

DEFINITION OF THE PROGRAM NOTION

V. E. Malyshkin, V. A. Perepelkin

Institute of Computational Mathematics and Mathematical Geophysics SB RAS,
630090, Novosibirsk, Russia
Novosibirsk State University,
630090, Novosibirsk, Russia
Novosibirsk State Technical University,
630073, Novosibirsk, Russia

DOI: 10.24412/2073-0667-2024-2-16-31

EDN: CEDVVD

When solving complex problems in programming an important role plays definition of the program notion. Depending on the way program is concerned an approach to its construction, as well as its properties, vary. In the paper the program notion is studied and defined based on the theory of parallel programs synthesis on the basis of computational models. The proposed definition conforms to the theory, starting from the description of the problem in the subject domain terms and up to imperative program execution with dynamic properties provided.

In programmers' work it is not usual to employ a precise definition of the program notion. Usually some partial definition is used, which is applicable in particular circumstances. In practice there is no problem with that. Nevertheless, in solving problems of automatic construction of any kind of programs (sequential, parallel, distributed, real-time, numerical, etc.) in various computational model (in various models of informatics) and for different bases it becomes necessary to precisely define the program notion. This allows to precisely define objects and concepts of the theory and practice of informatics and use them in precise proofs/argumentation of various statements in different theories. That's why we have chosen mathematical logic as the base theory within which all considerations in the papers are made.

The theory of synthesis of parallel programs and systems on the basis of computational models was originally formulated in [1] and further studied in [2, 3]. Computational model (CM) is a bipartite directed finite graph, the parts of which form two sets of vertices, called sets of operations and variables. The arcs entering and exiting the operation determine the input and output variables of this operation, respectively. A computational model describes a certain subject domain, where the properties of objects in the subject domain are described by the set of variables (property values are represented by variable values), and the ability to calculate the values of some properties from others is represented by the set of operations (see Fig. 1). CM defines an axiomatic theory.

The interpretation function I is defined on the vertices of the graph. Each variable x has the value $I(x)$, and each operation a computes a function $I(a)$. To make computations available each operation has a computational module assigned. An example of such module is a conventional serial subroutine capable of computing values of output variables of the operation, provided values of operation's input variables are submitted as inputs to the subroutine. Let's define two subsets on the set of variables of the computational model — V and W , and call them input and output variables of the problem. Values

This work was carried out under state contract with ICMMG SB RAS FWNM-2022-0005.

of all variables from V are considered given. In this case, we will say that the VW -problem is posed on a computational model. If in a computational model there is a subset of operations, the ordered application of which to known values of variables will allow obtaining values of new variables until all variables from W obtain values, then this set defines a set of functional terms, each of which calculates one of the variables in W and is calculated from the variables of set V . We will call this set of functional terms a VW -plan (or simply a plan). To solve any VW -problem, generally, zero or more VW -plans can be constructed. Obviously, for a given VW -problem, one can set the task of finding a VW -plan, and if successful, compute the values of all variables from W , given the values of the variables from V .

The above allows us to give the following definition of the notion of a program. **A program is a description of a process of computation of the values of the interpretation function for the variables included in the VW -plan, and only them.**

Execution of a plan demands definition of control, i.e. the order in which operations are to be executed. That order must not contradict information dependencies between operations. Also the resources have to be assigned: each variable must have a memory extent to store its value and each operation must have a processor time to execute its corresponding module. Generally both control definition and resources assignment should be done partially statically and the rest — dynamically. This allows to provide static and dynamic properties of the program. Derivation of a VW -plan to solve the formulated problem can also be derived dynamically.

The proposed definition of the program notion has a number of advantages, considered in the paper. The advantages include the ability of algorithmic optimizations, the ability to optimize non-functional properties, etc. Also the process of programming in terms of computational models is described. Other program definitions are concerned in comparison.

Key words: Program concept, automatic program construction, active knowledge.

References

1. Sintez parallelnykh programm i sistem na vychislitelnykh modelyakh / Valkovsky V. A., Malyshev V. E.; Onv. red. Kotov V. E.; AN SSSR, Sib. otd-nie, VC. Novosibirsk: Nauka. Sib. otd-nie, 1988. 126 s (In Russian).
2. Malyshev V. E. Tekhnologiya fragmentirovannogo programmirovaniya // Vestnik YuUrGU. Seriya: Vychislitel'naya matematika i informatika. 2012. N 46 (305) (In Russian).
3. Malyshev V. Active Knowledge, LuNA and Literacy for Oncoming Centuries. In Essays Dedicated to Pierpaolo Degano on Programming Languages with Applications to Biology and Security. V. 9465. Springer-Verlag, Berlin, Heidelberg, 2015. P. 292–303.
4. Ershov A. P. Vychislimost v proizvolnykh oblyastyah i bazisakh: Sb. nauchn. st. M: VINITI, 1982, P. 3–58. Semiotika i informatika; VyP. N 19 (In Russian).
5. Yanov Yu. I. Metod svyortok dlya razresheniya svoystv formalnykh sistem. M.: IPM im. M. V. Keldysha, 1977. VyP. 11. 41 s. (Institut prikladnoy matematiki AN SSSR. Preprint; N 11 za 1977 g.). [Electron. Res.]: <https://library.keldysh.ru/preprint.asp?id=1977-11> (In Russian).
6. Valkovsky V. A. O sinteze optimalnykh programm na baze vychislitelnykh modeley // Programmirovaniye. 1980. N 6. S. 27–36. (In Russian)
7. Malyshev V., Perepelkin V., Schukin G. Scalable Distributed Data Allocation in LuNA Fragmented Programming System // Journal of Supercomputing, S.I.: Parallel Computing Technologies - 2017. Springer, 2017. P. 1–7. DOI: 10.1007/s11227-016-1781-0.
8. Kudryavtsev A. A., Malyshev V. E., Nushtayev Yu. Yu. Perepelkin V. A., Spirin V. A. Effektivnaya fragmentirovannaya realizatsiya kraevoy zadachi filtratsii dvukhfaznoy zhidkosti // Problemy informatiki. 2023. N 2. S. 45–73. DOI: 10.24412/2073-0667-2023-2-45-73 (In Russian)

9. Akhmed-Zaki, D., Lebedev, D., Perepelkin, V. Implementation of a three dimensional three-phase fluid flow (“oil-water-gas”) numerical model in LuNA fragmented programming system // *Journal of Supercomputing* (2017). N 73(2). Springer, 2017. P. 624–630. DOI: 10.1007/s11227-016-1780-1.
10. Perepelkin V. A., Sofronov I. V., Tkacheva A. A. Avtomatizatsiya konstruirovaniya chislennykh parallelnykh programm s zadannymi nefunktsionalnymi svoystvami na baze vychislitelnykh modeley // *Zhurnal Problemy informatiki*, 2017. N 4. S. 47–60. (In Russian)
11. Malyshkin, V. E., Perepelkin, V. A. (2011). LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // In: Malyshkin, V. (eds) *Parallel Computing Technologies. PaCT 2011. Lecture Notes in Computer Science*, V. 6873. Springer, Berlin, Heidelberg. [Electron. Res.]: https://doi.org/10.1007/978-3-642-23178-0_5.
12. Malyshkin V., Perepelkin V., Lyamin A. 2023. Trace Balancing Technique for Trace Playback in LuNA System // In *Parallel Computing Technologies: 17th International Conference, PaCT 2023, Astana, Kazakhstan, August 21–25, 2023, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 42–50. [Electron. Res.]: https://doi.org/10.1007/978-3-031-41673-6_4.
13. Perepelkin V., Malkhanov V., Zakirov V. Preliminary results on fault tolerance support in LuNA system // *Bull. Nov. Comp. Center, Comp. Science*, 46 (2022), P. 43–55.
14. Malyshkin, V., Akhmed-Zaki, D., Perepelkin, V. Parallel programs execution optimization using behavior control in LuNA system // *J Supercomput.* — Springer, 2021. S. 9771–9779. DOI: 10.1007/s11227-021-03654-2.
15. Malyshkin V. E., Perepelkin V. A. Multiagentniy podkhod k povysheniyu effektivnosti ispolneniya fragmentirovannykh programm v sisteme LuNA // *Problemy informatiki*, 2023, N 3, s. 55–67. DOI: 10.24412/2073-0667-2023-3-55–67 (In Russian).
16. Belyaev, N., Kireev, S. LuNA-ICLU Compiler for Automated Generation of Iterative Fragmented Programs // In: Malyshkin V. (eds) *Parallel Computing Technologies. PaCT 2019. Lecture Notes in Computer Science* (2019). V. 11657. Springer, Cham. [Electron. Res.]: https://doi.org/10.1007/978-3-030-25636-4_2.

ОПРЕДЕЛЕНИЕ ПОНЯТИЯ ПРОГРАММЫ

В. Э. Малышкин, В. А. Перепелкин

Институт вычислительной математики и математической геофизики СО РАН,
630090, Новосибирск, Россия

УДК 004.4

DOI: 10.24412/2073-0667-2024-2-16-31

EDN: CEDVVD

При решении сложных задач в программировании важную роль играет определение понятия программы. От того, как понимается программа, зависят подход к ее конструированию и ее свойства. В работе рассматривается понятие программы и дается ему определение на базе теории синтеза параллельных программ на вычислительных моделях. Предлагаемое определение отражает подход к процессу конструирования программы, определяемый этой теорией, начиная с описания задачи в терминах предметной области и заканчивая исполнением императивной программы с динамическими свойствами. Подход обладает рядом преимуществ, рассматриваемых в статье, таких как возможность выполнения алгоритмических оптимизаций, возможность автоматического конструирования программы, возможность обеспечения нефункциональных требований и проч. Рассматриваются параллели с другими определениями программ и особенности практического применения предлагаемого подхода.

Ключевые слова: понятие программы, автоматическое конструирование программ, активные знания.

Введение. В работе программистов нечасто случается необходимость использовать точное понятие программы. Обычно используется некоторое частичное определение, справедливое в конкретном случае, и на практике вопросов к таким определениям не возникает. Однако в решении проблем автоматического конструирования любых различных программ (последовательных, параллельных, распределенных, реального времени, программ численного моделирования и пр. и пр.), в различных моделях вычислений (в различных моделях информатики) и в различных базисах¹ необходимо использовать точное понятие программы для того чтобы можно было давать точные и понятные определения объектам и понятиям в теории и практике информатики и использовать их в точных доказательствах/обоснованиях различных утверждений в различных теориях. Поэтому мы выбрали в качестве базовой теории, в рамках которой будут проходить все рассуждения в статье, математическую логику. Коллеги, знающие/помнящие университетский курс математической логики, найдут нашу статью простой, понятной и, надеемся, полезной.

Далее в статье излагается понятийный аппарат, который был разработан в области автоматического конструирования параллельных программ на вычислительных моделях, и владение которым позволяет решать сложные задачи системного программирования за счет того, что все ключевые этапы создания программы рассмотрены в их взаимосвязи, что позволяет в конкретных практических задачах выявлять проблемы, которые

Исследование выполнено в рамках государственного задания ИВМиМГ СО РАН FWNM-2022-0005.

¹Термин «базис» употреблен в том же смысле, что и в статье [4]

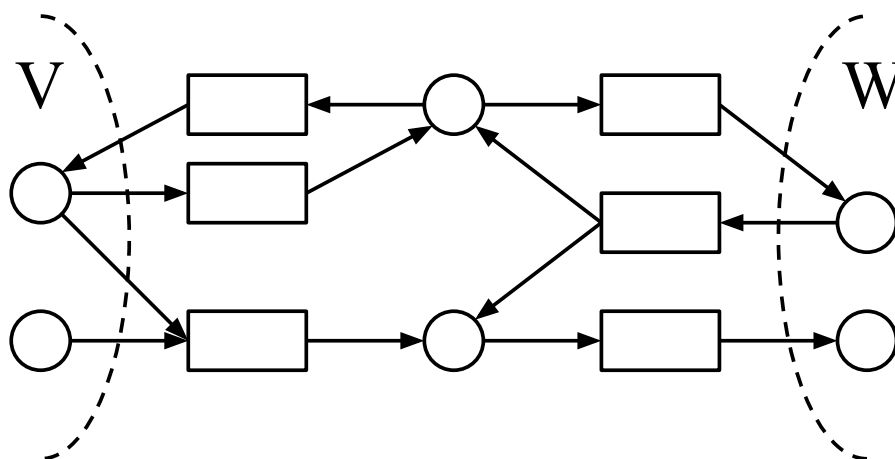


Рис. 1. Пример вычислительной модели. Круги — переменные, прямоугольники — операции

требуется решить, применять конкретные методы к их решению и получать результат с предсказуемым качеством.

Остальная часть статьи структурирована следующим образом. В разделе 1 вводится терминология и определяется понятие программы. В разделе 2 анализируются предложенные определения. В разделе 3 излагаются преимущества рассматриваемого подхода к определению понятия программы. В разделе 4 проводятся параллели с другими определениями понятия программы. В 5 разделе на примере рассматривается процесс конструирования программы. В заключении подводятся итоги работы.

1. Ключевые понятия. Излагаемые в разделе понятия вводятся в [1]. Формирование и развитие рассматриваемого подхода отражено в работах [2–3, 5–6].

Вычислительная модель (ВМ) — это двудольный ориентированный конечный граф, доли которого образуют два множества вершин, называемых множествами операций и переменных. Входящие в операцию и исходящие из нее дуги определяют соответственно входные и выходные переменные этой операции. Вычислительная модель описывает некоторую предметную область, где свойства объектов предметной области описаны множеством переменных (значения свойств представляются значениями переменных), а возможность вычислять значения одних свойств из других представлена множеством операций (см. рис. 1). ВМ определяет аксиоматическую теорию.

Например, в предметной области «тригонометрия» переменные могут описывать такие величины как длины сторон треугольника, величины его углов, радиус вписанной окружности и т. п., а операции — возможность вычисления одних величин через другие по известным тригонометрическим формулам, например, площади треугольника по двум сторонам и углу между ними, угол по двум другим и т. п.

Пусть I — функция, называемая *функцией интерпретации*, которая ставит в соответствие переменным элементы из некоторого множества значений D (базиса), а операциям — некоторые функции, арность которых соответствует количеству входных и выходных переменных операции. Значения, поставленные функцией I в соответствие переменным, будем называть значениями этих переменных. Также будем говорить, что операция a вычисляет функцию f , если $I(a) = f$.

Интерпретацию будем называть *корректной*, если для любой операции a из того, что определена ее интерпретация и интерпретация ее входных и выходных переменных, следу-

ет, что выполняется равенство: $\langle y_1, y_2, \dots, y_n \rangle = f(x_1, x_2, \dots, x_m)$, где y_1, y_2, \dots, y_n — это значения выходных переменных операции, x_1, x_2, \dots, x_m — это значения входных переменных операции, а f — функция, которую вычисляет операция. Далее в статье рассматриваются только корректные интерпретации.

Для обеспечения возможности вычисления значений выходных переменных операции из значений ее входных переменных назначим каждой операции a в соответствие вычислительный модуль (например, последовательную процедуру) mod_a , вычисляющий функцию $I(a)$. Тогда реализацией операции будем называть применение модуля к значениям входных переменных операций с выработкой модулем значений ее выходных переменных.

Определим на множестве переменных вычислительной модели два подмножества — V и W , и назовем их входными и выходными переменными задачи, зададим значения всех переменных из V . В таком случае будем говорить о том, что поставлена VW -задача на вычислительной модели. Если в вычислительной модели существует подмножество операций, упорядоченное применение которых к заданным значениям переменных позволит получать значения новых переменных до тех пор, пока все переменные из W не получат значения, то это множество задает множество функциональных термов, каждый из которых вычисляет одну из переменных множества W и вычисляется из переменных множества V . Это множество функциональных термов будем называть VW -планом (или просто планом). Для решения любой VW -задачи может быть сконструировано, вообще говоря, ноль или более VW -планов. Очевидно, что для заданной VW -задачи можно ставить задачу поиска VW -плана, и в случае успеха вычислить значения всех переменных из W , имея значения переменных из V .

Любой VW -план T содержит множество функциональных термов t_y вида $f_a(t_{x_1}, \dots, t_{x_n})$, где y — это переменная, вычисляемая некоторой операцией $a \in T$ из переменных x_1, \dots, x_n , f_a — это функциональный символ, обозначающий функцию $I(a)$, а t_{x_j} — это терм, вычисляющий переменную x_j , $j = 1 \dots n$, который либо является предметным символом, если $x_j \in V$, либо некоторым другим термом вида $f_b(t_{z_1}, \dots, t_{z_m})$, если переменная x_j вычисляется некоторой операцией $b \in T$ и имеет входные переменные z_1, \dots, z_m .

Каждый терм или подтерм VW -плана вычисляет некоторую переменную ВМ. Будем говорить, что такие переменные входят в VW -план.

В рассмотренном виде множество термов, представляющих алгоритм решения задачи, является конечным, т. к. ВМ — это конечный граф. Но в общем случае в [1] рассматриваются расширения понятия ВМ (рекурсивные ВМ, ВМ с массивами и пр.), где алгоритм решения задачи может быть представлен рекурсивно-перечислимое множеством функциональных термов. В настоящей статье эти случаи не рассматриваются, т. к. для изложения ключевых идей использование простых ВМ представляется достаточным.

Осуществление вычисления значений переменных, входящих в W , из значений переменных V на компьютере требует отображения элементов плана на ресурсы компьютера во времени — значения переменных должны храниться в памяти, а для выполнения модулей соответствующим операциям должен быть назначен процессорный элемент и отведено время. Сам же процесс вычислений можно рассматривать как процесс упорядоченного вычисления значений функции интерпретации для переменных, принадлежащих VW -плану.

Вышесказанное позволяет дать следующее определение понятию программы. **Программа** — это описание процесса вычисления значений функции интерпретации для переменных, входящих в VW -план, и только их.

2. Анализ понятия программы. В соответствии с введенными определениями процесс конструирования программы решения задачи в некоторой предметной области может рассматриваться следующим образом. Рассмотрение будет сопровождаться примером решения задачи в области школьной тригонометрии.

Сначала составляется описание предметной области в виде вычислительной модели. А именно, в вычислительной модели описываются представляющие интерес понятия предметной области в виде переменных, которые могут являться входными, промежуточными или выходными величинами в программе. Например, в тригонометрии это могут быть длины сторон треугольников, величины углов, радиусы вписанной и описанной окружностей, площадь и т. п. Также в вычислительную модель вносятся операции, которые могут быть полезны для получения решения задачи путем вычисления одних величин из других. Например, это могут быть операции, вычисляющие площадь треугольника по двум сторонам и углу между ними, радиус вписанной окружности по трем сторонам и т. п. Операции должны быть подкреплены возможностью их вычислить, т. е. для каждой операции должны иметься формула или алгоритм вычислений.

На следующем этапе на вычислительной модели ставится VW -задача, определяющая входные и выходные величины требуемой программы. Также формулируются критерии. По VW -задаче строится VW -план решения задачи. Если задать нефункциональные² свойства операций и переменных и функцию качества VW -плана, зависящую от этих свойств, то в отношении построения VW -плана может решаться оптимизационная задача. Применительно к тригонометрическому примеру на этом этапе будут определены множество операций и порядок их применения, приводящий к вычислению значений переменных W из V .

Далее требуется предоставить программные модули, реализующие операции VW -плана в соответствии с функцией интерпретации. Эти модули могут уже иметься в библиотеках прикладных подпрограмм данной предметной области, либо они должны быть разработаны.

Затем необходимо доопределить VW -план, по существу являющийся алгоритмом решения задачи, до программы, т. е. определить управление и распределение ресурсов. Управление определяет, в каком порядке будет осуществляться реализация операций (этот порядок не должен противоречить информационным зависимостям операций), а распределение ресурсов определяет, в какой области памяти вычислителя и в какие периоды времени будет храниться значение той или иной переменной, и на каком процессорном элементе будет реализовываться каждая операция. Другими словами, на этом этапе собственно разрабатывается программный код, который осуществляет упорядоченное вычисление значений функции интерпретации для переменных из VW -плана.

При этом управление и распределение ресурсов может быть статически определено в программе лишь частично, а окончательно будет определено в процессе исполнения, динамически (см, например, [7]). В этом случае в конструируемую программу должен быть добавлен программный код, принимающий и реализующий эти решения. Простыми примерами такого случая могут служить динамическое выделение памяти в куче (heap), когда конкретный адрес ячеек памяти, которые будут отведены под хранение значения перемен-

²Тут и далее под *функциональными* свойствами, характеристиками или требованиями понимается все, что относится к функции, которую вычисляет программа или модуль, т. е. то, как значения выходных аргументов зависят от входных; под *нефункциональными* — все остальные, такие как время выполнения, расход памяти, требования к программному окружению и т. п.

ной, определяется динамически, и портфель задач как шаблон проектирования программ, подразумевающий, что конкретный процессорный элемент для исполнения той или иной операции не определяется статически, а назначается динамически по мере освобождения процессорных элементов от выполнения очередной задачи из портфеля.

Примерами реализации задач на основе описанного подхода могут служить [8–9].

3. Преимущества предлагаемого определения. *Декомпозиция.* Преимуществом введенного определения является то, что оно подразумевает описанную в предыдущем разделе декомпозицию процесса конструирования программы на этапы, требующие решение задач разного типа. На начальном этапе работа выполняется в терминах предметной области, что позволяет однозначно задать постановку задачи (функциональную и нефункциональную) неспециалисту в программировании, в то время как на финальном этапе программа формируется в терминах, близких к аппаратному обеспечению. Задачи по выводу алгоритма и определения управления и распределения ресурсов могут быть выполнены специалистами в области системного программирования, не знакомыми с предметной областью, без риска внести ошибку в программу, связанную с незнанием предметной области.

Алгоритмические оптимизации. Вывод алгоритма решения задачи является одним из этапов конструирования программы. Наличие в вычислительной модели избыточных операций и переменных позволяет, вообще говоря, выводить множество VW -планов решения одной и той же задачи. Все эти планы будут функционально эквивалентны (т. е. вычисляют одну и ту же функцию), но их нефункциональные свойства могут отличаться. Это позволяет ставить и решать оптимизационную задачу по выводу VW -плана. В том числе, вывод алгоритма может осуществляться и автоматически, и динамически. Пример частной реализации этой идеи изложен в [10].

Автоматическое конструирование. На основе рассмотренного определения может быть обеспечено автоматическое конструирование программ в предметной области. Если человек описал вычислительную модель предметной области и предоставил вычислительные модули, реализующие операции, то далее возможно по постановке VW -задачи автоматически конструировать программу ее решения. Для этого требуется разработать и реализовать алгоритмы планирования (вывода VW -плана) и алгоритмы определения управления и распределения ресурсов.

В общей постановке проблема конструирования оптимальной (в смысле нефункциональных характеристик, существенных в соответствующей предметной области) программы является алгоритмически труднорешаемой, поэтому универсального ее решения не предполагается. Тем не менее, в конкретных предметных областях может быть обеспечено автоматическое конструирование программ удовлетворительного (по нефункциональным характеристикам) качества. Для этого должны быть разработаны (или использованы существующие при их наличии) частные системные алгоритмы. Наиболее подходящими для этого предметными областями являются те, в которых уже усилиями ручного программирования накоплены программные модули и сложилась практика их применения для решения задач в этой предметной области.

Примером реализации идеи автоматического конструирования класса программ численного моделирования может служить система LuNA [11].

Решение новых классов задач. Возможность автоматического конструирования программы решения поставленной задачи, включая автоматический вывод алгоритма, позволяет решать с приемлемыми затратами задачи, которые при ручном программирова-

нии решать затруднительно. Это касается задач, требующих существенной адаптивности в отношении свойств входных данных и вычислителя. Рассматриваемый подход позволяет автоматически и динамически реконструировать и конструировать программы, в том числе параллельные, в том числе с динамическим конструированием управления и распределения ресурсов (динамическая балансировка нагрузки на вычислительные узлы мультимониторного компьютера).

Простым иллюстративным примером может служить задача сортировки массива. Как известно, существуют различные алгоритмы сортировки, и при этом выбор оптимального алгоритма зависит от конкретных условий его работы. Например, на небольших массивах сортировка пузырьком обычно обгоняет быструю сортировку. На достаточно больших массивах быстрая сортировка обгоняет сортировку пузырьком, но проигрывает в скорости сортировке подсчетом элементов, которая, в свою очередь применима не всегда из-за своих ограничений. Есть и множество других алгоритмов, предпочтительных в своих частных случаях. Предлагаемый подход позволяет описать вычислительную модель, содержащую описание всех нужных алгоритмов сортировки с их нефункциональными свойствами, обеспечив возможность системе автоматического конструирования программ выбирать подходящий алгоритм по ситуации, в том числе и динамически, перестраиваясь под те данные, которые обрабатываются в настоящий момент, и под характеристики вычислителя, фактически им обеспечиваемые (т. е. с учетом производительности его компонентов, количества доступных ресурсов, нагрузки сторонними процессами и т. п.).

И хотя такую адаптивность возможно реализовать и вручную, например, запрограммировав динамический выбор подходящей процедуры сортировки, но принятие этого решения будет ограничено заложенными в программу алгоритмами. Предлагаемый же подход возлагает автоматическое принятие решений на систему, которая может со временем развиваться и учитывать все большее количество факторов (свойств решаемой задачи, входных данных и вычислителя), в том числе и на основе автоматического сбора статистики исполнения.

Профилирование. Известная техника профилирования позволяет инструментировать код и собирать статистику характеристик его исполнения для последующей оптимизации. Предлагаемый подход позволяет расширить возможности профилирования за счет того, что в сконструированной программе возможно сохранять информацию о соответствии частей кода конкретным операциям и переменным вычислительной модели и тем самым интерпретировать статистические данные в этих терминах. На основе анализа такого профиля возможно автоматически оптимизировать все этапы конструирования программы, начиная с вывода алгоритма (т. е. в т. ч. выполнять алгоритмические оптимизации автоматически).

Некоторые результаты по автоматической оптимизации эффективности конструируемых программ на основе профилирования в системе LuNA изложены в [12].

Накопление баз активных знаний. Предлагаемый подход позволяет аккумулировать накапливаемые в предметных областях прикладные алгоритмы и программные модули в форме баз активных знаний для обеспечения автоматического применения накопленных прикладных алгоритмов и программных модулей для решения новых задач. База активных знаний — это совокупность взаимосвязанных вычислительных моделей, программных модулей и специализированных для этой предметной области системных алгоритмов. База активных знаний содержит все необходимое для того, чтобы система автоматического конструирования программ могла строить нужные программы автоматически.

Например, если в некоторой прикладной программе для сортировки массива использовалась соответствующая база активных знаний, то при включении в эту базу активных знаний нового алгоритма сортировки это приведет к тому, что этот новый алгоритм будет использован в прикладной программе в подходящих условиях автоматически, без какой-либо необходимости прикладного программиста модифицировать прикладную программу или даже знать о существовании нового алгоритма сортировки.

Предельным развитием этой идеи можно считать ситуацию, когда для решения новой задачи на основе множества взаимосвязанных баз активных знаний система автоматически собирает ее решение из лучших модулей, накопленных в базах активных знаний без необходимости человека знать о том, какие вообще модули накоплены в этих базах активных знаний, какими свойствами они обладают и как технически осуществлять их запуск.

Дополнительные возможности отладки. По определению, если две операции имеют совпадающие множества входных и выходных переменных соответственно, то такие модули функционально эквивалентны. То есть, при подаче на вход модулям, реализующим эти операции, идентичные входные данные, результаты их работы также будут идентичны. И хотя функция, вычисляемая этими операциями, системе неизвестна, но знание о том, что для обеих операций эта функция совпадает, позволяет осуществлять автоматически некоторую отладку программных модулей. В частности, возможно осуществление дублирующих вычисление значений одних и тех же переменных различными модулями с последующим сравнением этих значений на эквивалентность. Какие значения считать эквивалентными — это вопрос, ответ на который зависит от конкретной предметной области. Например, если значением переменной является множество чисел, то в представлении этих значений в памяти может потребоваться их линейно упорядочить, причем порядок значения иметь не будет. При сравнении таких значений недостаточно, например, сравнить побайтово их представление в памяти. То же самое касается и вещественных чисел, представляемых в компьютере с ограниченной точностью, и в рамках допустимой погрешности (определяемой предметной областью) неравные представления вещественных чисел следует считать эквивалентными с точки зрения правильности работы модулей. Поэтому в базу активных знаний должен быть внесен критерий, алгоритм или модуль проверки значений на эквивалентность.

Надежность вычислений. Предлагаемый подход позволяет конструировать надежные программы. Например, если часть модулей содержит ошибки (что на практике скорее правило, чем исключение), то возможно значение каждой переменной вычислять несколькими операциями, реализуемыми различными модулями, и сравнивать полученное значение на эквивалентность. Если один из модулей дает отличное от остальных значение, то в этой ситуации следует прервать выполнение программы с соответствующим сообщением об ошибке, либо продолжить вычисления, приняв в качестве значения переменной то, которое выработали остальные операции.

Также может быть обеспечена и отказоустойчивость. Например, механизм сохранения контрольных точек может быть естественным образом реализован путем сохранения значений промежуточных переменных в процессе вычислений. При выходе из строя оборудования возможно будет продолжить вычисления, приняв все сохраненные значения в множество V . Попытка реализации этого механизма была предпринята в [13].

Функциональная спецификация. При автоматическом конструировании программ встает вопрос о том, как задавать функциональные требования к программе. А имен-

но, должно быть задано, какую функцию программа должна вычислять. В данном случае функция понимается в математическом смысле как отображение входных параметров программы на выходные. Известны различные способы определения функциональной спецификации программы, но в общем случае проблема описания функциональных свойств в виде, «удобном» и для пользователя системы конструирования, и для самой системы, является далеко не тривиальной. В рассматриваемом подходе функциональная спецификация каких-либо операций в явном виде не подразумевается. Вместо этого используется свойство функциональной эквивалентности операций, имеющих совпадающие входы и выходы.

Это дает возможность определять функциональную спецификацию требуемой программы через определение множеств V и W . Пользователь может сам доопределять вычислительную модель, добавляя в нее новые операции и переменные. При этом модули, реализующие операции, будут определять (неявно) функциональные связи переменных. Таким образом, пользователь может рассматривать функциональную спецификацию как композицию из модулей (фактически задаваемую через V и W), функциональная спецификация которых ему известна, либо ознакомившись с документацией относительно смысла переменных конкретной вычислительной модели. В любом случае, для системы конструирования сама функция будет неизвестна, а ее работа по выводу алгоритма будет заключаться в построении или перестроении VW -плана, т.к. все VW -планы функционально эквивалентны между собой для любых заданных V и W .

Так как каждой операции соответствует некоторая вычисляемая ею функция, то ее входные и выходные переменные оказываются друг относительно друга в соответствующем функциональном соотношении. Косвенно функционально оказываются связаны и переменные, не связанные операциями непосредственно. Вследствие этого выбор множеств V и W является способом задания функциональной спецификации задачи.

При этом для системы функция, соответствующая той или иной операции или VW -задаче, неизвестна и в явном виде отсутствует. Она определена неявно через программный код модулей, реализующих операции. Тем не менее, на практике этого достаточно для двух основных нужд: во-первых, человек, знающий функциональную спецификацию модулей, может задавать требуемую функциональную спецификацию конструируемой программы через функциональную композицию модулей (выбор V и W); во-вторых, система может заменять операции и подграфы на функционально эквивалентные им операции и подграфы автоматически. Последнее возможно за счет того, что модули и подграфы, имеющие идентичные множества входных и выходных переменных, функционально эквивалентны (это обеспечивается корректностью интерпретации).

Отметим, что VW -задача — это достаточно «грубый» способ задания функциональной спецификации, например, в сравнении с заданием через систему рекуррентных соотношений. Это выражается в том, что он позволяет задать не произвольную функцию в этой предметной области, а лишь такую, которая может быть сформулирована как композиция из функций, соответствующих операциям, причем только такая композиция, которая выражается некоторым подграфом вычислительной модели. На практике, тем не менее, этот способ вполне удобен, что показывает многочисленный опыт применения обычных библиотек прикладных подпрограмм в практических задачах, т.к. и там пользователь не может извлечь из библиотеки подпрограмму, решающую произвольную функцию, а должен выбирать из относительно небольшого набора функций, реализуемых готовыми библиотечными подпрограммами.

Возможность ручной настройки. Ввиду того, что задачи автоматического конструирования алгоритма, управления и распределения ресурсов — в общем случае алгоритмически труднорешаемые задачи, на практике полезен подход, при котором программист может вручную управлять процессом конструирования программы, отдавая предпочтения одним решениям и/или запрещая другие. В предлагаемом подходе это возможно за счет добавления в описание предметной области рекомендаций — описания желательного с точки зрения программиста распределения данных и вычислений по узлам мультикомпьютера, отображения переменных в память и проч. (см., например, [8, 14]).

4. Связь с другими определениями программы. Традиционно программой считается описание алгоритма, пригодное для исполнения компьютером. Описание вычислительной модели с заданными входными данными и множествами V и W на некотором формальном языке (например, языке LuNA [11]) также может быть интерпретировано автоматически, а значит тоже подпадает под традиционное определение программы. Предлагаемое же определение отличается тем, что в нем отражены ключевые составляющие процесса вычислений. Так или иначе, программа имеет дело с вычислением величин предметной области. Это происходит постоянно на протяжении всего вычислительного процесса.

Вычислительную модель можно сравнить с графом обработки данных, но есть существенные отличия. Во-первых, в вычислительной модели в целом нет определенной направленности вычислений — что из чего требуется вычислять (за исключением собственно операций, где входы и выходы определены явно). В зависимости от выбора множеств V и W «фронт вычислений» может проходить через вычислительную модель в разных направлениях. Во-вторых, в вычислительной модели возможна (и обычно должна быть) избыточность. Только часть операций (а именно — операции, входящие в VW -план) будет задействована для решения конкретной VW -задачи. Исключения составляют случаи, когда VW -план содержит дублирующие операции, например, в отладочных целях или для обеспечения надежности вычислений.

Интерпретатор и компилятор. Практическое применение описанного подхода в области высокопроизводительных вычислений требует учета эффективности — нефункциональных свойств модулей, таких как время выполнения, расход памяти, нагрузка на сеть и т. п.

С точки зрения решения VW -задачи на вычислительной модели это означает, что требуется выводить не произвольный VW -план, а наилучший (или, по крайней мере, удовлетворительный) по эффективности. Для этого необходимо формально описывать нефункциональные свойства модулей, ставить и решать оптимизационную задачу по выбору VW -плана.

Реализация на практике описанной выше базовой идеи возможна в двух основных формах и их различных сочетаниях — в виде интерпретатора и на основе генерации исполняемой программы (т. е. на базе компилятора). Первый вариант подразумевает наличие интерпретатора — некоторой виртуальной машины, которая принимает на вход описание вычислительной модели, постановку задачи, входные данные (значения переменных V). Она динамически конструирует VW -план и осуществляет на доступном оборудовании запуск модулей, реализуя операции до тех пор, пока все значения переменных из W не окажутся вычисленными.

Второй вариант подразумевает автоматическое конструирование программы, реализующей вычисление значений переменных из W , принимая на вход значения переменных из

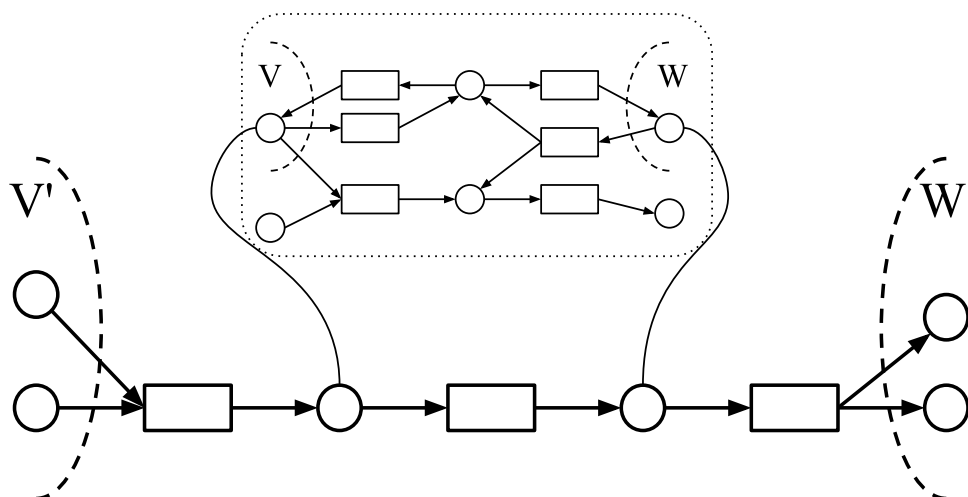


Рис. 2. Пример структурированной операции. Входные и выходные переменные структурированной операции отождествляются с переменными вложенной вычислительной модели, образуя множества V и W соответственно для VW -задачи на вложенной модели

V . Такую программу генерирует некоторый генератор, который принимает на вход описание вычислительной модели, VW -задачи, целевого вычислителя и, возможно, входных данных. При этом генератор строит VW -план статически, а конструируемая программа представляет собой управляющий код, целью которого является вызов модулей в порядке, предписываемом VW -планом. Далее сгенерированная программа компилируется в машинный код и выполняется с подачей ей на вход входных данных.

Интерпретатор предпочтителен в ситуациях, когда построение или реализация достаточно эффективного VW -плана возможны только в динамике. Примерами могут служить выбор подходящей операции при построении VW -плана в зависимости от свойств ее входных данных или необходимость динамически балансировать нагрузку по вычислительным узлам мультимьютера по мере освобождения ресурсов. Недостатком интерпретатора являются накладные расходы на динамическое принятие решений и реализацию операций и переменных как отдельных сущностей во время исполнения, поэтому при большом количестве «мелких» (по занимаемым ресурсам) операций и переменных, а также высоких требованиях к производительности, предпочтительным бывает генератор, конструирующий эффективную низкоуровневую программу с жестко заданным статически определенным управлением и распределением ресурсов. Комбинированный вариант (полуинтерпретация) подразумевает частичный переход к императивному представлению вычислений (т. е. в виде традиционной параллельной программы — Message Passing Interface, POSIX Threads и т. п.) при частичном сохранении фрагментированной структуры вычислений в динамике, когда динамические свойства реализуются только в части, необходимой для конкретной задачи (см. результаты работы в этом направлении в [15–16]).

5. Программирование в терминах вычислительных моделей. С точки зрения прикладного программиста при рассматриваемом подходе конструирование программы выглядит не как разработка императивной программы, в которой имеются обращения к внешним библиотечным подпрограммам, а как описание новой вычислительной модели, описывающей решение прикладной задачи. Для части операций, специфичных для данной задачи, программист самостоятельно вручную разрабатывает новые модули на обычном

процедурном языке программирования. Остальным операциям ставятся в соответствие не готовые программные модули, а VW -задачи на уже готовых вычислительных моделях (см. рис. 2). Таким образом, соответствующие операции будут реализованы автоматически с помощью каких-либо модулей, описанных в готовых вычислительных моделях без необходимости прикладному специалисту даже знать о том, какие модули в этих вычислительных моделях имеются и в каком количестве. Ему достаточно задать множества V и W для соответствующих подзадач (и, возможно, нефункциональные требования к их решению).

Отметим, что по VW -плану может быть построена как последовательная, так и параллельная программа. Во-первых, каждый модуль может сам по себе быть параллельной программой для общей или распределенной памяти, спецвычислителя и т. п. Во-вторых, если несколько операций информационно независимы, то их возможно выполнять параллельно, как на мультикомпьютере, так и на мультипроцессоре.

Заключение. Рассмотрено определение понятия программы, выработанное в рамках подхода к конструированию параллельных программ на основе вычислительных моделей. Описаны преимущества изложенного определения и рассмотрены различные аспекты его использования на практике при ручном и автоматическом конструировании программ. Предложенное определение может использоваться как обобщение для анализа других определений программы.

Список литературы

1. Вальковский В. А., Малышкин В. Э. Синтез параллельных программ и систем на вычислительных моделях / . Отв. ред. В. Е. Котов; АН СССР, Сиб. отд-ние, ВЦ. Новосибирск: Наука. Сиб. отд-ние, 1988.
2. Малышкин В. Э. Технология фрагментированного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2012. № 46 (305).
3. Malyshkin V. Active Knowledge, LuNA and Literacy for Oncoming Centuries // In Essays Dedicated to Pierpaolo Degano on Programming Languages with Applications to Biology and Security. V. 9465. Springer-Verlag, Berlin, Heidelberg, 2015. P. 292–303.
4. Ершов А. П. Вычислимость в произвольных областях и базисах: Сб. научн. ст. М: ВИНТИ, 1982. С. 3–58. (Семиотика и информатика; Вып. № 19).
5. Янов Ю. И. Метод сверток для разрешения свойств формальных систем. М.: ИПМ им. М. В. Келдыша, 1977. Вып. 11. 41 с. (Институт прикладной математики АН СССР. Препринт; № 11 за 1977 г.). [Электрон. Рес.]: <https://library.keldysh.ru/preprint.asp?id=1977-11>.
6. Вальковский В. А. О синтезе оптимальных программ на базе вычислительных моделей // Программирование. 1980. № 6. С. 27–36.
7. Malyshkin V., Perepelkin V., Schukin G. Scalable Distributed Data Allocation in LuNA Fragmented Programming System // Journal of Supercomputing, S.I.: Parallel Computing Technologies–2017. Springer, 2017. P. 1–7. DOI: 10.1007/s11227-016-1781-0.
8. Кудрявцев А. А., Малышкин В. Э., Нуштаев Ю. Ю., Перепелкин В. А., Спиринов В. А. Эффективная фрагментированная реализация краевой задачи фильтрации двухфазной жидкости // Проблемы информатики. 2023. № 2. С. 45–73. DOI: 10.24412/2073-0667-2023-2-45-73.
9. Akhmed-Zaki D., Lebedev D., Perepelkin V. Implementation of a three dimensional three-phase fluid flow (“oil-water-gas”) numerical model in LuNA fragmented programming system // Journal of Supercomputing (2017). N 73(2). Springer, 2017. P. 624–630. DOI: 10.1007/s11227-016-1780-1.

10. Перепелкин В. А., Софронов И. В., Ткачева А. А. Автоматизация конструирования численных параллельных программ с заданными нефункциональными свойствами на базе вычислительных моделей // Проблемы информатики. 2017. № 4. С. 47–60.
11. Malyshkin, V. E., Perepelkin, V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // In: Malyshkin, V. (eds) Parallel Computing Technologies. PaCT 2011. Lecture Notes in Computer Science, vol 6873. Springer, Berlin, Heidelberg. [Electron. Res.]: https://doi.org/10.1007/978-3-642-23178-0_5.
12. Malyshkin V., Perepelkin V., Lyamin A. 2023. Trace Balancing Technique for Trace Playback in LuNA System // In Parallel Computing Technologies: 17th International Conference, PaCT 2023, Astana, Kazakhstan, August 21–25, 2023, Proceedings. Springer-Verlag, Berlin, Heidelberg, 42–50. [Electron. Res.]: https://doi.org/10.1007/978-3-031-41673-6_4.
13. Perepelkin V., Malkhanov V., Zakirov V. Preliminary results on fault tolerance support in LuNA system // Bull. Nov. ComP. Center, ComP. Science, 46 (2022), P. 43–55.
14. Malyshkin, V., Akhmed-Zaki, D., Perepelkin, V. Parallel programs execution optimization using behavior control in LuNA system // J. Supercomput. Springer, 2021. P. 9771–9779. DOI: 10.1007/s11227-021-03654-2.
15. Малышкин В. Э., Перепелкин В. А. Мультиагентный подход к повышению эффективности исполнения фрагментированных программ в системе LuNA // Проблемы информатики. 2023, № 3, С. 55–67. DOI: 10.24412/2073-0667-2023-3-55-67.
16. Belyaev, N., Kireev, S. LuNA-ICLU Compiler for Automated Generation of Iterative Fragmented Programs // In: Malyshkin V. (eds) Parallel Computing Technologies. PaCT 2019. Lecture Notes in Computer Science (2019). V. 11657. Springer, Cham. [Electron. Res.]: https://doi.org/10.1007/978-3-030-25636-4_2.



Виктор Эммануилович Малышкин — получил степень магистра математики в Томском государственном университете (1970), степень доктора технических наук в Новосибирском государственном университете (1993). В настоя-

щее время является заведующим лабораторией синтеза параллельных программ в Институте вычислительной математики и математической геофизики СО РАН. Он также основал и в настоящее время возглавляет кафедру параллельных вычислений в Национальном исследовательском университете Новосибирска. Является одним из организаторов международной конференции PaCT (Parallel Computing Technologies), проводимых каждый нечетный год в России.

Имеет более 110 публикаций по параллельным и распределенным вычислениям, синтезу параллельных программ, суперкомпьютерному программному обеспечению и приложениям, параллельной реализации крупномасштабных численных моделей.

В настоящее время область его интересов включает параллельные вычислительные технологии, языки и системы параллельного программирования, методы и средства параллельной реализации крупномасштабных численных моделей, технологию активных знаний.

Victor Emmanuilovich Malyshev graduated from Tomsk State University with the master of sciences degree in 1970, defended his candidate dissertation in physical and mathematical sciences in Computing Centre of SB RAS in 1984, and defended his doctoral dissertation in technical sciences in Novosibirsk State University (1993). Currently is head of parallel programs synthesis laboratory in the Institute of computational mathematics and mathematical geophysics SB RAS.

Is also a founder and head of the chair of parallel computing in Novosibirsk State University. Is one of organizers of the PaCT (Parallel Computing Technologies) international conference, being held each odd year in Russia. Has more than 110 publications on parallel and distributed computing, parallel programs synthesis, supercomputing software and

applications, parallel implementation of large-scale numerical models. Current scientific interests include parallel computing technologies, languages and systems of parallel computing, methods and tools of software implementation of large-scale numerical models, the active knowledge technology.



Перепелкин Владислав Александрович — кандидат технических наук, старший научный сотрудник Института вычислительной математики и математической геофизики СО РАН; старший преподаватель кафедры параллельных вычислений факультета информационных технологий Новосибирского государственного университета. Тел.: (383) 330-89-94, e-mail: perpelkin@ssd.sccc.ru. В 2008 году окончил Новосибирский государственный университет с присуждением степени магистра техники и технологии по направлению «Информатика и вычислительная техника». В 2023 году защитил кандидатскую диссертацию. Имеет более 20 опубликованных статей по теме автоматизации конструирования параллельных программ в области численного моделирования. Является одним из основных разработчиков экспери-

ментальной системы автоматизации конструирования численных параллельных программ для мультикомпьютеров LuNA (от Language for Numerical Algorithms). Область профессиональных интересов включает автоматизацию конструирования параллельных программ, языки и системы параллельного программирования, высокопроизводительные вычисления.

Perepelkin Vladislav Aleksandrovich — PhD in Computer Science. Graduated from Novosibirsk State University in 2008 with the master degree in engineering and technology in computer science. Defended his PhD thesis in 2023. Nowadays has the senior researcher position at the Institute of Computational Mathematics and Mathematical Geophysics (Siberian Branch of Russian Academy of Sciences), and also has a part time senior professor position in the Novosibirsk State University. He is an author of more than 20 papers on automation of numerical parallel programs construction. He is one of main developers of system LuNA (Language for Numerical Algorithms) for automatic construction of numerical parallel programs. Professional interests include automation of parallel programs construction, languages and systems of parallel programming, high performance computing.

Дата поступления — 22.05.2024