parallelized (b) *N*-body simulation algorithms

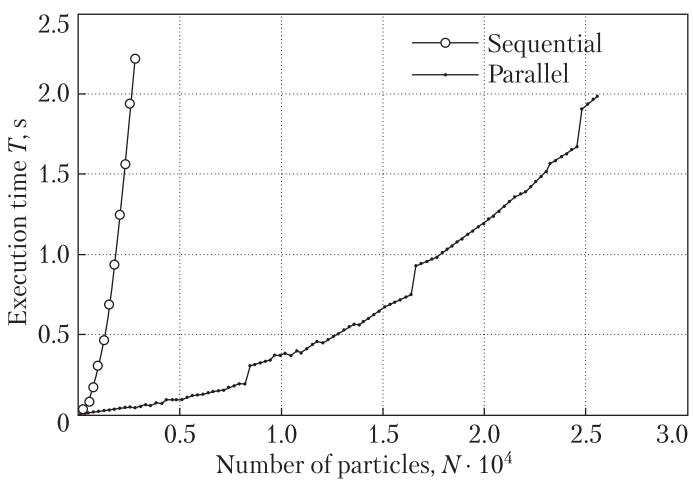
ferential equations is integrated numerically with discretization by t using Hermite interpolating polynomials [10]. The general scheme for the sequential *N*-body simulation algorithm is shown in Fig. 2, *a*. The basic idea of parallelizing the algorithm consists in executing operations associated with each particle $i = \overline{1, N}$ in a separate GPU thread (see Fig. 2, *b*); the operations with GPU memory are also added. The sequential algorithm was designed using IDS and then transformed to parallel with the help of TermWare.

As an example, consider the fragment of the sequential SAA scheme:

FOR (*i* from 0 to *N*)
"Calculate the acceleration of the particle (*i*) and its derivative"
END OF LOOP

The transformation of the loop to the parallel version consists in adding the GPU kernel call operation and the synchronizer:

FOR (*i* from 0 to *N*/512)
CALL GPU KERNEL (*blocksPerGrid*, *threadsPerBlock*) (
"Calculate the acceleration of the particle (*i*) and its derivative");
WAIT 'All threads completed work'
END OF LOOP,



where *N* is divided by 512 for a more stable GPU work; *threadsPerBlock* = = 256, *blocksPerGrid* = *N* / *threadsPerBlock*.

The sequential and the parallel programs were executed on a computer with i5-3570 CPU (4 cores) and GeForce GTX 650 Ti GPU (768 CUDA cores). Fig. 3 shows the dependence of the execution time of the programs on *N*.

Fig. 3. The dependence of the execution time of sequential and parallel *N*-body simulation programs on the number of particles *N*

The maximum multiprocessor speedup T_{CPU} / T_{GPU} (where T_{CPU} , T_{GPU} are the execution times of the sequential and parallel program, respectively) was 502.64 obtained at $N = 65536$. The efficiency of GPU cores usage was 66 %.

Conclusions. The formal methods and tools for automated design and synthesis of parallel programs are proposed. The approach uses the algorithmic language based on the Glushkov system of algorithmic algebras focused on the natural linguistic representation of algorithms. The advantage of the developed toolkit consists in using the method of syntactically correct algorithm schemes design, which eliminates syntax errors during the construction of algorithms. The approach is illustrated on developing the parallel *N*-body simulation program. The experiment consisting in executing the program on a graphics processing unit was conducted, which showed a good level of computation parallelization.

REFERENCES

1. Doroshenko, A., & Shevchenko, R. (2006). A rewriting framework for rule-based programming dynamic applications. Fundamenta Informaticae, 72(1-3), pp. 95-108.
2. Andon, P. I., Doroshenko, A. Yu., Zhereb, K. A., & Yatsenko, O. A. (2018). Algebra-algorithmic models and methods of parallel programming. Kyiv: Akademperiodyka. https://doi.org/10.15407/akademperiodyka.367.192
3. Doroshenko, A. Yu., & Yatsenko, O. A. (2020). Formal and adaptive methods for automation of parallel programs construction. (unpublished manuscript)
4. Butler, R. W. (2001). What is formal methods? Retrieved from http://shemesh.larc.nasa.gov/fm/fm-what.html
5. Flener, P. (2002). Achievements and prospects of program synthesis. In A. C. Kakas & F. Sadri (Eds.). Computational Logic: Logic Programming and Beyond. LNCS (Vol. 2407, pp. 310-346). Berlin: Springer. https://doi.org/10.1007/3-540-45628-7_13
6. Gulwani, S. (2010). Dimensions in program synthesis. Proc. of the 12th Int. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (pp. 13-24). New York: ACM. https://doi.org/10.1145/1836089.1836091
7. Nvidia CUDA technology (n.d.). Retrieved from http://www.nvidia.com/cuda
8. Prusov, V. & Doroshenko, A. (2018). Computational techniques for modeling atmospheric processes. Hershey, PA: IGI Global. https://doi.org/10.4018/978-1-5225-2636-0
9. Aarseth, S. J. (2003). Gravitational N-body simulations: Tools and algorithms. Cambridge: Cambridge University Press.
10. Makino, J., & Aarseth, S. J. (1992). On a Hermite integrator with Ahmad-Cohen. Publications of the Astronomical Society of Japan, 44, pp. 141-151.

Received 28.03.2020

А.Ю. Дорошенко, О.А. Яценко
Інститут програмних систем НАН України, Київ
E-mail: doroshenkoanatoliy2@gmail.com, oayat@ukr.net

ФОРМАЛЬНІ МЕТОДИ АВТОМАТИЗАЦІЇ
ПРОЕКТУВАННЯ ПАРАЛЕЛЬНИХ ПРОГРАМ

Запропоновані формальні методи та інструментальні засоби автоматизованого проектування та синтезу паралельних програм. Розроблені засоби використовують мову, яка ґрунтується на системах алгоритмічних алгебр Глушкова і орієнтована на високорівневе та природно-лінгвістичне подання алгоритмів, а також застосовують техніку переписувальних правил для трансформації програм. Особливістю розробленого інструментарію є також використання методу проектування синтаксично правильних схем алгоритмів, який виключає можливість виникнення помилок у процесі побудови специфікацій алгоритмів та

програм. Підхід проілюстровано на розробці паралельної програми чисельного інтегрування задачі N тіл, призначеної для виконання на графічному прискорювачі.

Ключові слова: *автоматизоване проектування, алгебра алгоритмів, графічний прискорювач, паралельні обчислення, переписування термів, формальні методи.*

А.Е. Дорошенко, Е.А. Яценко

Институт программных систем НАН Украины, Киев
E-mail: doroshenkoanatoliy2@gmail.com, oayat@ukr.net

ФОРМАЛЬНЫЕ МЕТОДЫ АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Предложены формальные методы и инструментальные средства автоматизированного проектирования и синтеза параллельных программ. Разработанные средства используют язык, базирующийся на системах алгоритмических алгебр Глушкова и ориентированный на высокоуровневое и естественно-лингвистическое представление алгоритмов, а также применяют переписывающие правила для трансформации программ. Особенность разработанного инструментария также состоит в использовании метода проектирования синтаксически правильных схем алгоритмов, который исключает возможность возникновения ошибок в процессе построения спецификаций алгоритмов и программ. Подход проиллюстрирован на разработке параллельной программы численного интегрирования задачи N тел, предназначенной для выполнения на графическом ускорителе.

Ключевые слова: *автоматизированное проектирование, алгебра алгоритмов, графический ускоритель, параллельные вычисления, переписывание термов, формальные методы.*