

DEBUGGING SYSTEM OF SIMPLE COMPUTATIONAL MODELS IN GRAPH REPRESENTATION

A. Vlasenko, M. Gorodnichev, M. Kurbatov*

Institute of Computational Mathematics and Mathematical Geophysics SB RAS,
630090, Novosibirsk, Russia

Novosibirsk State Technical University,
630073, Novosibirsk, Russia

*Novosibirsk State University,
630090, Novosibirsk, Russia

DOI: 10.24412/2073-0667-2026-1-24-39

EDN: NPOQMK

Computational models (CM) is a formalism for representing knowledge about computing in problem domains. A CM defines a set of domain quantities (variables) and a set of operations that functionally connect the quantities. The choice of subsets V (known variables) and W (desired variables) determines the formulation of the problem on the CM, and the partial order of operations that allows one to “compute” W by V is called a (V, W) -plan. With appropriate computer implementation of variables and operations, automated problem solving based on CMs becomes possible. Building a CM — as a theory of the problem domain — is a complex task that requires significant skills and experience.

It is convenient to represent a CM as a bipartite graph with vertices of two types: variables and operations. Some operations produce variable values as results that will be input for other operations.

There are several software systems for designing and using CMs (LuNA, HPC Community Cloud, CoCoViLa, Orlando Tools, etc.) These systems have different interfaces and the user experience varies significantly depending on the system. For example, in the HPC Community Cloud, a user can place elements corresponding to values and operations on a canvas in the web interface and link them with arrows. In the LuNA system, the user writes a program in the language of the same name, similar to functional languages.

When using such systems, the user can make a number of mistakes, both when developing a CM and when setting problems. As a result, the solutions to the problem may either not exist at all, or they may differ significantly depending on the choice of the (V, W) -plan.

The following are several types of situations that should be taken into account.

1) The CM-graph has several connectivity components. If the graph is large enough (several dozen or more vertices with many arcs), then this situation may be difficult to detect visually. The existence of several connectivity components may indicate that the modeled subject area should be divided into several independent areas and investigated separately.

2) The inability to calculate any variables from others. In many fields of physics, mathematics, and other sciences, all values are interconnected by formulas, and it is always possible to calculate any value through others. In this case, the presence of a variable that is not the result of any operation is an error in CM-development. Although there may be subject areas in which some variables are always only input and are not calculated through others.

This work was carried out under state contract with ICMMG SB RAS FWNM-2025-0005.

3) A number of possible (V, W) -plans for solving the problem. This situation may be normal, since often the same problem can be solved using different methods, but it may also indicate unnecessary connections that were mistakenly entered into the CM.

4) The computational model design system chooses an inefficient (V, W) -plan according to some criterion. For example, it was not taken into account that the computing system on which the task will be run has graphics accelerators; an inefficient method for solving a problem was chosen (for example, the Kramer method was chosen instead of an iterative method for solving a system of linear equations), etc.

5) When designing the CM, the limitations of the problem domain were not taken into account (for example, that a certain value cannot be negative, that the values are dependent on each other, etc.)

Therefore, it is necessary to develop a CM debugging system that performs a number of checks on the CM in an automated mode and is capable of analyzing the (V, W) -problems. This paper is devoted to the description of such a system.

The system under development receives a CM in the form of a JSON file with a description in a specific format (this format is described in the paper). At the moment, the user is downloading it from the file system. In the future, imports will be implemented directly from CM design systems. To do this, it is necessary to develop a parser for the internal CM representation of each of these systems. After import, the CM graph is displayed in the system's web interface. The user has the opportunity to set a (V, W) -problem by selecting the vertices of the sets V and W . In this case, all existing (V, W) -plans for this problem will be displayed on the screen. If there are no solutions to the problem in this CM, then this will be notified to the user. The system also detects the presence of several connectivity components of the CM graph.

Key words: computational model, high performance computing systems, bipartite graph, automated debugging, visualization.

References

1. Valkovskii V. A., Malyshkin V. E. Sintez parallelnih program i sistem na vichislitelnykh modeliah. Novosibirsk: Nauka, 1988. 129 s.
2. Malyshkin V. E., Korneev V. D. Parallelnoe programmirovaniye multicomputerov: uchebnik. Novosibirsk: Izd-vo NGTU, 2006. 296 s.
3. Tiugu E. H. Konceptualnoye programmirovaniye. M.: Nauka, 1984. 256 s.
4. Gorodnichev M., Lebedev, D. Semantic tools for development of high-level interactive applications for supercomputers // J. Supercomput. 2021. P. 11866–11880. DOI: <https://doi.org/10.1007/s11227-021-03731-6>.
5. Malyshkin V., Akhmed-Zaki D., Perepelkin V. Parallel programs execution optimization using behavior control in LuNA system // J. Supercomput. 2021. P. 9771–9779. DOI: [10.1007/s11227-021-03654-2](https://doi.org/10.1007/s11227-021-03654-2).
6. Kotkas V., Ojamaa A., Grigorenko P., Maigre R., Harf M., Tyugu E. CoCoViLa as a multifunctional simulation platform // SIMUTOOLS 2011. 4th International ICST Conference on Simulation Tools and Techniques: March 21–25, - Barcelona, Spain: Brussels: ICST, 2011. P. 1–8.
7. Feoktistov A. G., Kostromin R. O., Voskoboynikov M. L., Li-De D. I. Organizatsiya vichislitelnoy sredi razrabotki i primeneniya nauchnykh rabochnykh processov na osnove konteynerizatsii // Vichislitelnye tehnologii. 2023. T. 28. N 6. S. 151–164.
8. Kurbatov M. A. Poisk oshibok vo fragmentirovannykh programmakh s pomosh'yu abstractnogo sintaksicheskogo dereva // Materialy 62-I Mezhdunarodnoy studencheskoy konferentsii. Novosibirsk, 2024. S. 158–159.
9. Malyshkin V., Vlasenko A., Michurov M. Automated Debugging of Fragmented Programs in LuNA System // D. Balandin et al. (Eds.): 22nd International Conference, MMST 2022, Nizhny

Novgorod, Russia, November 14–17, 2022. CCIS 1750, P. 266–280. DOI: 10.1007/978-3-031-24145-1_22.

10. Vlasenko A., Michurov M., Tsarev V., Kurbatov M. Postroenie kompleksa avtomatizirovannoy otladki fragmentirovannih programm // Vestnik NGU. Seriya: Informacionnie tehnologii. 2024. T. 22, N 1. S. 5–20. DOI: 10.25205/1818-7900-2024-22-1-5-20.

11. Gorodnichev M.A., Nalepova E.D., Merkulova E.A., Rudych P.D., Savostyanov A.N. Automation of EEG Data Processing with HPC Community Cloud // IEEE 24th International Conference of Young Professionals in Electron Devices and Materials (EDM), Novosibirsk, Russian Federation, 2023. P. 1320–1323. DOI: 10.1109/EDM58354.2023.10225226.

12. Fielding R.T. REST: Architectural Styles and the Design of Network-Based Software Architectures. Doctoral Dissertation, University of California, 2000.

13. Aho A., Hopcroft D., Ulman D. Structuri danih i algoritmi. M.: Vilyams, 2003. S. 189–197.

14. Knut D. Iskusstvo programmirovaniya. T. 1. 3-e izd. M.: Vilyams, 2006. S. 174–175.

СИСТЕМА ОТЛАДКИ ПРОСТЫХ ВЫЧИСЛИТЕЛЬНЫХ МОДЕЛЕЙ В ГРАФОВОМ ПРЕДСТАВЛЕНИИ

А. Ю. Власенко, М. А. Городничев, М. А. Курбатов*

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

Новосибирский государственный технический университет,
630073, Новосибирск, Россия

*Новосибирский национальный исследовательский государственный университет,
630090, Новосибирск, Россия

УДК 004.4

DOI: 10.24412/2073-0667-2026-1-24-39

EDN: PROQMK

Вычислительные модели (ВМ) — формализм для представления знаний о вычислениях в предметных областях. ВМ определяет множество переменных, соответствующих величинам предметной области, и множество операций, выражающих функциональные связи между величинами. Выбор подмножеств V — известных величин — и W — искомых величин — определяет постановку задачи на ВМ, а подмножество операций и переменных, связанных отношением частичного порядка, задающее схему вычисления W по V , называется (V, W) -планом. При соответствующей компьютерной реализации величин и операций обеспечивается возможность автоматизированного решения задач на основе ВМ. Построение ВМ — сложная задача, требующая значительных навыков и опыта.

Возможно допустить ряд ошибок, как при разработке ВМ, так и при постановке задач, вследствие чего решений у задачи может либо не быть вообще, либо они могут существенно отличаться в зависимости от выбора (V, W) -плана. Следовательно, необходима разработка системы отладки ВМ, проводящей ряд проверок над ВМ в автоматизированном режиме и способной выполнять анализ поставленных (V, W) -задач. Настоящая статья посвящена описанию такой системы.

Ключевые слова: вычислительная модель, высокопроизводительные вычислительные системы, двудольный граф, автоматизированная отладка, визуализация.

Введение. В рамках этой работы будут рассматриваться простые вычислительные модели (ПВМ). Согласно определению, данному в [1], ПВМ — это пара $M = (X, F)$, где: $X = \{x, y, z, \dots\}$ — конечное множество переменных для представления вычисляемых и измеряемых величин; $F = \{a, b, c, \dots\}$ — конечное множество функциональных символов операций арности $m \times n$, $m \geq 0$; m и n могут быть различны для разных операционных символов. Эти символы служат для выражения функциональных зависимостей между величинами предметной области; с каждым символом операции a арности $m \times n$ связан набор $in(a) = (x_1, \dots, x_m)$ входных и набор $out(a) = (y_1, \dots, y_n)$ выходных переменных, при этом $i \neq j \implies y_i \neq y_j, x_i \neq x_j$.

Исследования выполнены в рамках государственного задания ИВМиМГ СО РАН FWNM-2025-0005.

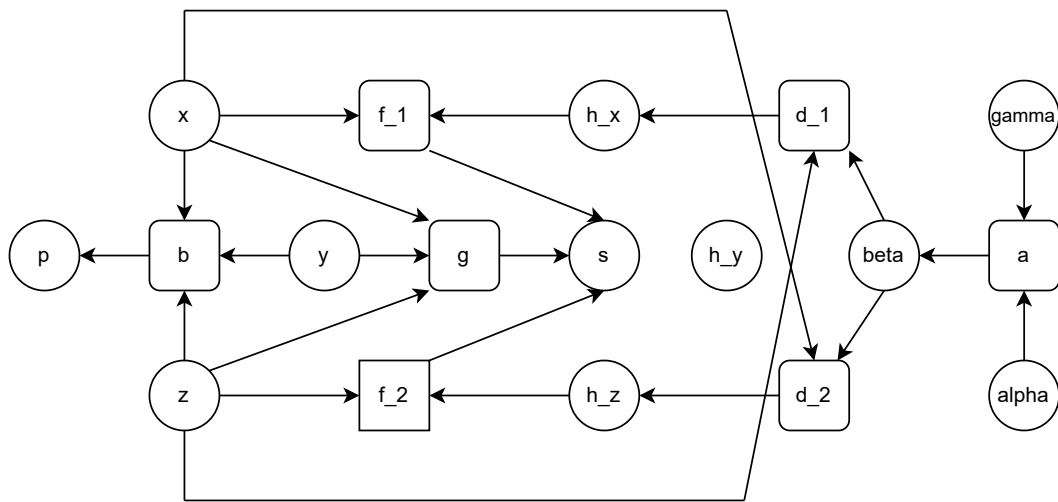


Рис. 1. ПВМ «треугольник»

ПВМ удобно представлять в виде двудольного графа переменных и операций. Пример такого графа для предметной области «треугольник» изображен на рис. 1.

На рис. 1 кругами обозначены символы переменных, а квадратами — символы операций. x, y, z — стороны треугольника; p — периметр; s — площадь; α, β, γ — углы; h_x, h_y, h_z — высоты, опущенные на соответствующие стороны. Операции d_1 и d_2 вычисляют h_x и h_z соответственно. f_1, f_2 и g вычисляют s . a и b вычисляют β и p соответственно. Данная модель, естественно, охватывает не всю предметную область и приведена для демонстрации. Так, в модели, например, отсутствуют медианы и биссектрисы треугольника.

Необходимо также ввести понятие термина, которое тесно связано с вычислительными моделями. Терм — математическая абстракция, определяемая по индукции [2]:

- 1) Термами длины 1 называются односимвольные слова из множества переменных.
- 2) Далее, пусть для некоторого $k > 1$ термы длины меньше k уже определены. Тогда слово t длины k называется термом, если оно имеет вид $f(t_1, t_2, \dots, t_n)$, где $t_1, t_2, \dots, t_n$ — термы длины меньшей чем k , f — n -арный функциональный символ (n входных переменных).

Применительно к приведенной выше ПВМ термами являются, например, $g(x, y, z)$; $f_1(d_1(\beta))$; $f_2(d_2(a(\alpha, \gamma)))$. Можно заметить, что терм определяет некоторый алгоритм — задает частичный порядок применения операций.

Для практического использования ПВМ требуется реализовать представление ее в компьютере [3]. Операции и величины должны иметь программную реализацию. Например, операции могут быть реализованы программными модулями, величины представлены в переменных или файлах. Должна быть обеспечена возможность организовать вызов программной реализации операции, подав на вход операции значения входных величин, чтобы вычислить значения выходных.

На ПВМ можно ставить задачи (так называемые (V, W) -задачи) в следующей форме: дано подмножество вершин V , значения каждой из которых известны. Необходимо найти значения всех вершин другого подмножества W . Используя операции из F , входными

параметрами для которых являются переменные из V , можно вычислить другие переменные, после чего применить следующую группу операций и еще расширить множество вычисленных переменных. Этот процесс можно продолжать до тех пор, пока среди вычисленных переменных не окажутся все элементы подмножества W . Однако может так оказаться, что далеко не все операции действительно было необходимо применять и не все полученные результаты нужны для решения задачи. В то же время вычислительные ресурсы на получение данных результатов были затрачены.

В связи с этим при решении подобных задач требуется составление так называемого (V, W) -плана вычислений, представляющего собой частично-упорядоченное подмножество операций и вычисляемых переменных. Применение операций в заданном порядке обеспечивает вычисление значения всех переменных из W при заданных значениях V . При этом в (V, W) -план не должно включаться выполнение «лишних» операций, не требующихся для нахождения значений элементов множества W . Данный план по сути своей задает алгоритм решения задачи в терминах операций, и таких планов для одной и той же задачи на ПВМ может быть множество. Следует отметить, что (V, W) -план удобно задавать термом, определяющим порядок операций, где термами длины 1 являются входные переменные. Однако это справедливо только для тех ВМ, где каждая операция имеет единственную выходную величину, либо множество выходных величин представимо в виде вектора, который поступает на вход следующей операции целиком.

Существуют системы описания вычислительных моделей, которые могут выглядеть совершенно различным образом. Например, HPC Community Cloud (HPC2C) [4] — система управления вычислениями на распределенных высокопроизводительных вычислительных ресурсах — предоставляет возможность в web-интерфейсе размещать на холсте символы операций и переменных, соединяя их связями, а система LuNA [5] содержит текстовый язык описания вычислительных моделей, схожий с функциональными языками. Система CoCoViLa [6] предлагает богатые визуальные и текстовые средства для описания сложных схем и детального описания структурированных компонентов. Программный комплекс Orlando Tools использует язык BPEL для представления ВМ с целью создания распределенных научных приложений в гетерогенной вычислительной среде [7].

Существенной проблемой при описании сложных предметных областей с помощью ПВМ и последующем их использовании для решения задач является возможность допустить множество семантических ошибок или получить неэффективный (относительно интересующих пользователя нефункциональных требований) (V, W) -план. Примерами могут служить включение в модель некорректных модулей в качестве реализации операций (вычисляющих не ту функцию, которая предполагалась разработчиком модели), установка некорректных связей (ошибка в составлении наборов *in* и *out* для операций), постановка неразрешимой на данной ПВМ (V, W) -задачи, построение неоптимального по некоторому критерию (V, W) -плана и др. Если ПВМ насчитывает десятки или сотни элементов (переменных и операций), то для обнаружения таких ошибок «вручную» может потребоваться очень длительное время. Соответственно, актуальной задачей является разработка алгоритмов обнаружения ошибок и программной системы, реализующей эти алгоритмы.

В связи с этим была поставлена задача разработки системы отладки с возможностью импорта ПВМ и (V, W) -задач из различных систем и последующего их анализа с целью выявления ошибок.

Стоит отметить, что некоторые системы конструирования ВМ обладают собственными средствами отладки. Так, например, для системы LuNA велись разработки инструментов

статического анализа [8], анализа по трассе [9], а в настоящее время поддерживается программный комплекс ADAPT [10], комбинирующий несколько подходов. Однако подобные инструменты способны определять ошибки, свойственные конкретной системе или технологии и не предназначены для обнаружения ошибок, характерных вычислительным моделям в общем.

1. Система графового представления простых вычислительных моделей. В рамках разрабатываемой системы необходимо было решить вопрос хранения ПБМ в некотором формате. ПБМ представима в виде двудольного графа. Таким образом, для его компьютерного представления можно использовать такие структурные языки, как JSON или XML. За основу взят формат ПБМ в HPC2C [11]. На базе данного формата был создан упрощенный унифицированный формат, который имеет ту же структуру и упрощенный набор полей: идентификатор (id), название (title) и тип (node_type) — переменная (variable) или операция (operation) для списка узлов (nodes) и идентификаторы начала (source) и конца (target) для дуг графа (edges). Таким образом, система отладки хранит граф как список вершин и список ребер.

Листинг 1. Пример графа в унифицированном формате.

```
{
  "nodes": [
    {
      "id": 0,
      "node_type": "VARIABLE",
      "title": "VARIABLE 2",
    },
    {
      "id": 1,
      "node_type": "OPERATION",
      "title": "OPERATION 0",
    },
    {
      "id": 2,
      "node_type": "VARIABLE",
      "title": "VARIABLE 2",
    }
  ],
  "edges": [
    { "source": 0, "target": 1 },
    { "source": 1, "target": 2 }
  ]
}
```

Унифицированный формат удобен для хранения ПБМ и построения визуализаций. Также подходит для решения задачи импорта ПБМ из различных систем (подробнее об этом далее). Но для задачи поиска необходимой информации в графе в целях анализа модели данный формат оказывается неподходящим. Например, для поиска всех входных переменных, нужных для работы определенной операции, в JSON-файле необходимо пе-

Таблица 1

Сложности операций для графовых структур данных

Структура данных	Поиск вершин, которые зависят от текущей	Перечисление всех дуг	Поиск вершин, от которых зависит текущая	Проверка существования дуги между u и v	Затраты по памяти на хранение структуры
Перечисление множеств	$\Theta(E)$	$\Theta(E)$	$\Theta(E)$	$\Theta(E)$	$\Theta(E + V)$
Матрица смежности	$\Theta(V)$	$\Theta(V ^2)$	$\Theta(V)$	$\Theta(1)$	$\Theta(V ^2)$
Матрица инцидентности	$\Theta(E)$	$\Theta(V \times E)$	$\Theta(E)$	$\Theta(E)$	$\Theta(V \times E)$
Перечень дуг	$\Theta(E)$	$\Theta(E)$	$\Theta(E)$	$\Theta(E)$	$\Theta(E)$
Вектор/ массив/ список смежности	$\Theta(\deg(v))$	$\Theta(E)$	$\Theta(E)$	$\Theta(\min(\deg(u), \deg(v)))$, если список вершин не отсортирован	$\Theta(V + E)$
Структура с оглавлением	$\Theta(\deg(v))$	$\Theta(E)$	$\Theta(E)$	$\Theta(\min(\deg(u), \deg(v)))$, если список вершин не отсортирован	$\Theta(V + E)$
Список вершин и список дуг	$\Theta(E)$	$\Theta(E)$	$\Theta(E)$	$\Theta(E)$	$\Theta(V + E)$

ребирать все дуги (edges), после чего по идентификаторам искать нужную информацию в узлах (nodes).

В целях обеспечения возможности экономичного поиска и анализа ПВМ был проведен обзор структур данных, служащих для представления графов (таблица 1). В таблице приведены сложности операций, требующихся для анализа ПВМ.

В таблице 1 употреблены следующие обозначения:

- 1) Θ — оператор, означающий, что количество операций пропорционально аргументу;
- 2) $|V|$ — количество вершин;
- 3) $|E|$ — количество дуг;
- 4) $\deg(u)$ — полная степень вершины u .

При помощи «Массива смежности» (рис. 2) можно достаточно быстро находить пути между вершинами, и она не занимает много места в памяти, в связи с чем было принято решение использовать данную структуру в целях дальнейшей отладки ПВМ. В терминах ПВМ эта структура выглядит следующим образом. Все вершины операций объединены в массив, и каждая имеет ссылку на список вершин переменных — множество входящих в эту операцию дуг. Дополнительно, каждая операция имеет ссылку на вершину переменной (или ссылки на несколько вершин), которая является результатом данной операции.

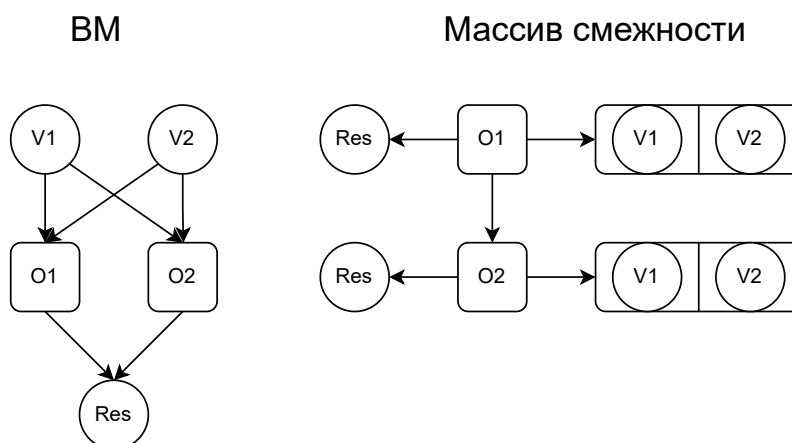


Рис. 2. Массив смежности

Разрабатываемая система отладки не хранит массив смежности в файловой системе. Он создается после импорта ПВМ перед проведением проверок, информация о которых изложена далее.

2. Обнаружение ошибок и выполнение проверок. Пользователь сам конструирует вычислительные модели, исходя из знаний о предметной области. При создании ВМ и (V, W) -планов могут возникать разного рода ошибки, которые могут приводить к некорректным вычислениям (результаты не соответствуют ожидаемой функциональной зависимости в моделируемой предметной области) или оптимизационным проблемам. Далее рассмотрены ситуации, потенциально свидетельствующие о наличии таких ошибок.

Во-первых, стоит обратить внимание на количество компонент связности (КС) в графе ВМ. Если их несколько (более одной), то нужно поставить вопрос об адекватности составленной модели предметной области и о разбиении одной модели на несколько независимых. Это дает преимущество в производительности последующего анализа ВМ за счет разбиения большого графа на несколько более мелких. Однако данный недочет не приводит к ошибкам вычислений, а скорее является плохой практикой.

Далее важно проанализировать возможные варианты вычислений переменных множества W по переменным из V . Если при конструировании неправильно проставить связи между вершинами или забыть добавить нужные операции, связать лишние переменные с операциями и т. д., то это вполне может привести к отсутствию возможности вычислить какую-либо переменную из W . На исходной же ПВМ (безотносительно поставленных (V, W) -задач) можно проверять тот факт, что любую переменную можно вычислить хотя бы одним способом. Противное может и не являться ошибкой, если, например, в ПВМ существуют переменные, которые всегда должны быть только входными. Однако во многих областях математики и физики все величины, которые имеет смысл заложить в ПВМ при моделировании, связаны между собой формулами, используя которые, можно выразить любую переменную через другие.

Часто встречаются ситуации, когда одну и ту же переменную из W можно вычислить по переменным из V несколькими разными способами. Например, систему линейных уравнений можно решать методом Гаусса, Крамера или итерационным методом, и все эти

модули решения СЛАУ могут присутствовать в ПВМ одновременно. Соответственно, для одной и той же (V, W) -задачи в таком случае можно построить несколько (V, W) -планов. Эта множественность вычислений допускается при конструировании ПВМ и не является ошибкой. Однако если при реализации операций допущены ошибки, то результат решения задачи будет зависеть от выбранного плана. Это может объясняться, например, ошибками округления и с прикладной точки зрения быть приемлемым, а может быть следствием принципиальных ошибок. Такие ситуации также необходимо выявлять, а для определения прикладной эквивалентности результатов должна быть соответствующая процедура. Очевидно, что такая процедура не может быть реализована только на основе графа ВМ, поскольку требуется непосредственный запуск операций, составляющих (V, W) -планы.

При составлении (V, W) -плана встает вопрос об оптимальности каждого возможного варианта вычислений по каким-либо критериям. Важным может быть учет следующих параметров:

- 1) Время работы наиболее часто вызываемых модулей на конкретном вычислителе.
- 2) Потребление оперативной памяти модулями.
- 3) Максимальная масштабируемость модулей по процессорным ядрам и вычислительным узлам.
- 4) Возможность использовать спец.ускорители (GPU, NPU и др.)
- 5) И др.

Разные ВМ могут быть связаны с разными предметными областями. Например, с биологией, физикой, математикой и т. д. Важно понимать, что каждая предметная область имеет свои ограничения. Например, сумма углов треугольника в евклидовой геометрии всегда равна 180 градусов, при нормальном атмосферном давлении вода кипит при 100 градусах Цельсия и т. д. Данные ограничения можно закладывать в ВМ при ее конструировании. Для этого необходимо выйти за рамки абстракции ПВМ и предоставить специалисту в предметной области возможность формализации подобных ограничений в прикладном инструменте работы с ВМ.

По ходу вычисления переменных можно собирать статистику использования элементов ВМ и результатов работы операций. Если периодически какая-то операция выдает некорректный результат, то выдавать составителю ВМ сообщение об этом и при составлении (V, W) -планов использовать альтернативные маршруты. Если по статистике использования выясняется, что какие-то переменные или операции из вычислительной модели не используются, то стоит поднять вопрос корректности и/или целесообразности хранения данных элементов в ВМ.

3. Архитектура и реализация системы отладки ПВМ.

3.1. Архитектура системы. Для возможности выполнения обозначенных проверок над ПВМ и (V, W) -планами в автоматизированном режиме разрабатывается система отладки, представляющая собой веб-приложение, в которое загружается ПВМ в описанном выше JSON-формате. Система позволяет отрисовывать граф ПВМ и проводить ряд проверок, список которых приведен ниже. Она состоит из трех слоев (рис. 3):

1) Слой “Frontend” (интерфейс приложения) предоставляет пользователю возможность отрисовывать граф ВМ, искать по нему нужную информацию и запускать проверки.

2) Слой “Backend” реализует всю логику работы с графами, термами и компонентами связности. На данный момент этот слой представлен монолитным компонентом. В дальнейшем возможен переход к микросервисам: поиск термов и анализ связности могут быть вынесены в отдельные сервисы по мере развития системы.

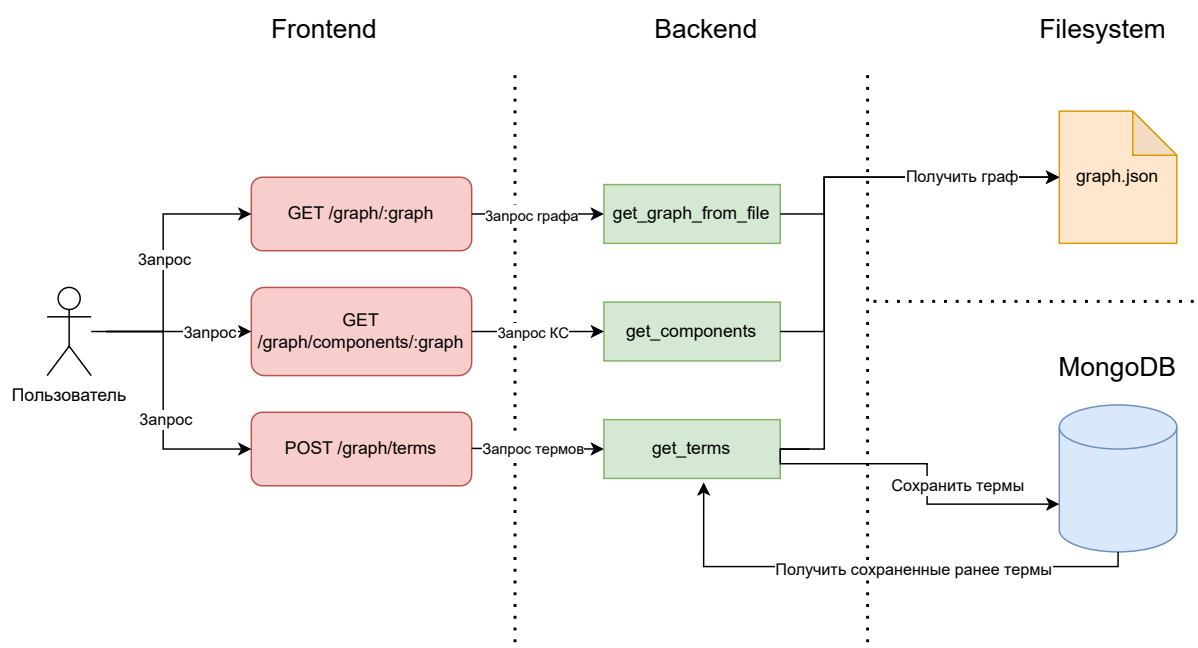


Рис. 3. Архитектура системы

3) Файловая система и БД (база данных). Данный слой хранит всю информацию о графах и результатах их анализа. Исходные графы хранятся в файловой системе (на рис. 3 — “Filesystem”), а найденные термы в базе MongoDB.

3.2. *Технологический стек.* В качестве основного языка программирования используется Rust, веб-приложение строится на основе фреймворк Axum. Для разработки фронтенда используется реактивный фреймворк Vue.js в версии с TypeScript и сборщиком Vite. Управление состоянием реализовано через глобальное хранилище Pinia, а для визуализации графов применяется библиотека Vis.js (vis-network и vis-data).

3.3. *Доступные функции.* На данный момент реализованы следующие функции:

1) «Импортировать граф» — загрузка графа PBM из JSON-файла с персонального компьютера пользователя и визуализация графа. В этом файле граф должен быть представлен в унифицированном формате, описанном в разделе 1. Система выполняет укладку графа на плоскость при помощи библиотеки Vis.js, автоматически задавая при этом координаты вершин. Вершины переменных отображаются сплошной границей, а операции пунктирной (рис. 1). В дальнейшем планируется добавить импорт графов из ряда других систем конструирования ВМ.

2) «Получить компоненты связности» — запрос данных о КС графа по его названию. Для удобства восприятия каждая компонента отображается своим цветом.

3) «Выбор V и W» — каждую из вершин переменных можно добавить во множество входных (V) или выходных (W) переменных, создав таким образом (V, W)-задачу. Входные вершины помечаются синим цветом, а выходные — красным.

4) «Поиск термов» — если заданы множества V и W, то можно определить все возможные пути вычисления выходных переменных из входных (см. 4.2. «Алгоритм поиска термов»). Используемые переменные и пути вычисления отображаются зеленым цветом, а целевые переменные — синим. Термы отображаются в правом верхнем углу в форме $z = f(y, x, w)$ (рис. 4).

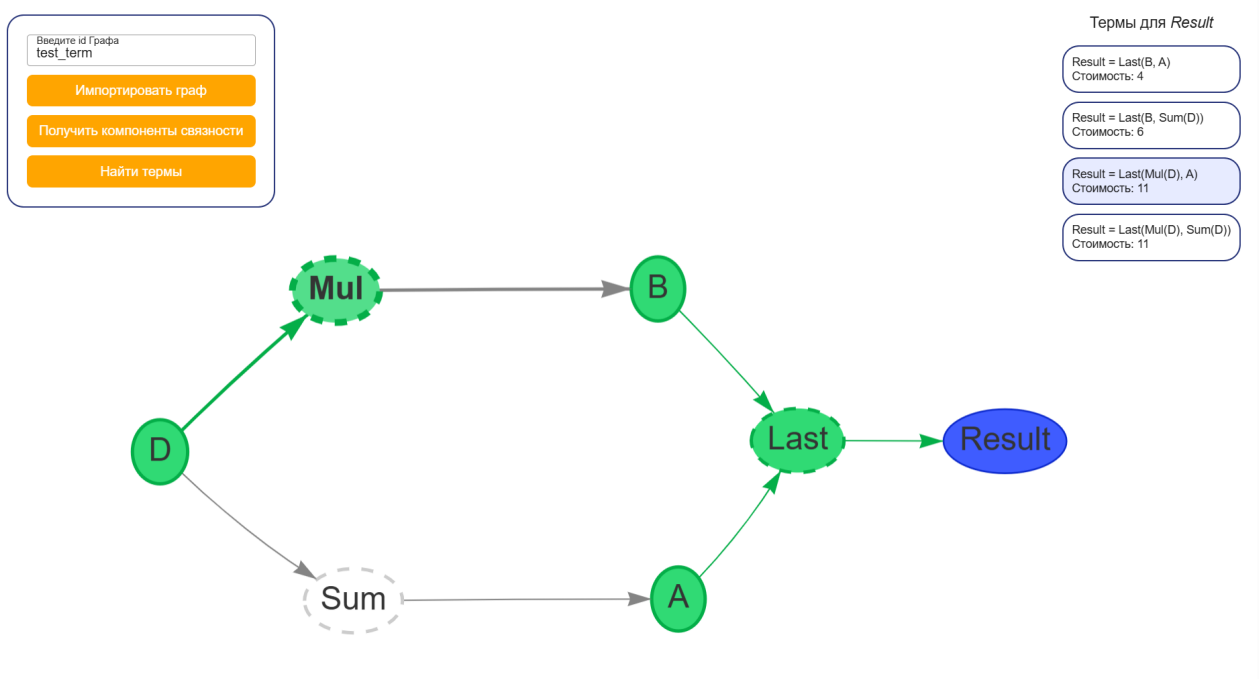


Рис. 4. Найденные термы

3.4. *База данных.* В качестве системы управления базами данных выбрана MongoDB, потому как данные в системе не имеют сложных связей и зависимостей между собой. В базе сохраняются множества термов, которые могут вычислить заданные переменные. Пример записи в базе данных, который показывает термы, вычисляющие переменную с $id = 6$.

Листинг 2. Результат поиска термов.

```
{
  "id": "681f93884fef4f937f95ac29",
  "graph_id": "big",
  "terms": [
    {
      "operations": [4],
      "variables": [5, 3],
      "result_var": 6
    },
    {
      "operations": [1],
      "variables": [0, 2],
      "result_var": 6
    }
  ],
  "result_var": 6,
  "created_at": "2025-05-10T17:57:28.777Z"
}
```

В листинге 2:

- 1) `id` — идентификатор записи, сгенерированный MongoDB.
- 2) `graph_id` — название файла, содержащего ПВМ.
- 3) `terms` — список термов (список `id` операций и переменных и `id` вычисляемой переменной).
- 4) `result_var` — `id` переменной, которую вычисляют термы.
- 5) `created_at` — дата создания записи.

3.5. Импорт ПВМ из других систем. В перспективе в описываемой системе отладки ПВМ предполагается реализовать возможность работать с множеством систем конструирования ВМ благодаря конвертации разных представлений вычислительных моделей в унифицированный формат. Для того чтобы добавить поддержку нового формата ПВМ из какой-либо системы, достаточно создать соответствующий парсер. Например, HPC2C предоставляет RESTful интерфейс [12] для управления вычислительными моделями. Имеется возможность автоматически получить доступ к вычислительным моделям, провести их анализ, модифицировать или внести сведения об ошибках в метаданные вычислительной модели, операций и переменных и сохранить исправления в хранилище моделей HPC2C.

4. Алгоритмы анализа вычислительных моделей. Выполнение проверок. После импорта графа и построения массива смежности в разрабатываемой системе выполняются проверки в соответствии с описанными ниже алгоритмами.

4.1. Алгоритм поиска компонент связности. Ориентированный граф рассматривается как неориентированный, направления дуг игнорируются. Строится система непересекающихся множеств (СНМ) по следующему алгоритму [13]:

1) Создается n множеств, где каждая вершина графа образует отдельное множество, т.е. является единственным представителем своего множества.

2) Для каждого ребра объединяются множества вершин, которые оно соединяет. Пусть ребро соединяет две вершины: x и y . Если они являются представителями одного и того же множества, то переходим к другому ребру. Иначе объединяем, присоединяя множество с меньшим рангом (мощность множества во внутреннем представлении СНМ) к множеству с большим рангом. Если ранги равны — выбираем любое множество для присоединения.

Сложность данного алгоритма построения СНМ: $O(E \times \alpha(V))$, где E — число ребер, V — число вершин, α — обратная функция Аккермана [14].

4.2. Алгоритм поиска термов. Поиск термов — основополагающая задача при анализе многих ошибок в ПВМ. Например, выполнить анализ вариантов и определить саму возможность вычисления переменной можно только при наличии информации о том, в каких термах она участвует.

Для поиска термов был реализован алгоритм обхода графа в глубину от каждой выходной переменной (листинг 2). Пусть V — множество входных переменных, а W — множество выходных, `visited` — список уже посещенных вершин. Требуется найти все термы, которые бы вычисляли каждую переменную w из W , используя лишь множество V и операции из F . Для каждого w из W выполняется процедура *build_term* (листинг 3).

Листинг 3. Алгоритм поиска термов.

```
fn build_term(graph, w, V, visited) -> Term[]:  
0: terms = []  
1: Если visited содержит w,  
   то вернуть пустой массив термов terms.  
2: Добавить w в visited.  
3: Если нет операции, вычисляющей w,  
   и w не содержится во множестве входных переменных V,  
   то вернуть пустой список термов (terms).  
   Однако если w содержится в V,  
   то вернуть единственный терм - сам w.  
4: Для каждой операции, вычисляющей w:  
   4.1: Рекурсивно построить термы для аргументов (build_term)  
   4.2: Скомбинировать все варианты  
        подтермов (что определяет множество  
        вариантов вычисления w) и добавить в terms  
5: Вернуть terms
```

Также дополнительно результаты анализа сохраняются в MongoDB для последующего использования.

4.3. *Реализованные проверки.* На данный момент система отладки ПВМ способна обнаруживать следующие ситуации:

- 1) Наличие более чем одной компоненты связности в графе ПВМ.
- 2) Отсутствие термов, которые бы могли вычислить выходную переменную по наперед заданному множеству входных (невозможность решить (V, W) -задачу).
- 3) Возможность вычисления выходной переменной различными способами (множество возможных (V, W) -планов).

Заключение. Синтез программ на вычислительных моделях — актуальное направление исследований в области автоматизации конструирования программ. Описав предметную область в виде ВМ, исследователи получают возможность решать на ней различные задачи в автоматизированном режиме, при этом процесс получения решения является отслеживаемым, объяснимым, воспроизводимым. Существуют и развиваются системы для организации решения задач на основе этого формализма.

Существенной проблемой в разработке ВМ является возможность допустить ряд ошибок, как со стороны специалиста в предметной области при конструировании ВМ, так и со стороны разработчиков при создании системы для работы с ВМ, в результате чего (V, W) -планы могут строиться не оптимальным образом по выбранным критериям или даже некорректно. Если ВМ достаточно громоздкая, то обнаружить ошибки «вручную» может быть крайне затруднительно.

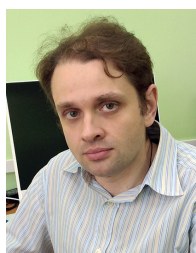
Описанная в настоящей статье система отладки предназначена для автоматизации поиска ошибок в ПВМ и (V, W) -планах. Она обладает web-интерфейсом, где пользователь может загрузить ПВМ из JSON-файла определенного формата. В перспективе будет добавлена возможность импорта ПВМ и (V, W) -планов из ряда систем управления вычислительными моделями. В интерфейсе системы отображается граф загруженной ПВМ, система осуществляет поиск ситуаций, свидетельствующих о возможных ошибках в ПВМ, результаты анализа визуализируются в интерфейсе.

В полной функциональности система будет обладать возможностями обнаружения нескольких компонент связности, невозможности вычислить какую-либо переменную из других, множественности решения (V, W) -задачи, а также будет предоставлять специалисту в предметной области возможность формализации ограничений на элементы ПВМ. На данный момент значительная часть из перечисленных функций уже реализована. Отдельной веткой исследования является разработка алгоритмов анализа ПВМ, эффективно (в смысле использования доступных ресурсов) реализуемых на параллельных вычислительных машинах, что актуально для обработки больших графов ПВМ.

Предоставляемые системой функции должны значительно сократить время специалистов, затрачиваемое на отладку ПВМ и (V, W) -планов.

Список литературы

1. Вальковский В. А., Малышкин В. Э. Синтез параллельных программ и систем на вычислительных моделях. Новосибирск: Наука, 1988. 129 с.
2. Малышкин В. Э., Корнеев В. Д. Параллельное программирование мультимикомпьютеров: учебник. Новосибирск: Изд-во НГТУ, 2006. 296 с.
3. Тыугу Э. Х. Концептуальное программирование. М.: Наука, 1984. 256 с.
4. Gorodnichev M., Lebedev, D. Semantic tools for development of high-level interactive applications for supercomputers // J. Supercomput. 2021. P. 11866–11880. DOI: <https://doi.org/10.1007/s11227-021-03731-6>.
5. Malyshkin V., Akhmed-Zaki D., Perepelkin V. Parallel programs execution optimization using behavior control in LuNA system // J. Supercomput. 2021. P. 9771–9779. DOI: 10.1007/s11227-021-03654-2.
6. Kotkas V., Ojamaa A., Grigorenko P., Maigre R., Harf M., Tyugu E. CoCoViLa as a multifunctional simulation platform // SIMUTOOLS 2011 — 4th International ICST Conference on Simulation Tools and Techniques: March 21–25. Barcelona, Spain: Brussels: ICST, 2011. P. 1–8.
7. Феоктистов А. Г., Костромин Р. О., Воскобойников М. Л., Ли-Дэ Д. И. Организация вычислительной среды разработки и применения научных рабочих процессов на основе контейнеризации // Вычислительные технологии. 2023. Т. 28. № 6. С. 151–164.
8. Курбатов М. А. Поиск ошибок во фрагментированных программах с помощью абстрактного синтаксического дерева // Материалы 62-й Международной научной студенческой конференции. Новосибирск, 2024. С. 158–159.
9. Malyshkin V., Vlasenko A., Michurov M. Automated Debugging of Fragmented Programs in LuNA System // D. Balandin et al. (Eds.): 22nd International Conference, MMST 2022, Nizhny Novgorod, Russia, November 14–17, 2022. CCIS 1750. P. 266–280. DOI: 10.1007/978-3-031-24145-1_22.
10. Власенко А. Ю., Мичуров М. А., Царев В. Д., Курбатов М. А. Построение комплекса автоматизированной отладки фрагментированных программ // Вестник НГУ. Серия: Информационные технологии. 2024. Т. 22. № 1. С. 5–20. DOI: 10.25205/1818-7900-2024-22-1-5-20.
11. Gorodnichev M. A., Nalepova E. D., Merkulova E. A., Rudych P. D., Savostyanov A. N. Automation of EEG Data Processing with HPC Community Cloud // IEEE 24th International Conference of Young Professionals in Electron Devices and Materials (EDM), Novosibirsk, Russian Federation, 2023. P. 1320–1323. DOI: 10.1109/EDM58354.2023.10225226.
12. Fielding R. T. REST: Architectural Styles and the Design of Network-Based Software Architectures. Doctoral Dissertation, University of California, 2000.
13. Ахо А. В., Хопкрофт Дж. Э., Ульман Дж. Д. Структуры данных и алгоритмы. М.: Вильямс, 2003. С. 189–197.
14. Кнут Д. Э. Искусство программирования. Т. 1. 3-е изд. М.: Вильямс, 2006. С. 174–175.



Власенко Андрей Юрьевич — канд. техн. наук, научный сотрудник лаборатории синтеза параллельных программ ИВМиМГ СО РАН, доцент кафедры параллельных вычислений НГУ, доцент ка-

федры параллельных вычислительных технологий НГТУ. Научные интересы: системы поддержки параллельных вычислений, автоматизированная отладка и оптимизация параллельных программ. Автор более 30 печатных работ. E-mail: a.vlasenko@g.nsu.ru.

Andrey Vlasenko — candidate of Engineering Sciences, associate research at the Parallel Program Synthesis Laboratory of ICMMG SB RAS, associate professor of the Parallel Computing Department of Novosibirsk State University, associate professor of the Parallel Computational Technologies Department of Novosibirsk State Technical University. Scientific interests: parallel computing support systems, automated debugging and optimization of parallel programs. Author of more than 30 publications. E-mail: a.vlasenko@g.nsu.ru.

Городничев Максим Александрович — научный сотрудник Института вычислительной математики и математической геофизики СО РАН. Старший преподаватель в Новосибирском государственном университете и

Новосибирском государственном техническом университете. Научные интересы: автоматизация решения задач на суперкомпьютерах, интеллектуальные системы, разработка научного программного обеспечения. E-mail: maxim@ssd.sccc.ru.



Maksim Gorodnichev — researcher at the Institute of Computational Mathematics and Mathematical Geophysics SB RAS. Senior Lecturer at Novosibirsk State University and Novosibirsk State Technical University. Research interests: automation of solving problems on supercomputers, intelligent systems, scientific software development. E-mail: maxim@ssd.sccc.ru.



Курбатов Максим Андреевич — магистрант 2 года обучения факультета информационных технологий НГУ. Научные интересы: параллельное программирование, системы поддержки параллельных вычислений. E-mail: m.kurbatov@g.nsu.ru.

Maksim Kurbatov — 2nd year undergraduate student of the Information Technology Faculty of NSU. Research interests: parallel programming, parallel computing support systems. E-mail: m.kurbatov@g.nsu.ru.

Дата поступления — 18.02.2026