Сибирское отделение
Российской академии наук

ISSN 2073-0667

ПРОБЛЕМЫ ИНФОРМАТИКИ

ПРОБЛЕМЫ ИНФОРМАТИКИ № 2 2024

ТЕОРЕТИЧЕСКАЯ И СИСТЕМНАЯ ИНФОРМАТИКА

ПРИКЛАДНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

2

2024

ПРОБЛЕМЫ ИНФОРМАТИКИ № 2 (63) 2024 г.

Журнал выходит ежеквартально, издается с 2008 г.

Учредитель журнала — Институт вычислительной математики и математической геофизики СО РАН.

Адрес редакции, издателя: 630090, г. Новосибирск, просп. Академика Лаврентьева, д. 6, ИВМ и МГ СО РАН
тел. (383) 330-96-43; e-mail: problem-info@sscc.ru, http://www.problem-info.sscc.ru.

Журнал зарегистрирован в Федеральной службе по надзору в сфере массовых коммуникаций, связи и охраны культурного наследия. Свидетельство ПИ № ФС77-32088 от 27 мая 2008 г. Журнал распространяется по подписке. Оформление подписки на сайте «Объединенного каталога „Пресса России“» https://www.pressa-rf.ru/cat/1/edition/y_e69980/, подписной индекс 69980, и через интернет-магазин «Пресса по подписке» https://www.akc.ru/itm/problemy_i-informatiki/. Цена свободная. Журнал распространяется на территории России.

Журнал включен в Перечень ведущих рецензируемых научных журналов, рекомендованных для публикаций Высшей аттестационной комиссией.

Отпечатано в типографии «АЛЕКСПРЕСС» ИП Малыгин Алексей Михайлович.
Адрес: 630090, Новосибирск, пр-т Академика Лаврентьева, 6/1, оф. 104, тел. +7 (383) 217-43-46.
Формат 60 × 84 1/8. Усл. печ. л. 11,63. Печать офсетная.
Тираж 50 экз. Заказ № 911. Подписано в печать 20.06.2024 г. Выход в свет 30.06.2024 г.

JOURNAL "PROBLEMS OF INFORMATICS". No. 2 (63) 2024

Publisher: Institute of Computational Mathematics and Mathematical Geophysics of Siberian Branch of Russian Academy of Sciences.

Address of the editorial office: 630090, pr. Lavrentieva, 6, Novosibirsk, Russia, Institute of Computational Mathematics and Mathematical Geophysics of SB RAS.
Phone: +7 (383) 330-96-43; e-mail: problem-info@sscc.ru, http://www.problem-info.sscc.ru.

The journal has been registered in accordance with Legislation of the Russian Federation. Certificate of Mass Media Registration: ПИ № ФС77-32088, of 27 May, 2008, ISSN 2073-0667. The journal is distributed in Russia.

The journal "Problems of Informatics" is in the List of Peer-Reviewed Scientific Journals for publication of scientific results of Ph.D. and Dr. of Sci.

СОДЕРЖАНИЕ

Теоретическая и системная информатика

Прикладные информационные технологии

ПРОБЛЕМЫ ИНФОРМАТИКИ
№ 2 (63) 2024 г.

CONTENTS

Theoretical informatics

Applied information technologies

PROBLEMS OF INFORMATICS
N 2 (63) 2024

CALCULATION OF THE RELIABILITY OF EXTENDED TRI-CONNECTED NETWORKS

P. O. Perminov, D. A. Migov

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

DOI: 10.24412/2073-0667-2024-2-5-15
EDN: VZCSXV

When analyzing the reliability of networks for various purposes, the apparatus of random graphs is usually used. The most common indicator of reliability is the probability of connectivity of a random graph with unreliable edges, which describes the reliability of a network in terms of the ability to establish a connection between each pair of network nodes. However, the problem of calculating the probability of network connectivity is NP-hard. To reduce the dimensional when carrying out precise calculations, methods based on the use of structural features of networks are widely used, primarily various methods of reduction and decomposition.

Networks with an extended structure are used in number of applications. These are, for example, networks located in extended objects — mines, ships, other objects. Linear wireless sensor networks, designed for monitoring various long-distance objects, such as pipelines, bridges, roads, also have an extended structure. Despite their linear physical structure, the topological graph of such a network can be either linear or non-linear, since wireless communication channels are possible not only between the nearest neighboring nodes. For example, if each node can communicate with three nodes on the right and three nodes on the left, we obtain a network containing a group of three-vertex cross separators.

If the graph of an extended network is linear, then calculating its probabilistic connectivity is not difficult. The use of a serial-parallel transformation, or other techniques, allows us to make the calculation within polynomial complexity. If the network graph is biconnected and contains a separators of two nodes, then the calculation can be significantly accelerated by using decomposition along these separators.

In this paper, we study the possibility of quickly calculating the reliability of extended three-connected networks using decomposition according to the previously proposed formula. Such decomposition will lead to the production of 10 new extended graphs of a smaller size. As experiments have shown, this approach is quite effective and makes it possible to calculate the reliability of extended networks, for which it is not possible to calculate the reliability using an accurate method.

Key words: network reliability, random graph, triconnected graph, probabilistic connectivity, factorization method, network decomposition, cut, separator.

The work was carried out within the framework of the project N 0251-2021-0005 of a state assignment of the Institute of Computational Mathematics and Mathematical Geophysics SB RAS.

References

1. Zhukovskij M. E., Rajgorodskij A. M. Sluchajny‘e grafy‘: modeli i predel‘ny‘e xarakteristiki // Uspexi matematicheskix nauk. 2015. T. 70. N 1 (421). P. 35–88.

2. Mochalov V. A., Mochalova A. V. Primenenie e‘kspertny‘x sistem dlya rascheta veroyatnosti svyaznosti mezhdu uzlami grafa // V sbornike: Gibridny‘e i sinergeticheskie intellektual‘ny‘e sistemy‘. Materialy‘ V Vserossijskoj Pospelovskoj konferencii s mezhdunarodny‘m uchastiem. Pod redakciej A. V. Kolesnikova. 2020. P. 226–235.

3. Rodionov A. S. Mozhno li dobit‘sya dal‘nejshego uskoreniya rascheta xarakteristik svyaznosti sluchajnogo grafa? // Problemy‘ informatiki. 2022. N 4 (57). P. 39–52.

4. Valiant L. The complexity of enumeration and reliability problems. // SIAM Journal on Computing. 1979. T. 8. N 3. P. 410–421.

5. Rodionova O. K., Rodionov A. S., Choo H. Network probabilistic connectivity: exact calculation with use of chains // Lecture Notes in Computer Science. 2004. T. 3045. C. 315–324.

6. Satyanarayana A., Wood R. K. A linear-time algorithm for computing K—terminal reliability in series-parallel networks // SIAM. J. Comput. 1985. T. 14. P. 818–883.

7. Migov D., Rodionova O., Rodionov A., Choo H. Network probabilistic connectivity: using node cuts // Springer Lecture Notes in Computer Science (in EUC Workshops). V. 4097, 2006, P. 702–709.

8. Tarxanova O. Yu., Shaxov V. V. K voprosu ocenki e‘ffektivnosti besprovodny‘x sensorny‘x setej // Problemy‘ informatiki. 2020. N 1 (46). P. 35–65.

9. Farxadov M. P. O., Blinova O. V., Vas‘kovskij S. V. Ocenka nadezhnosti sistemy‘ svyazi s podvizhny‘mi uzlami // Datchiki i sistemy‘. 2018. N 5 (225). P. 3–8.

10. Shaxov V. V., Chen X., Yurgenson A. N., Loshkarev A. V. K voprosu ocenki nadezhnosti linejny‘x besprovodny‘x sensorny‘x setej // Problemy‘ informatiki. 2022. N 4 (57). P. 120–128.

11. Mohamed N., Al-Jaroodi J., Jawhar I., Lazarova-Molnar S. Failure impact on coverage in linear wireless sensor networks // 2013 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Toronto, ON, Canada. IEEE Press, 2013. P. 188–195.

12. Migov. D. A. Dissertaciya na soiskanie uchenoj stepeni kandidata fiziko-matematicheskix nauk “Raschyot veroyatnosti svyaznosti sluchajnogo grafa s primeneniem sechenij”. Novosibirsk: ICMMG SB RAS. 2008. 97 P.

13. Migov D. A. Ispolzovanie vershinnyx razrezov dlya tochnogo vychisleniya veroyatnosti svyaznosti seti // Trudy Mezhdunarodnoj konferencii “Vychislitelnye i informacionnye texnologii v nauke, texnike i obrazovanii” (Pavlodar, 20–22 sentyabrya 2006 goda) Tom II. S. 51–58.

14. Page L. B., Perry J. E. A Practical Implementation of the Factoring Theorem for Network Reliability // IEEE transactions on reliability. 1988. V. 37, N 3. P. 259–267.

15. D. Migov. Dekompoziciya seti po secheniyam pri raschyote eyo nadyozhnosti // Prikladnaya diskretnaya matematika. 2020. N 47. P. 62–86.

16. Burgos J. M. Factorization of network reliability with perfect nodes II: Connectivity matrix // Discrete Applied Mathematics. 2016. V. 198. P. 91–100.

РАСЧЕТ НАДЕЖНОСТИ ПРОТЯЖЕННЫХ ТРЕХСВЯЗНЫХ СЕТЕЙ

П. О. Перминов, Д. А. Мигов

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

УДК 621.311.1+519.17
DOI: 10.24412/2073-0667-2024-2-5-15
EDN: VZCSXV

Рассматривается задача расчета вероятности связности протяженных сетей с ненадежными каналами связи. Точный расчет данного показателя — NP-трудная задача, что делает его затруднительным для сетей реальной размерности.

Предполагается, что сеть имеет протяженную структуру, что характерно для сетей, расположенных в шахтах, кораблях, других протяженных обьектов, а также линейных беспроводных сенсорных сетей, предназначенных для мониторинга различных протяженных объектов, таких как трубопроводы. В отличие от ранее исследованного нами случая двусвязной сети, здесь мы предполагаем, что сеть трехсвязная и имеет группу поперечных сечений, т. е. вершинных разрезов.

Для ускорения расчета надежности подобных сетей предлагается использовать структурную декомпозиции сети на основе формулы разложения по трехвершинному сечению. Этот метод, с использованием рекурсивного обхода сечений, ускоряет расчет надежности протяженных сетей по сравнению с известными методами. Результаты численных экспериментов подтверждают эффективность предлагаемого подхода.

Ключевые слова: надежность сети, случайный граф, трехсвязный граф, вероятность связности, метод факторизации, декомпозиция сети, сечение, сепаратор.

Введение. При анализе надежности сетей различного назначения пользуются, как правило, аппаратом случайных графов [1]. Наиболее распространенным показателем надежности является вероятность связности вершин случайного графа с ненадежными ребрами, описывающая надежность сети с точки зрения возможности установления соединения между каждой парой узлов сети. Этот показатель является наиболее универсальным, позволяет описывать способность функционирования сетей различного назначения в условиях независимых отказов элементов [2–3]. Однако, задача расчета вероятности связности сети является NP-трудной [4]. Для снижения размерности при проведении точного расчета широко используются методы, основанные на использовании структурных особенностей сетей, в первую очередь — различные методы структурной редукции и декомпозиции [5–7].

Для ряда прикладных областей характерны сети с протяженной структурой. Это, в первую очередь, сети, расположенные в протяженных объектах — шахтах, кораблях и

Работа выполнена в рамках проекта № 0251-2021-0005 ПФИ ИВМиМГ СО РАН.

прочих объектах. Другой важный класс протяженных сетей — это беспроводные сенсорные сети [8–9], предназначенные для мониторинга различных протяженных объектов, таких как трубопроводы, мосты, дороги, и других объектов. Такие сети выделяют в особый класс — так называемые линейные беспроводные сенсорные сети [10]. Несмотря на их линейную физическую структуру, топологически граф такой сети может быть как линейным, так и нелинейным [11], так как возможны каналы беспроводной связи не только между ближайшими соседними узлами.

Если граф протяженной сети является линейным, то расчет ее вероятности связности не представляет труда — использование последовательно-параллельного преобразования [6] или других приемов позволит сделать расчет за полиномиальную трудоемкость. Если граф сети — двусвязный и содержит сечение (сепараторы) из двух узлов, то расчет значительно ускорится использованием декомпозиции по этому сечению [7]. Однако, если сеть имеет протяженную структуру, то и граф ее может содержать множество поперечных сечений, каждое из которых разделяет граф на также протяженные компоненты. Эффективные алгоритмы расчета надежности таких сетей, основанные на рекурсивном использовании декомпозиции по двухвершинному сечению, изучались нами ранее в [12].

В данной статье мы изучаем возможность быстрого расчета надежности протяженных трехсвязных сетей при помощи декомпозиции по формуле из [13]. Например, если в линейной беспроводной сенсорной сети каждый узел может связываться с тремя узлами справа и тремя узлами слева [11], получаем сеть, содержащую группу трехвершинных поперечных сечений. В отличие от случая с сечением из двух вершин, когда мы при рассмотрении очередного сечения мы переходим к 4 новым протяженным графам меньшего размера, декомпозиция приведет к получению 10 новых протяженных графов меньшего размера. Однако, как показано в экспериментах, такой подход является достаточно эффективным и позволяет делать расчет надежности протяженных сетей, для которых посчитать надежность не представляется возможным точным методом, в данном случае — методом факторизации [14].

1. Основные определения и обозначения. Рассмотрим неориентированный граф $G = (V,E)$, где V — это множество вершин, а E — множество ребер графа G. Пусть для каждого ребра задана вероятность его присутствия в графе, при этом предполагается, что вершины абсолютно надежны.

Элементарным событием будем называть частную реализацию графа, определяемую исправностью или отказом каждого ребра.

Вероятность элементарного события равна произведению вероятностей присутствия исправных ребер, умноженному на произведение вероятностей отсутствия отказавших ребер.

Произвольное событие (событие есть объединение некоторых элементарных событий) будем называть *успешным*, если в каждом элементарном событии, входящем в это событие, вершины могут быть связаны исправными ребрами.

Вероятность связности графа G, $R_K(G)$ есть вероятность того, что вершины связаны исправными ребрами, то есть вероятность события, состоящего из всех успешных событий и только из них.

2. Декомпозиция сети по сечению. Предположим, что графа G содержит сечение (сепаратор) из трех вершин, которое разделяет его на два подграфа G_1 и G_2 (рис. 1). В таком случае расчет $R(G)$ может быть существенно ускорен при помощи соответствующей формулы, для представления которой понадобятся следующие обозначения [15].

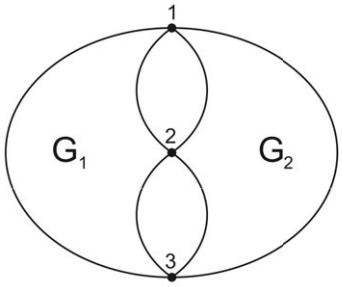

Рис. 1. Граф с сечением из трех вершин

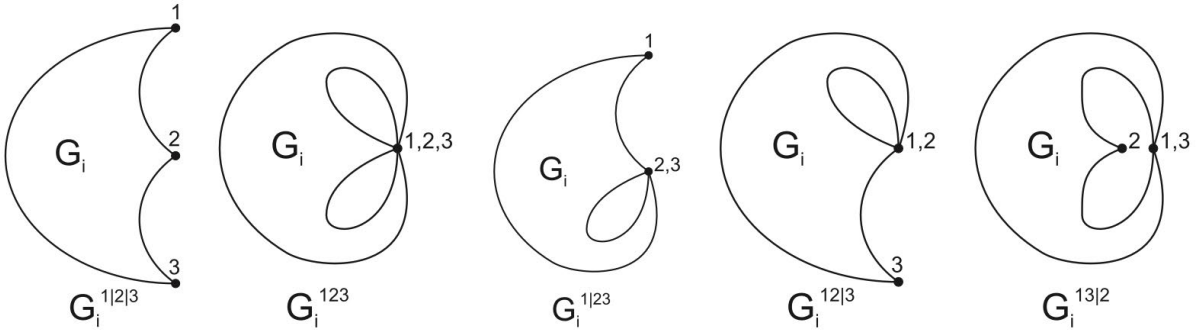

Рис. 2. Графы, стянутые по разбиениям

Через S обозначим множество всех разбиений множества вершин сечения. Произвольное разбиение $\{\{v_1 \dots v_l\}, \dots \{u_1 \dots u_t\}\} \in S$ будем обозначать для простоты изложения как:

$$v_1 \dots v_l| \dots |u_1 \dots u_t = \{\{v_1 \dots v_l\}, \dots \{u_1 \dots u_t\}\}.$$

Через $G_i^{\Phi}(\Phi \in S)$ обозначим граф G_i, стянутый по каждому элементу из Φ, то есть если u и v входят в один и тот же элемент Φ, то в графе G_i^{Φ} вершины u и v стянуты в одну. Всего имеется 5 разбиений, образующих множество S. Соответствующие графы, стянутые по разбиениям, приведены на рис. 2.

Тогда для графа G с сечением, состоящим из вершин $1, 2, 3$, которое разделяет его на два подграфа G_1 и G_2, справедлива формула:

$$\begin{aligned} R(G) = \frac{1}{2}\Big[& R(G_1^{1|23})\left(R(G_2^{12|3}) + R(G_2^{13|2}) - R(G_2^{1|23})\right) + \\ & R(G_1^{12|3})\left(R(G_2^{1|23}) + R(G_2^{13|2}) - R(G_2^{12|3})\right) + \\ & R(G_1^{13|2})\left(R(G_2^{12|3}) + R(G_2^{1|23}) - R(G_2^{13|2})\right) - \\ & R(G_1)\left(R(G_2^{12|3}) + R(G_2^{13|2}) + R(G_2^{1|23})\right) - \\ & R(G_2)\left(R(G_1^{12|3}) + R(G_1^{13|2}) + R(G_1^{1|23})\right) + \\ & R(G_1)R(G_2)\Big] + R(G_1)R(G_2^{123}) + R(G_1^{123})R(G_2). \end{aligned} \tag{1}$$

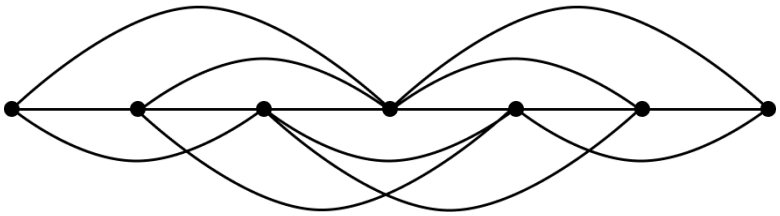

Рис. 3. Граф топологии ЛБСС с 7 узлами

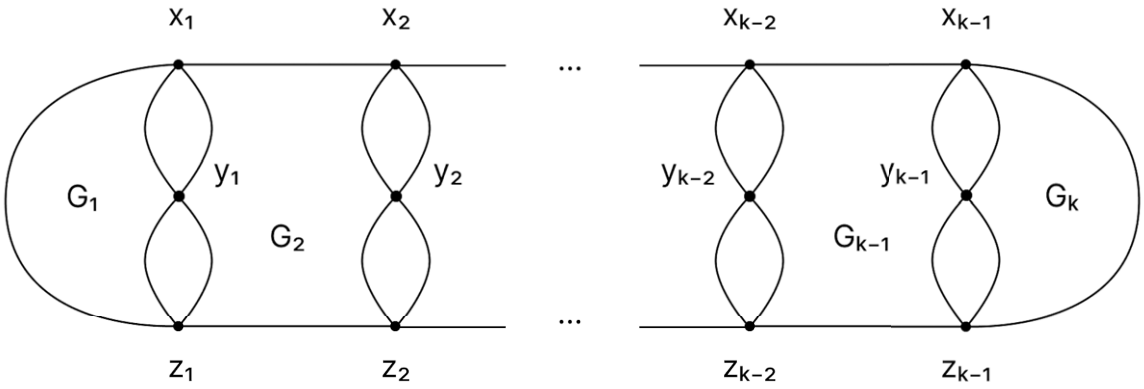

Рис. 4. Продольный трехсвязный граф

Эта формула была представлена нами впервые в 2006 в [13]. Позже, в 2016, она же была опубликована Х. М. Бургосом [16]. Как показали эксперименты, этот подход значительно сокращает расчет, несмотря на переход к рассмотрению 10 графов. В протяженных сетях может возникать множество таких сечений, последовательно разделяющих исходную сеть. Будем называть такие сечения поперечными. Например, в линейной беспроводной сенсорной сети, при наличии каналов беспроводной связи между каждым узлом и тремя узлами справа и слева (рис. 3), каждая тройка соседних узлов будет сечением, за исключением крайних троек.

Рассмотрим общий случай, когда сеть содержит группу подобных сечений. Граф соответствующей структуры будем называть продольным трехсвязным. Структура подобного графа приведена на рис. 4. Ниже мы приводим формальное определение этого объекта.

Пусть $G_1, G_2, \ldots, G_k$ — последовательность связных графов, таких, что:
1) Графы G_i, G_{i+1}, то есть соседние графы, имеют ровно три общие вершины x_i, y_i, z_i и не имеют общих ребер;
2) Графы G_i, G_j при $|i-j| > 1$ не имеют общих элементов.
Тогда граф $G = \bigcup_{i=1}^{k} G_i$ назовем *продольным трехсвязным графом*, а подграфы $G_i, i = 1, \ldots, k$ — его *компонентами*.

Каждая тройка вершин $\{x_i, y_i, z_i | 1 \leqslant i \leqslant k-1\}$ образует сечение в графе G. Множество $\bigcup_{i=1}^{k-1}\{\{x_i, y_i, z_i\}\}$ назовем *продольным 3-сечением* в графе G $\{x_i, y_i, z_i\}$ — *компонента* продольного сечения или *поперечное сечение*. Продольное сечение является *максимальным*, если к нему не может быть добавлено ни одно трехвершинное сечение так, чтобы оно осталось продольным. Два поперечных сечения $\{x_i, y_i, z_i\}$ и $\{x_j, y_j, z_j\}$ *соседние*, если $|i-j| = 1$.

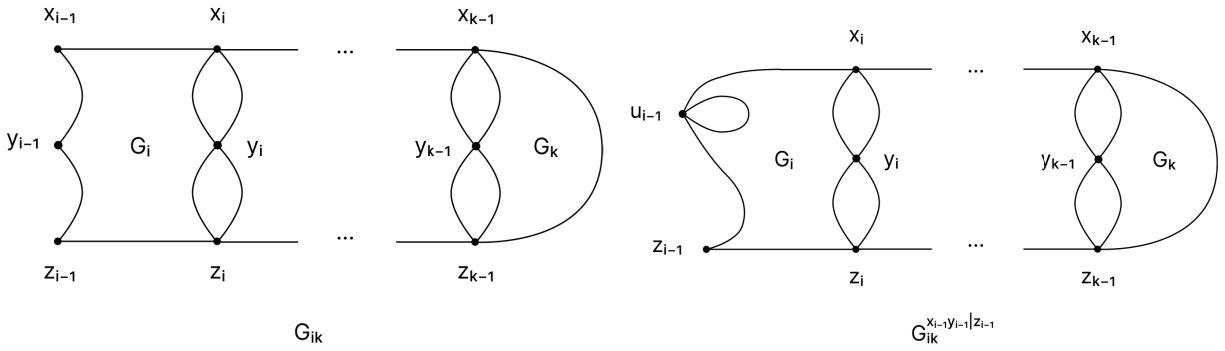

Рис. 5. Примеры промежуточных трехсвязных продольных графов

При использовании формулы (1) при последовательном переборе компонент продольного сечения, кроме G_i, будут возникать подграфы $G_i^{\Phi,\Psi}$, $\Phi, \Psi \in \{1|23, 12|3, 13|2, 123, 0\}$. Кроме того, будут возникать новые продольные трехсвязные графы вида (рис. 5)

$$G_{ik} = \bigcup_{i \leqslant j \leqslant k} G_j, 1 \leqslant i \leqslant k;$$

$$G_{ik}^{\Phi} = G_i^{\Phi,0} \bigcup_{i+1 \leqslant j \leqslant k} G_j, 1 < i \leqslant k,$$

где для краткости 0 соответствует отсутствию стягиваний, то есть $0 = 1|2|3$ при $i = 2, \ldots, k-1$, либо же отсутствию вторых 3-сечений для крайних случаев, и Φ отвечает за стягивание вершин с левой стороны графа G_i, а Ψ — с правой. При дальнейшем применении формулы (1) к новым продольным трехсвязным графам будут возникать графы вышеописанного типа.

3. Тестовые результаты разработанных алгоритмов. В данной главе представлен алгоритм *3-Cuts Long*, который основан на результатах, полученных в предыдущей главе, а также проанализированы результаты численных экспериментов. Вычисления были выполнены на ПЭВМ с процессором Apple M1, 3.22 GHz, RAM: 8Gb.

3.1. *Описание алгоритма.* Ниже представлены результаты сравнения времени расчета для трех методов, включая расчет без использования декомпозиции по сечениям (метод факторизации *Factoring* [14]), усиленный последовательно-параллельным преобразованием [5] и окончанием рекурсивных вызовов на графах с пятью вершинами. Два других метода (3-*Cuts Long* и 3-*Cuts*) используют этот метод для расчета надежности компонент. Для нахождения максимального продольного сечения необходимо перебрать все тройки узлов сети с проверкой на связность графа после удаления каждой из них, что занимает $O(MN^3)$ операций.

Полученный алгоритм 3-*Cuts* можно описать тремя основными пунктами:

1) Поиск продольного сечения продольного графа G;
2) Декомпозиция графа по выбранному сечению из 3 узлов;
3) Расчет надежности графов $R(G_{ik}^{\Phi})$ рекурсивно переходом к п. 2;
4) Расчет надежности графов $R(G_i^{\Phi,\Psi})$ каким-либо точным методом;
5) Вычисление $R(G)$ по формулам (1).

Отдельно стоит описать первый пункт указанного выше алгоритма:

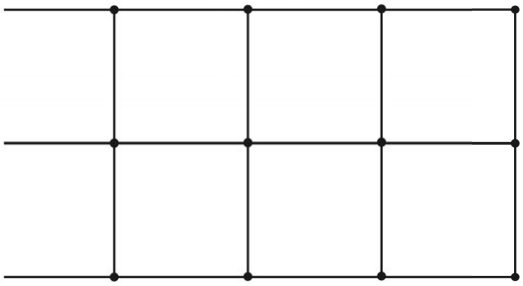

Рис. 6. Решетка 3×5

Таблица 1

Решетки $3 \times L$

L	*3-Cuts Long*			*3-Cuts*	*Factoring*	*R(G)*
	$\beta = 5$	$\beta = 6$	$\beta = 7$	β_*		
8	0.0003 s	0.0004 s	0.0005 s	0.0009 s	0.000529 s	0.462632
18	0.009 s	0.01 s	0.005 s	0.035 s	23.718 s	0.236108
20	0.04 s	0.09 s	0.03 s	0.088 s	214.411 s	0.206388

Сперва скажем, что продольное сечение хранится так, что для каждой внутренней компоненты G_i графа G хранятся тройки номеров вершин для сечения слева и сечения справа. То есть продольный граф — это $2 + 2(k-2)$ троек, где k — число компонент продольного графа.

Таким образом, поиск продольного сечения (алгоритм *CutCut*) для графа G можно описать следующими пунктами:

1) находим 3-сечение в графе G и формируем подграфы G_1, G_2:

2) если существуют соседние сечения, то проверяем, чтобы новое сечение не пересекалось с соседними;

3) проверяем, чтобы каждое старое сечение оставалось в одном из новых подграфов, и старые сечения лежали в разных подграфах;

4) вызываем алгоритм *CutCut* для G_1;

5) добавляем в продольное сечение 3-сечение (тройку) для G_1 и 3-сечение для G_2;

6) вызываем алгоритм *CutCut* для G_2.

Этот же параметр был использован и для расчета надежности рекурсивно с пересчетом компонент алгоритмом 3-*Cuts*. Экспериментально было получено, что оптимально выбирать для такого алгоритма β равным приблизительно половине вершины, что логично, так как исходный граф G будет разбит 3-сечением на приблизительно одинаковые подграфы (по количеству вершин), что позволяет за одно время расчитывать каждый подграф. Таким образом, оптимальное β обозначим, как β_*, и $\beta_* \approx \frac{|V|}{2}$.

Заключение. 3.2. *Численные эксперименты.* Численные эксперименты были проведены на графах решетках $3 \times L$ при $L = 8, 18, 20$. Вероятность присутствия каждого ребра положим равной 0,75. В таблице 1 приведены время работы алгоритмов для расчета надежности указанных решеток и вычисленное значение надежности. Видно, что с ростом L расчет факторизацией уже не представляется возможным. Использование же подхода с декомпозицией делает такой расчет достаточно быстрым, применение же продольных сечений и алгоритма *3-Cuts Long* позволяет дополнительно ускорить расчет.

Таблица 2

Графы для ЛБСС

$\lvert V\rvert$	*3-Cuts Long*			*3-Cuts*	*Factoring*	*R(G)*
	$\beta = 5$	$\beta = 6$	$\beta = 7$	β_*		
20	0.0005 s	0.0004 s	0.0013 s	0.0009 s	0.048 s	0.951241
28	0.0008 s	0.001 s	0.0013 s	0.009 s	11.93 s	0.947482
36	0.011 s	0.009 s	0.003 s	0.127 s	2991 s	0.943739

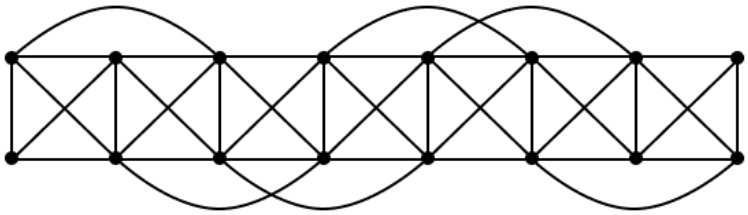

Рис. 7. Граф для ЛБСС в удобном представлении, $|V| = 16$

Также были рассмотрены графы, которые напрямую относятся к линейным беспроводным сенсорным сетям. Узлы графов таких сетей могут быть расположены на одной прямой и имеют каналы связи не только с соседними узлами (см. рис. 8).

Такие графы можно изобразить и более удобным способом (см. рис. 9), где уже наглядно можно увидеть наличие продольного трехвершинного сечения. Для экспериментов были выбраны графы, где каждый узел, кроме крайних, был связан с тремя узлами слева и с тремя узлами справа. Для всех ребер таких графов надежность положим равной 0,75. Время работы алгоритмов на таких графах и их надежности указаны в табл. 2.

Результаты показывают, что разумное время вычисления надежности на таких графах дают алгоритмы 3-*Cuts Long* и 3-*Cuts*, при это с ростом числа вершин алгоритм 3-*Cuts Long* показывает большее ускорение относительно 3-*Cuts*.

В работе предложены новые способы повышения эффективности разработанных ранее методов расчета вероятности связности множества вершин случайного неориентированного графа с абсолютно надежными вершинами и ненадежными ребрами, основанных на декомпозиционном подходе. В частности, была развита идея совместного расчета методом факторизации вероятностей связности подграфов, получающихся при декомпозиции по 2-сечению исходного случайного графа, которая возникла из наблюдений об их структурной схожести. В итоге было разработано и протестировано три новых алгоритма. Согласно проведенным экспериментам, два из трех алгоритмов совместного расчета показывают на всех тестах лучшие вычислительные и временные результаты, нежели имеющийся на текущий момент метод независимого расчета, что может свидетельствовать о целесообразности их применения и последующего развития.

Дальнейшие исследования по рассматриваемой теме могут быть связаны с проведением более обширных тестирований, дополнительными модификациями и оптимизациями предложенных алгоритмов, основанных на изучении стратегий факторизации, обобщением результатов для варианта 3-сечений и более общих случаев, а также с расчетом иных показателей надежности сетей, в том числе с расчетом вероятности связности некоторого подмножества вершин графа.

Список литературы

1. Жуковский М. Е., Райгородский А. М. Случайные графы: модели и предельные характеристики // Успехи математических наук. 2015. Т. 70. № 1 (421). С. 35–88.

2. Мочалов В. А., Мочалова А. В. Применение экспертных систем для расчета вероятности связности между узлами графа // В сборнике: Гибридные и синергетические интеллектуальные системы. Материалы V Всероссийской Поспеловской конференции с международным участием. Под редакцией А. В. Колесникова. 2020. С. 226–235.

3. Родионов А. С. Можно ли добиться дальнейшего ускорения расчета характеристик связности случайного графа? // Проблемы информатики. 2022. № 4 (57). С. 39–52.

4. Valiant L. The complexity of enumeration and reliability problems. // SIAM Journal on Computing. 1979. Т. 8. N 3. P. 410–421.

5. Rodionova O. K., Rodionov A. S., Choo H. Network probabilistic connectivity: exact calculation with use of chains // Lecture Notes in Computer Science. 2004. Т. 3045. С. 315–324.

6. Satyanarayana A., Wood R. K. A linear-time algorithm for computing K—terminal reliability in series-parallel networks // SIAM. J. Comput. 1985. Т. 14. P. 818–883.

7. Migov D., Rodionova O., Rodionov A., Choo H. Network probabilistic connectivity: using node cuts // Springer Lecture Notes in Computer Science (in EUC Workshops). V. 4097, 2006, P. 702–709.

8. Тарханова О. Ю., Шахов В. В. К вопросу оценки эффективности беспроводных сенсорных сетей // Проблемы информатики. 2020. № 1 (46). С. 35–65.

9. Фархадов М. П. О., Блинова О. В., Васьковский С. В. Оценка надежности системы связи с подвижными узлами // Датчики и системы. 2018. № 5 (225). С. 3–8.

10. Шахов В. В., Чен Х., Юргенсон А. Н., Лошкарев А. В. К вопросу оценки надежности линейных беспроводных сенсорных сетей // Проблемы информатики. 2022. № 4 (57). С. 120–128.

11. Mohamed N., Al-Jaroodi J., Jawhar I., Lazarova-Molnar S. Failure impact on coverage in linear wireless sensor networks // 2013 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Toronto, ON, Canada. IEEE Press, 2013. P. 188–195.

12. Мигов. Д. А. Диссертация на соискание ученой степени кандидата физико-математических наук «Расчет вероятности связности случайного графа с применением сечений». Новосибирск: ИВМиМГ СО РАН. 2008. 97 с.

13. Мигов Д. А. Использование вершинных разрезов для точного вычисления вероятности связности сети // Труды Международной конференции «Вычислительные и информационные технологии в науке, технике и образовании» (Павлодар, 20–22 сентября 2006 года). Том II. С. 51–58.

14. Page L. B., Perry J. E. A Practical Implementation of the Factoring Theorem for Network Reliability // IEEE transactions on reliability. 1988. V. 37, N 3. P. 259–267.

15. Мигов Д. А. Декомпозиция сети по сечениям при расчете ее надежности // Прикладная дискретная математика. 2020. № 47. С. 62–86. DOI: 10.17223/20710410/47/6.

16. Burgos J. M. Factorization of network reliability with perfect nodes II: Connectivity matrix // Discrete Applied Mathematics. 2016. V. 198. P. 91–100.

Перминов Павел Олегович — бакалавр ММФ НГУ, магистрант первого курса ММФ НГУ, e-mail: p.perminov@g.nsu.ru.

Перминов Павел окончил в 2023-м году механико–математический факультет Новосибирского государственного университета по направлению «Математика и компьютерные науки» (02.03.01, бакалавриат), защитив выпускную квалификационную работу на тему «Быстрые методы расчета надежности протяженных сетей». В настоящее время является студентом магистратуры механико–математического факультета Новосибирского государственного

университета по направлению «Математика и механика» (01.04.00, магистратура). Проходит специализацию в Институте вычислительной математики и математической геофизики Сибирского отделения Российской академии наук. Является лауреатом 61-й Международной научной студенческой конференции МНСК–23, Новосибирск. Область научных интересов включает в себя теорию графов и методы анализа надежности сетей.

Perminov Pavel graduated in 2023 from the Department of Mechanics and Mathematics of Novosibirsk State University (Novosibirsk, Russia) in the direction of "Mathematics and Computer Science" (02.03.01, bachelor's degree), having defended his final qualifying work on the topic "Fast methods for calculating the reliability of extended networks". Currently, he is a master's student of the Faculty of Mechanics and Mathematics of Novosibirsk State University in the direction of "Mathematics and Mechanics" (01.04.00, Master's degree). He is specializing at the Institute of Computational Mathematics and Mathematical Geophysics of the Siberian Branch of the Russian Academy of Sciences. He is a laureate of the third degree of the 61st International Scientific Student Conference ISSC–23, Novosibirsk. His research interests include graph theory and network reliability analysis.

Мигов Денис Александрович — канд. физ.-мат. наук, старш. науч. сотр. Института вычислительной математики и математической геофизики СО РАН, e-mail: mdinka@rav.sscc.ru.

Денис Мигов окончил в 2003 году Механико-математический факультет Новосибирского государственного университета, получив квалификацию «Математик, системный программист» по специальности «Прикладная математика и информатика». В 2008 году защитил диссертацию «Расчет вероятности связности случайного графа с применением сечений» на соискание ученой степени кандидата физико-математических наук по специальности 05.13.18 — «Математическое моделирование, численные методы и комплексы программ» в диссертационном совете при Институте вычислительной математики и математической геофизики Сибирского отделения Российской академии наук. В настоящее время является старшим научным сотрудником лаборатории Системного моделирования и оптимизации указанного института. Денис Мигов является дважды лауреатом Именной премии правительства Новосибирской области в 2011 г. и в 2015 г, также неоднократно становился призером различных конференций. Область его научных интересов включает в себя теорию графов, методы анализа надежности сетей, структурную оптимизацию сетей, переупорядочивание разреженных матриц, беспроводные сенсорные сети и параллельные алгоритмы на графах и сетях.

Denis Migov received the diploma of mathematician and programmer in applied mathematics and informatics from the Novosibirsk State University, Novosibirsk, Russia, in 2003. In 2008, he received Ph. D. (Candidate of science) degree in the field of Mathematical modeling, numerical methods, and program complexes from the Institute of Computational Mathematics and Mathematical Geophysics of the Siberian Branch of the Russian Academy of Sciences. He is a senior research fellow at the Laboratory of System modeling and optimization of the named institute. Denis Migov is twice a laureate of the Name Prize of the Government of the Novosibirsk Region in 2011 and in 2015; he also repeatedly became a prize-winner of various conferences. His scientific interests are in graph theory, network reliability analysis, network topology optimization, spares matrix reordering, wireless sensor networks, and parallel algorithms on graphs and networks.

Дата поступления — 06.12.2023

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., Malyshkin V. E.; Onv. red. Kotov V. E.; AN SSSR, Sib. otd-nie, VC. Novosibirsk: Nauka. Sib. otd-nie, 1988. 126 s (In Russian).

2. Malyshkin V. E. Tekhnologiya fragmentirovannogo programmirovaniya // Vestnik YuUrGU. Seriya: Vychislitelnaya matematika i invormatika. 2012. N 46 (305) (In Russian).

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. 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 c. (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 // Programmirovanie. 1980. N 6. S. 27–36. (In Russian)

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. Kudryavtsev A. A., Malyshkin 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 базисах[1] 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 рассмотрения 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.

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

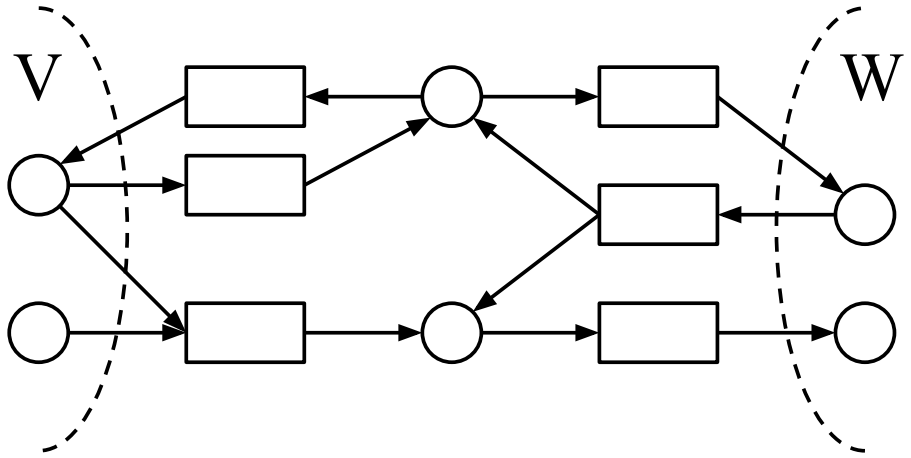

Рис. 1. Пример вычислительной модели. Круги — переменные, прямоугольники — операции

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: $\langle y_1, y_2, ..., y_n \rangle = f(x_1, x_2, ..., x_m)$, где $y_1, y_2, ..., y_n$ — это значения выходных переменных операции, $x_1, x_2, ..., x_m$ — это значения выходных 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 $f_a(t_{x_1},...,t_{x_n})$, где y — это переменная, вычисляемая некоторой операцией $a \in T$ из переменных $x_1,...,x_n$, f_a — это функциональный символ, обозначающий функцию $I(a)$, а t_{x_j} — это терм, вычисляющий переменную $x_j, j = 1...n$, который либо является предметным символом, если $x_j \in V$, либо некоторым другим термом вида $f_b(t_{z_1},...,t_{z_m})$, если переменная x_j вычисляется некоторой операцией $b \in T$ и имеет входные переменные $z_1,...,z_m$.

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-перечислимым 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 нефункциональные[2] 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-

[2]Тут 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 мультикомпьютера).

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. Исключение 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

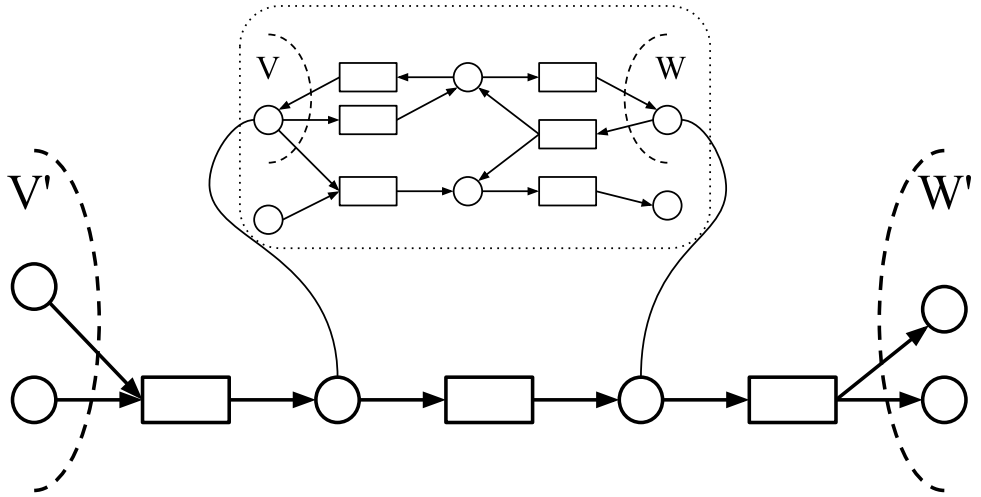

Рис. 2. Пример структурированной операции. Входные и выходные переменные структурированной операции отождествляются с переменными вложенной вычислительной модели, образуя множества V и W соответственно для VW-задачи на вложенной модели

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 исполняется 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 мультикомпьютера 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: ВИНИТИ, 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., Спирин 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 настоящее 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 Malyshkin 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: perepelkin@ssd.sscc.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 экспериментальной 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

STOCHASTIC PETRI NETS SOFTWARE TOOLS

A. V. Bystrov, I. B. Virbitskaite, E. S. Oshevskaya

A. P. Ershov Institute of Informatics Systems,
630090, Novosibirsk, Russia

DOI: 10.24412/2073-0667-2024-2-32-57
EDN: KNBRYZ

The behavior of a wide variety of systems, biochemical, transport, industrial, software, and so on, is inherently parallel, non-deterministic, and stochastic. The study and design of these systems requires the use of models that take into account all these aspects, as well as appropriate software tools. Stochastic Petri nets and their various extensions are successfully used as such models. They combine the clarity and intuitiveness of the graphical representation with well-developed mathematical and algorithmic apparatus of analysis. These models allow us to study not only qualitative but also quantitative properties of systems, such as bandwidth, reliability, waiting time, etc. Software tools that support the construction, modification and analysis of system models based on various variants of stochastic Petri nets have already been developed and continue to appear.

This paper provides a detailed overview of several such multiplatfom software tools, namely, GreatSPN, ORIS, PetriNuts, TimeNet and PIPE2 that are available on the Internet, and got recognized by users. The introduction, informally, but with proper references to the literature, gives the basic concepts, defines the classes of Petri nets and terms used later. Then, for each of the software tools, its structure, features and peculiarities are considered. The tools are then compared in terms of their functional and performance analysis capabilities, and recommendations to users on how to use the tools depending on what type of stochastic models need to be investigated are discussed. The main purpose of the paper is to facilitate the researcher and engineer in selecting the most appropriate modeling and analysis tool for the task at hand.

Key words: stochastic Petri nets, modelling, simulation, performance analysis, Petri net tools.

References

1. Reisig W. Petri Nets: An Introduction. V. 4. Springer, 1985. (EATCS Monographs on Theoretical Computer Science), DOI: 10.1007/978-3-642-69968-9.

2. Boyer M., Roux O. On the Compared Expressiveness of Arc, Place and Transition Time Petri Nets // Fundamenta Informaticae. 2008. Jan. V. 88. P. 225–249.

3. Berthomieu B., Diaz M. Modeling and verification of time dependent systems using time Petri nets // IEEE Transactions on Software Engineering. 1991. Mar. V. 17, N 3. P. 259–273. DOI: 10.1109/32.75415.

4. Molloy M. Performance Analysis Using Stochastic Petri Nets // IEEE Trans. Computers. 1982. V. 31, N 9. P. 913–917. DOI: 10.1109/TC.1982.1676110.

5. Vicario E., Sassoli L., Carnevali L. Using Stochastic State Classes in Quantitative Evaluation of Dense-Time Reactive Systems // IEEE Trans. Software Eng. 2009. V. 35, N 5. P. 703–719. DOI: 10.1109/TSE.2009.36.

6. Wang J. Stochastic Timed Petri Nets and Stochastic Petri Nets // Timed Petri Nets: Theory and Application. Boston, MA : Springer US, 1998. P. 125–153. DOI: 10.1007/978-l-4615-5537-7_5.

7. Ajmone Marsan M. et al. An introduction to generalized stochastic Petri nets // Microelectronics Reliability. 1991. Jan. V. 31, N 4. P. 699–725. DOI: 10.1016/0026-2714(91)90010-5.

8. Ajmone Marsan M., Chiola G. On Petri nets with deterministic and exponentially distributed Bring times // Advances in Petri Nets 1987, covers the 7th European Workshop on Applications and Theory of Petri Nets, Oxford, UK, June 1986. V. 266 / ed. by G. Rozenberg. Springer, 1986. P. 132–145. (Lecture Notes in Computer Science), DOI: 10.1007/3-540-18086-9_23.

9. Dugan J. et al. Extended Stochastic Petri Nets: Applications and Analysis // Performance '84, Proceedings of the Tenth International Symposium on Computer Performance Modelling, Measurement and Evaluation / ed. by E. Gelenbe. North-Holland, 1984. P. 507–519.

10. Ajmone Marsan M. et al. The Effect of Execution Policies on the Semantics and Analysis of Stochastic Petri Nets // IEEE Trans. Software Eng. 1989. V. 15, N 7. P. 832–846. DOI: 10.1109/32.29483.

11. German R., Lindemann C. Analysis of stochastic Petri nets by the method of supplementary variables // Performance Evaluation. 1994. May. V. 20, N 1–3. P. 317–335. DOI: 10.1016/0166-5316(94)90020-5.

12. Stoxasticheskie seti Petri — formalizm dlya modelirovaniya i analiza proizvoditcl'nosti vychislitel'nyx processov // Sistemnaya Informatika. Novosobirsk, 2004. S. 135–193 (In Russian).

13. German R. Performance analysis of communication systems — modelling with non- Markovian stochastic Petri nets : Modeling with Non-Markovian Stochastic Petri Nets. Wiley, 2000. P. 456.

14. Biagi M. et al. Exploiting Non-deterministic Analysis in the Integration of Transient Solution Techniques for Markov Regenerative Processes // Quantitative Evaluation of Systems. Springer International Publishing, 2017. P. 20–35. DOI: 10.1007/978-3-319-66335-7_2.

15. Martina S. et al. Performance Evaluation of Fischer's Protocol through SteadyState Analysis of Markov Regenerative Processes // IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). 09/2016. P. 355–360. DOI: 10.1109/MASCOTS.2016.72.

16. Horvath A. et al. Transient analysis of non-Markovian models using stochastic state classes // Performance Evaluation. 2012. V. 69, N 7/8. P. 315–335. DOI: 10.1016/j.peva.2011.11.002.

17. Amparore E. Stochastic Modelling and Evaluation Using GreatSPN // ACM- SIGMETRICS Performance Evaluation Review. New York, NY, USA, 2022. June. V. 49, N 4. P. 87–91. DOI: 10.1145/3543146.3543165.

18. Amparore E. et al. Years of GreatSPN // Principles of Performance and Reliability Modeling and Evaluation: Essays in Honor of Kishor Trivedi on his 70th Birthday / ed. by L. Fiondella, A. Puliafito. Cham : Springer International Publishing, 2016. P. 227–254. DOI: 10.1007/978-3-319-30599-8-9.

19. K. J., Kristensen L. Coloured Petri Nets. Springer Berlin Heidelberg, 2009. DOI: 10.1007/b95112.

20. ISO/IEC. Software and Systems Engineering - High-level Petri Nets, Part 2: Transfer Format, International Standard ISO/IEC 15909, February 2011.

21. Kindler E. The Petri Net Markup Language and ISO/IEC 15909-2: Concepts, Status, and Future Directions // Tagungsband Entwurf komplcxcr Automatisierungssysteme EKA. 2006. P. 35–55.

22. Clarke E., Emerson E. Design and synthesis of synchronization skeletons using branching time temporal logic // Logics of Programs / ed. by D. Kozen. Springer Berlin Heidelberg, 1982. P. 52–71.

23. Deharbe D. A Tutorial Introduction to Symbolic Model Checking // Logic for Concurrency and Synchronisation / ed. by R. de Queiroz. Dordrecht : Springer Netherlands, 2003. P. 215–237. DOI: 10.1007/0-306-48088-3_5.

24. Beccuti M., Franceschinis G., Haddad S. Markov Decision Petri Net and Markov Decision Well-Formed Net Formalisms // Petri Nets and Other Models of Concurrency — ICATPN 2007 / ed. by J. Kleijn, A. Yakovlev. Springer Berlin Heidelberg, 2007. P. 43–62. DOI: 10.1007/978-3-540-73094-1_6.

25. Emerson E., Sistla A. Symmetry and model checking // Formal Methods in System Design. 1996. V. 9, N 1. P. 105–131. DOI: 10.1007/BF00625970.

26. Babar J. et al. GrcatSPN Enhanced with Decision Diagram Data Structures // Applications and Theory of Petri Nets / ed. by J. Lilius, W. Penczek. Springer Berlin Heidelberg, 2010. P. 308–317.

27. Chaki S., Gurfinkel A. BDD-Based Symbolic Model Checking // Handbook of Model Checking / ed. by E. Clarke et al. Springer, 2018. P. 219–245. DOI: 10.1007/978-3-319-10575-8_8.

28. R. R., S. B., Zimmermann A. An Evaluation Framework for Comparative Analysis of Generalized Stochastic Petri Net Simulation Techniques // IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2020. V. 50. P. 2834–2844. DOI: 10.1109/TSMC.2018.2837643.

29. Pernice S. et al. Multiple Sclerosis Disease: A Computational Approach for Investigating Its Drug Interactions // Computational Intelligence Methods for Bioinformatics and Biostatistics / ed. by P. Cazzaniga et al. Cham : Springer International Publishing, 2020. P. 299–308. DOI: 10.1007/978-3-030-63061-4_26.

30. Amparore E., Donatelli S., Landini E. Modelling and Evaluation of a Control Room Application // Application and Theory of Petri Nets and Concurrency / ed. by W. van der Aalst, E. Best. Cham: Springer International Publishing, 2017. P. 243–263.

31. Richard L. Performance Results for the CSMA/CD Protocol Using GrcatSPN // Journal of Systems and Software. 1997. V. 37, N 1. P. 75–90. DOI: 10.1016/S0164-1212(96)00041-6.

32. Amparore E., Donatelli S. GreatTeach: A Tool for Teaching (Stochastic) Petri Nets // Application nd Theory of Petri Nets and Concurrency. Springer International Publishing, 2018. P. 416–425. DOI: 10.1007/978-3-319-91268-4_24.

33. Castagno P. et al. A computational framework for modeling and studying pertussis epidemiology and vaccination // BMC Bioinformatics. 2020. V. 21, N 8. P. 344. DOI: 10.1186/S12859-020-03648-6.

34. The GrcatSPN Framework. [El. Res.]: `https://github.com/greatspn/SOURCES`. Accessed: 2024-03-15.

35. Paolieri M. et al. The ORIS Tool: Quantitative Evaluation of Non-Markovian Systems // IEEE Trans. Software Eng. 2021. V. 47, N 6. P. 1211–1225. DOI: 10.1109/TSE.2019.2917202.

36. Carnevali L., Paolieri M., Vicario E. The ORIS tool: app, library, and toolkit for quantitative evaluation of non-Markovian systems // ACM SIGMETRICS Performance Evaluation Review. 2022. V. 49, N 4. P. 81–86. DOI: 10.1145/3543146.3543164.

37. Stewart W. Introduction to the Numerical Solution of Markov Chains. Princeton University Press, 1995. DOI: 10.1515/9780691223384.

38. Carnevali L. et al. Non-Markovian Performability Evaluation of ERTMS/ETCS Level 3 // Computer Performance Engineering - 12th European Workshop, EPEW 2015. V. 9272 / ed. by M. Beltran, W. Knottenbelt, J. Bradley. Cham : Springer, 2015. P. 47–62. (Lecture Notes in Computer Science), DOI: 10.1007/978-3-319-23267-6_4.

39. Biagi M. et al. Model-Based Quantitative Evaluation of Repair Procedures in Gas Distribution Networks // ACM Trans. Cyber Phys. Syst. 2019. V. 3, N 2. 19:1–19:26. DOI: 10.1145/3284037.

40. Carnevali L., Tarani F., Vicario F. Performability Evaluation of Water Distribution Systems During Maintenance Procedures // IEEE Trans. Syst. Man Cybern. Syst. 2020. V. 50, N 5. P. 1704–1720. DOI: 10.1109/TSMC.2017.2783188.

41. Carnevali L. et al. Using the ORIS Tool and the SIRIO Library for Model- Driven Engineering of Quantitative Analytics // Computer Performance Engineering / ed. byK. Gilly, N. Thomas. Cham: Springer International Publishing, 2023. P. 200–215. DOI: 10.1007/978-3-031-25049-1_13.

42. ORIS Tool. [El. Res.]: `http://www.oris-tool.org`. Accessed: 2024-03-15.

43. ORIS Tool: The Sirio Library. [El. Res.]: `https://github.com/oris-tool/sirio`. Accessed: 2024-03-15.

44. Heiner M. et al. Snoopy — A Unifying Petri Net Tool // Application and Theory of Petri Nets. PETRI NETS 2012. V. 7347 / ed. by S. Haddad, L. Pomello. Springer Berlin Heidelberg, 2012. P. 398–407. (Lecture Notes in Computer Science), DOI: 10.1007/978-3-642-31131-4_22.

45. David R., Alla H. Discrete, Continuous, and Hybrid Petri Nets. Springer Berlin Heidelberg, 2010. P. 550. DOI: 10.1007/978-3-642-10669-9.

46. Liu F., Heiner M., Gilbert D. Fuzzy Petri nets for modelling of uncertain biological systems // Briefings in Bioinformatics. 2018. Dec. V. 21, N 1. P. 198–210. DOI: 10.1093/bib/bbyll8.

47. Fujita M., McGeer P., Yang J. Multi-Terminal Binary Decision Diagrams: An Efficient DataStructure for Matrix Representation // Form. Methods Syst. Des. USA, 1997. Apr. V. 10, N 2/3. P. 149–169. DOI: 10.1023/A: 1008647823331.

48. Hucka M. et al. Systems Biology Markup Language (SBML) Level 2 Version 5: Structures and Facilities for Model Definitions // Journal of Integrative Bioinformatics. 2015. V. 12. N 2. P. 731–901. DOI: 10.2390/biecoll-jib-2015-271.

49. Heiner M., Schwarick M., Wegener J.-T. Charlie — An Extensible Petri Net Analysis Tool // Application and Theory of Petri Nets and Concurrency / ed. by R. Devillers, A. Valmari. Cham : Springer International Publishing, 2015. P. 200–211. DOI: 10.1007/978-3-319-19488-2_10.

50. Heiner M., Rohr C., Schwarick M. MARGIE — Model Checking and Reachability Analysis Done Efficiently // Application and Theory of Petri Nets and Concurrency / ed. by J.-M. Colom, J. Desel. Springer Berlin Heidelberg, 2013. P. 389–399. DOI: 10.1007/978-3-642-38697-8_21.

51. Baier C. et al. Model Checking Continuous-Time Markov Chains by Transient Analysis // Computer Aided Verification / ed. by E. Emerson, A. Sistla. Springer Berlin Heidelberg, 2000. P. 358–372.

52. Donaldson R., Gilbert D. A Model Checking Approach to the Parameter Estimation of Biochemical Pathways // Computational Methods in Systems Biology / ed. by M. Heiner, A. M. Uhrmacher. Springer Berlin Heidelberg, 2008. P. 269–287.

53. Chodak J., Heiner M. Spike — Reproducible Simulation Experiments with Configuration File Branching // Computational Methods in Systems Biology. Springer International Publishing, 2019. P. 315–321. DOI: 10.1007/978-3-030-31304-3_19.

54. Gilbert D., Donaldson R. A Monte Carlo model checker for probabilistic LTL with numerical constraints : tech. reP. / Bioinformatics Research Centre, University of Glasgow. 01/2008.

55. Gilbert D. et al. Spatial quorum sensing modelling using coloured hybrid Petri nets and simulative model checking // BMC Bioinformatics. 2019. V. 20, supplement 4. DOI: 10.1186/sl2859-019-2690-z.

56. Hcrajy M. et al. Snoopy's hybrid simulator: a tool to construct and simulate hybrid biological models // BMC Systems Biology. 2017. July. V. 11, N 1. DOI: 10.1186/S12918-017-0449-6.

57. Zimmermann A. Modelling and Performance Evaluation with TimeNET 4.4 // Quantitative Evaluation of Systems - 14th International Conference, QEST 2017. V. 10503 / ed. by N. Bertrand, L. Bortolussi. Springer, 2017. P. 300–303. (Lecture Notes in Computer Science), DOI: 10.1007/978-3-319-66335-7_19.

58. Selic B. Modeling And Analysis Of Realtime And Embedded Systems With Uml And Marte Developing Cyberphysical Systems. Elsevier Science & Technology, 2014. DOI: 10.1016/C2012-0-13536-5.

59. Zimmermann A. et al. Analysis of Safety-Critical Cloud Architectures with MultiTrajectory Simulation // Annual Reliability and Maintainability Symposium (RAMS). 01/2022. P. 1–7. DOI: 10.1109/RAMS51457.2022.9893923.

60. Fedorova A., Beliautsou V., Zimmermann A. Colored Petri Net Modelling and Evaluation of Drone Inspection Methods for Distribution Networks // Sensors. 2022. V. 22, N 9. DOI: 10.3390/s22093418.

61. Dingle N., Knottenbelt W., Suto T. PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets // SIGMETRICS Performance Evaluation Review ACM. New York, NY, USA, 2009. Mar. V. 36, N 4. P. 34–39. DOI: 10.1145/1530873.1530881.

62. Platform Independent Petri Net Editor v4. [El. Res.]: `https://sourceforge.net/projects/pipe2`. Accessed: 2024-03-15.

63. PIPE 5. [El. Res.]: `https://github.com/sarahtattersall/PIPE`. Accessed: 2024-03-15.

ИНСТРУМЕНТАЛЬНЫЕ СИСТЕМЫ, ПОДДЕРЖИВАЮЩИЕ СТОХАСТИЧЕСКИЕ СЕТЕВЫЕ МОДЕЛИ

А. В. Быстров, И. Б. Вирбицкайте, Е. С. Ошевская
Институт систем информатики им. А. П. Ершова,
630090, Новосибирск, Россия

УДК 004.94+519.876.5
DOI: 10.24412/2073-0667-2024-2-32-57
EDN: KNBRYZ

Стохастические сети Петри — мощное средство моделирования параллельных недетерминированных систем с вероятностными характеристиками, применяемое в самых разных областях человеческой деятельности. Они сочетают наглядность графического представления с развитым математическим и алгоритмическим аппаратом анализа, позволяют изучать не только качественные, но и количественные свойства систем, такие как пропускная способность, надежность, время ожидания и т. п. Разработаны и продолжают появляться новые программные инструменты, поддерживающие создание, модификацию и анализ свойств моделей систем на основе различных вариантов стохастических сетей Петри. В данной работе рассматриваются несколько таких инструментов, доступных в сети Интернет и получивших признание пользователей, обсуждаются предоставляемые ими возможности и проводится их сравнение. Основная цель работы — облегчить исследователю и инженеру выбор наиболее подходящего инструмента моделирования и анализа для решения стоящей перед ним задачи.

Ключевые слова: стохастические сети Петри, моделирование, анализ производительности, инструментальные системы.

Введение. Сети Петри (СП) [1] — одна из наиболее популярных моделей, используемых как для теоретических исследований структуры и поведения параллельных недетерминированных систем, так и для практических применений в различных областях автоматизированной обработки информации: распределенных базах данных и операционных системах, архитектурах вычислительных машин, систем и сетей, системах программного обеспечения, протоколах коммуникаций, системах с элементами искусственного интеллекта, производственных системах, системах экологии и микробиологии, сельскохозяйственном производстве и др. Структура СП состоит из *переходов*, моделирующих действия системы; *мест*, представляющих память действий; *направленных дуг* от входных мест к переходам и от переходов к выходным местам. *Разметка* СП, моделирующая состояние поведения СП, присваивает каждому месту натуральное количество *фишек*. Переход *разрешен (к срабатыванию) при разметке*, если каждое его входное место содержит достаточное количество фишек; достаточность определяется числом дуг из входного места в переход. При *срабатывании* переход удаляет из каждого входного места столько фишек, сколько дуг идет из места в переход, и добавляет в каждое выходное место столько фишек, сколько дуг идет в место из перехода. Таким образом, срабатывания переходов приводят к смене разметок. При функционировании СП порождается *последовательность срабатываний переходов*. К достоинствам СП относятся наглядное графическое представление

их структуры и эффективные методы анализа их поведения. Однако сети Петри не могут быть использованы для изучения количественных (например, временных) параметров функционирования моделируемых систем, таких как время отклика системы или ее производительность.

Особое место среди моделей с реальным временем занимают *временные сети Петри* (ВСП). В теории сетей Петри известны разнообразные временные расширения, в которых временные характеристики сопоставляются различным элементам (местам, переходам, дугам, фишками) сетевых моделей [2]. Наибольшую популярность получили расширения сетей Петри, где с переходами связываются временные задержки, а модельные часы измеряют ход времени в состояниях сети. В литературе рассматриваются два подхода — *дискретно-временные* СП (ДВСП) и *непрерывно-временные* СП (НВСП). В ДВСП имеются глобальные часы и каждому переходу сопоставляется целочисленное значение — длительность срабатывания перехода, тогда как в НВСП с каждым переходом связываются локальные часы и интервал действительных чисел — временных задержек срабатывания перехода (при этом срабатывание перехода считается мгновенным). Известно, что НВСП равномощны машинам Тьюринга и включают класс ДВСП, поэтому именно модели НВСП вызывают наибольший интерес у исследователей. НВСП имеют следующее поведение. *Состояние* НВСП содержит текущую разметку (распределение фишек по местам сети) и временной вектор показаний часов разрешенных переходов (т. е. переходов, все входные места которых имеют достаточное количество фишек при данной разметке). Переход *разрешен (к срабатыванию) в состоянии* только в том случае, если переход разрешен при этой разметке и его часы достигли момента времени, который находится в пределах временного интервала данного перехода. При функционировании НВСП порождаются *последовательности срабатываний переходов с временными характеристиками*, приводящие к достижимым состояниям. Пространство состояний НВСП, представляемое графом достижимых состояний (графом достижимости), бесконечно и недискретно, что увеличивает сложность анализа модели, поэтому используется метод классов состояний [3]. При функционировании НВСП последовательность срабатываний переходов может запускаться с различными временными характеристиками и при этом достигать одной и той же разметки, но, возможно, с разными временными векторами, содержащими показания часов разрешенных переходов. Множество временных векторов со значениями, попадающими в интервалы разрешенных переходов, принимает форму выпуклого линейного многогранника, обычно называемого *зоной*. *Класс состояний* — это композиция разметки и зоны. *Граф классов состояний (абстракция графа достижимости)* представляет собой символьное представление пространства состояний НВСП, где все часы разрешенных переходов продвигаются с одинаковой скоростью. ВСП пригодны для спецификации и анализа систем реального времени, таких как управление процессами, производственные системы, роботы, авиационные системы.

Другое временное расширение СП — *стохастические сети Петри* (ССП) [4], основанные на концепции стохастических временных задержек. В ССП предполагается, что все переходы запускаются с временными задержками — случайными величинами с известными законами распределения. Часто в ССП используется экспоненциальный закон распределения времени срабатывания переходов, что упрощает анализ таких ССП. В ССП с переходами, имеющими неэкспоненциальное распределение задержек, как правило, накладываются существенные ограничения на структуру сетей с целью облегчения их анализа. В настоящее время известно несколько подходов к определению ССП. Например,

разработаны ССП с непрерывным временем (временная шкала — неотрицательные действительные числа) [5], обозначаемые НВССП, и дискретным временем (временная шкала — натуральные числа) [6], обозначаемые ДВССП. В *обобщенных ССП* (ОССП) [7] имеются экспоненциальные (с экспоненциально распределенными задержками) и мгновенные (с нулевыми задержками) переходы. *Детерминированные ССП* (ДССП) [8] — это модель с экспоненциальными и детерминированными (с фиксированной задержкой) переходами. *Детерминированные обобщенные ССП* (ДОССП) имеют экспоненциальные, мгновенные и детерминированные переходы. В *расширенных ССП* (РССП) [9–10] возможны переходы с неэкспоненциальными задержками. *Расширенные ДССП (РДССП)* [11] могут иметь детерминированные переходы с неэкспоненциальными задержками. Использование случайных временных величин обогащает анализ систем путем вычисления количественных показателей их эффективности (производительности) и надежности, что позволяет на ранней стадии оценить выбор метода проектирования и предоставляет рекомендации для этапов внедрения и интеграции проектируемых систем. ССП лучше подходят в качестве моделей систем с разделением ресурсов, архитектур «клиент-сервер», коммуникационных систем, производственных линий и т. д. В стохастических сетевых моделях нередко присутствуют так называемые *«вознаграждения»* (rewards) — функции, которые сопоставляют численные значения состояниям или переходам и которые используются для получения таких характеристик моделируемой системы, как потребление ресурсов, доступность, надежность и др. «Вознаграждения» могут также суммироваться в ходе функционирования модели.

Детальное изложение математического аппарата, примеры и обсуждение различных классов стохастических сетей Петри читатель может найти в работе [12]. Мы также используем аббревиатуры классов, введенные в этой работе.

Анализируют ССП обычно с помощью *марковских процессов* (МП), называемых *марковскими цепями* (МЦ) с дискретным временем (ДВМЦ) для ДВССП и с непрерывным временем (НВМЦ) для НВССП. Граф достижимости ССП может быть непосредственно сопоставлен с МП, который удовлетворяет Марковскому свойству (говоря нестрого, при фиксированном настоящем будущее независимо от прошлого). Каждое состояние в графе достижимости сопоставляется с состоянием в марковском процессе, а срабатывание перехода с некоторой временной задержкой — с переходом в марковское состояние с соответствующей вероятностью. МЦ имеют множество применений в качестве статистических моделей реальных процессов, например, при изучении систем круиз-контроля в автомобилях, очередей клиентов, курсов обмена валют, динамики популяции животных и т. д. Для анализа НВМЦ, как правило, используют численные подходы. Когда в расширенных ССП (РССП) имеются переходы с неэкспоненциальными задержками, лежащий в основе модели стохастический процесс относится к более сложным классам — немарковским процессам [13], но численное решение все еще может быть получено, если модель удовлетворяет Марковскому свойству в определенные моменты времени, называемые точками *регенерации*. В частности, если новая регенерация всегда достигается с вероятностью 1, то модель поддерживает марковский регенерирующий процесс (МРП), который может быть решен численно. Другими словами, МРП — это стохастический процесс, который рано или поздно обязательно достигнет регенерации.

Стохастический процесс имеет как стационарное (устойчивое) состояние, так и переходное. По истечении определенного времени в процессе устанавливается *стационарное состояние*, т. е. когда текущее поведение системы сохранится и в будущем. *Переходное состояние* — это, по сути, время между началом некоторого события (например, сраба-

тыванием переходов) и стационарным состоянием. *Стационарный анализ* стохастического процесса используется для определения стационарных состояний. При *переходном анализе* процесса осуществляется исследование вероятностей нахождения в определенном состоянии в определенный момент времени. Переходный анализ процессов занимает значительно больше времени и требует больших вычислительных мощностей для получения результатов. Регенеративный переходный/стационарный анализ [14–15] — это переходный/стационарный анализ, выполненный в рамках МРП. Для анализа стохастических процессов часто используется симуляция их поведения. При *стационарной симуляции* процессов вычисляется решение, которое не меняется со временем, тогда как при *переходной симуляции* — мгновенные значения в каждый момент времени для каждой случайной величины. Стационарная симуляция происходит намного быстрее, чем переходная. Кроме того, для анализа ССП применяется метод *классов стохастических состояний* [5, 16], объединяющий проверку корректности возможных вариантов поведения (идентифицируемых базовой НВСП) с количественной оценкой их вероятности (индуцируемой стохастическими параметрами НВССП).

Очевидно, что при разработке прикладных проектов необходимо использовать программный инструментарий. Инструменты, поддерживающие ССП, можно использовать для анализа производительности стохастических систем, а также для генерации кода, симуляции, анализа и верификации разных классов ССП. На сегодняшний день существует ряд таких инструментов, разработанных для различных целей и сред (платформ). Однако имеет смысл рассматривать только те инструментальные системы, которые обладают следующими свойствами:

— поддерживают стохастические модели сетей Петри и их обобщения;

— включают графический редактор моделей, симулятор, аниматор, анализатор и верификатор;

— требуют разумные аппаратные и общедоступные программные ресурсы;

— поддерживают мультиплатформенность;

— достаточно хорошо документированы;

— бесплатны, по крайней мере, для академических исследований.

В данной работе дается детальный обзор пяти выдержавших проверку временем, свободно распространяемых и ориентированных на различные платформы инструментальных систем (а именно, GreatSPN, ORIS, PetriNuts, TimeNet, PIPE2), поддерживающих широкий спектр разновидностей стохастических моделей и обеспечивающих их оценку производительности с помощью эффективных методов решения, включая симуляцию стохастических процессов. Рассматриваемые инструменты сравниваются с точки зрения их возможностей функционального и временного анализа моделей, а также обсуждаются рекомендации пользователям по использованию инструментов в зависимости от того, какой тип стохастических моделей требуется исследовать и с какими операционными системами пользователи работают.

1. GreatSPN. GreatSPN [17–18] (изначально разрабатывался в университете г. Торонто (Италия), в настоящее время также поддерживается в университете г. Алессандрии (Италия)) представляет собой многофункциональный инструментальный программный комплекс на базе различных моделей стохастических сетей Петри, предназначенный для спецификации, анализа, валидации и оценки производительности параллельных/распределенных систем, функционирующих в режиме реального времени.

В GreatSPN версии 3.0 поддерживаются следующие стохастические модели:

— обобщенные стохастические сети Петри (ОССП), базовые сети которых включают приоритеты переходов и ингибиторные дуги и которые являются удобным формализмом для оценки производительности больших систем благодаря возможности их автоматического преобразования в непрерывно-временную Марковскую цепь (НВМЦ);

— ОССП с цветными фишками (ОЦССП) [19], которые расширяют возможности моделирования многокомпонентных систем;

— ОССП с детерминированными переходами (ДОССП) и с неэкспоненциальными переходами (РОССП).

Модели могут быть модульными, т. е. состоять из отдельных независимых сетей, которые объединяются с помощью алгебраических операторов композиции. Цель анализа поддерживаемых стохастических моделей состоит в том, чтобы проверить их правильность с помощью структурного/поведенческого анализа и проверки модели (model checking). Как только пользователь уверен, что модель отражает поведение проектируемой системы, процесс анализа концентрируется на вероятностных аспектах посредством стохастической проверки модели и вычисления классических показателей оценки производительности и надежности.

Функциональные возможности системы GreatSPN включают: структурный анализ, исследование пространства состояний, преобразования между поддерживаемыми формализмами, верификацию, поддержку принятия решения на основе Марковских цепей, численные решения, симуляцию методом Монте-Карло.

Система GreatSPN представляет собой набор инструментов, которые могут быть организованы по их функциям в следующие категории:

— унифицированный графический интерфейс, реализованный на Java;

— инструменты трансляции, композиции, развертки моделей для работы с несколькими форматами;

— структурный анализатор свойств моделей;

— генератор и оптимизатор Марковских процессов (цепей) для анализа стохастических моделей;

— проверка моделей (model checker) на базе темпоральных логик;

— стохастический решатель.

Рассмотрим подробнее возможности и особенности каждого инструмента.

Графический редактор является центральным модулем системы GreatSPN, поскольку он используется для построения моделей в графическом представлении и определения их свойств, а также для интерактивной симуляции с анимацией «игры с (цветными) фишками» модели. Он отвечает также за вызов командной строкой всех основных модулей (моделирования, верификации, анализа, симуляции и т. д.) и за визуализацию результатов их работы. С целью более удобного обмена моделями с другими инструментами система GreatSPN поддерживает стандартный формат PNML [20–21]. К сожалению, в действующей версии PNML еще нет надлежащего способа представления стохастических атрибутов элементов сети, и при экспорте модели в формате PNML такие атрибуты опускаются. Основным форматом ввода остается нестандартный, но широко поддерживаемый формат net/def, при этом доступны и другие форматы (APNN, GrML, NetLogo). Возможна трансляция из PNML в net/def и из UML в сети Петри, развертка ОЦССП в ОССП, а также композиция сетевых моделей.

Структурный анализатор включает модули:

— вычисления P/T-инвариантов/потоков;

— вычисления структурной ограниченности и неограниченных последовательностей переходов;

— вычисления структурных конфликтных множеств, тупиков, ловушек и взаимоблокировок;

— анализ сетевых свойств, представленных формулами темпоральных логик LTL, CTL, CTL^* [22].

Модуль проверки моделей (model checker) включает модуль символьной проверки моделей (symbolic model checker) [23] на базе темпоральной логики CTL^* и модуль стохастической проверки моделей (stochastic model checker) на базе непрерывно-временной стохастической темпоральной логики CSL^{TA} для временных автоматов, в которой вычисленные показатели производительности выражаются с использованием детерминированных временных автоматов. Возможно графическое представление таких автоматов в интерфейсе системы GreatSPN, а затем их использование для вычисления показателей производительности моделей.

Генератор и оптимизатор Марковских процессов (цепей) осуществляет их построение из высокоуровнего языка Markov Decision Petri Nets [24], предназначенного для спецификации вероятностного и недетерминированного поведения моделируемой системы и синтаксического способа определения переключения между этими двумя типами поведения, а также вычисляет решение Марковских процессов с использованием модулей как переходного, так и стационарного численного анализа для получения показателей производительности из Марковских цепей.

Стохастический решатель включает в себя набор инструментов для проведения анализа с использованием графов достижимости, симуляции и дифференциальных уравнений.

Численные решения для переходных и стационарных процессов используют построение графов достижимости сетевых моделей. Существует несколько разновидностей графов достижимости, которые можно строить в системе:

— граф достижимости ОССП-моделей;

— граф достижимости ОЦССП-моделей;

— граф достижимости символьных разметок (объединяющих разметки, при которых разрешены одни и те же переходы) в ОЦССП, использующий симметрии моделей [25];

— граф достижимости марковских регенерирующих процессов (МРП) стохастических моделей с возможностью одновременных срабатываний переходов;

— граф достижимости, закодированный с применением диаграмм принятия решений (Binary Decision Diagrams) [26–27].

Решатель для ДОССП использует расширенный формализм сетевой модели, который включает параметры разметки, параметры скорости переходов, особенности срабатывания переходов (например, задержка перехода может определяться функцией, зависящей от разметки), общее, возможно неэкспоненциальное, время задержки срабатывания переходов и предохранители переходов (guards). Этот решатель может обрабатывать как НВМЦ в переходном и стационарном режимах, так и МРП только в стационарном режиме и использует передовые численные методы.

Симулятор Монте-Карло позволяет вычислять показатели производительности с использованием пакетной симуляции в переходном или стационарном режиме. Для ОЦССП симуляция может быть выполнена либо с использованием обычных разметок и переходов, либо с использованием символьных разметок. Симулятор поддерживает стохастиче-

ские модели с произвольным числом переходов с различными распределениями задержек срабатывания переходов (равномерное, Эрланга, Кокса и др.). Сравнительный анализ возможностей симулятора GreatSPN можно найти в статье [28], где показана высокая точность и производительность алгоритмов анализа, используемых симулятором. Инструмент ориентируется на вычисление целевых показателей производительности и выполняет итерации по созданным событиям до тех пор, пока все показатели не окажутся ниже порогового значения точности.

Сгенерированные показатели производительности выражаются с использованием языка, специфичного для предметной области, могут быть экспортированы в форматах CSV, Excel, PDF, PNG для обработки с помощью внешних инструментов.

Предусмотрены два основных типа взаимодействия GreatSPN с другими инструментами:

— экспорт моделей из GreatSPN в принимающий инструмент, чтобы применять методы анализа, доступные в этом инструменте;

— повторное использование модулей GreatSPN (или их части) другими инструментами.

Инфраструктура GreatSPN3.0 использовалась в таких контекстах как биологическое моделирование [29], проектирование диспетчерского центра в газовой компании [30], оценивание эффективности сетевых протоколов [31] и многих других. В ходе разработки GreateSPN особое внимание уделялось удобству использования этого инструмента при обучении работе со стохастическими моделями [32] и для изучения биологических систем [33].

Дистрибутив системы GreatSPN включает также дополнительные утилиты:

— *algebra* предназначена для композиции моделей и визуализации графов достижимости;

— *multisolve* предназначена для серийного запуска анализатора с варьируемыми параметрами модели.

GreatSPN имеет открытый исходный код, доступный [34] на GitHub, и работает на всех основных платформах (Linux, Windows, macOS).

2. ORIS. ORIS [35–36] (разрабатывался и поддерживается Университетом г. Флоренции (Италия) и Университетом Южной Калифорнии г. Лос Анжелеса (США) — это программный инструмент для моделирования и анализа производительности и надежности стохастических систем, управляемых немарковскими таймерами (например, временем между прибытием и/или обслуживанием, временем отказа, временем ремонта, колеблющимися задержками, тайм-аутами), задаваемыми с использованием различных функций плотности вероятности.

В целом, ORIS представляет собой хорошо спроектированный программный инструмент, полностью реализованный на Java и разработанный таким образом, чтобы облегчить его расширение новыми средствами работы с непрерывно-временными стохастическими сетями Петри (НВССП) и реализацию новых методов решения, лежащих в их основе стохастических процессов.

В качестве графического формализма для моделирования стохастических систем ORIS использует НВССП, в базовых НВСП которых с каждым переходом могут быть связаны:

— функция, контролирующая количество фишек во входных местах, необходимое для срабатывания перехода;

— функция, определяющая дополнительное количество фишек в выходных местах после срабатывания перехода;

— множество переходов, для которых принудительно переустанавливается время с целью их срабатывания;

— приоритет и вес (при конкуренции на срабатывания выбираются переходы с наивысшим приоритетом и с вероятностью, пропорциональной их весам);

— временные интервалы срабатывания переходов (т. е. возможно задание как непрерывно-временных задержек, так и нулевых).

В НВССП функции плотности вероятности, связанные с переходами, могут быть немедленными, детерминированными, экспоненциальными и неэкспоненциальными. Следует подчеркнуть, что ORIS поддерживает НВССП, в каждом состоянии которых несколько таймеров могут быть запущены одновременно.

ORIS предоставляет как графический пользовательский интерфейс (GUI), так и интерфейс прикладного программирования (Java API), основанный на встроенной Java-библиотеке SIRIO.

Графический пользовательский интерфейс предназначен для моделирования стохастических систем в виде НВССП с помощью графического редактора, симуляции НВССП (с возможной анимацией) и количественной оценки НВССП, выполняемой с использованием разнообразных механизмов принятия решений, применимых при различных предположениях о лежащих в основе моделей стохастических процессов. При этом Java-библиотека SIRIO позволяет интегрировать методы анализа, доступные в графическом интерфейсе, в пользовательские программы и инструменты для выполнения обширных параметрических исследований, где свойства систем (такие как производительность, надежность, пригодность) оцениваются при комбинациях многих параметров и вариантов моделей, специфицированных с помощью Java API.

Чтобы упростить такую интеграцию, модели НВССП можно экспортировать из графического интерфейса в виде Java-кода. Как только модель построена с использованием Java API, ее можно анализировать, вызвав доступные анализаторы, и результаты (например, стационарные распределения, временные ряды переходных процессов, графы классов состояний) могут быть сохранены в файл или показаны с использованием функций, доступных в графическом редакторе. В то время как построение и анализ моделей непосредственно в графическом редакторе полезны для быстрого перебора различных вариантов моделей и анализа их поведения, Java API дает возможность, варьируя стохастические параметры, проводить обширные параметрические исследования моделей.

ORIS предоставляет пользователю широкий выбор возможностей стохастического анализа, которые реализуют различные методы решения и оценки свойств НВССП, заданных с использованием одного и того же формата ввода параметров (например, «вознаграждений»[1] и условий остановки[2]). ORIS также поддерживает анализ НВСП, основанный на абстракции поведения НВСП — графе классов состояний. Каждый метод анализа накладывает различные ограничения на класс лежащих в основе НВССП стохастических процессов, что приводит к более эффективным методам решения.

Рассмотрим методы стохастического анализа, доступные в ORIS.

[1] «Вознаграждения» — функции, которые сопоставляют численные значения состояниям или переходам и которые используются для получения таких характеристик моделируемой системы, как потребление ресурсов, доступность, надежность и др.

[2] Условие остановки — логическое выражение, при истинности которого анализатор останавливает анализ до истечения ранее назначенного срока.

Марковский метод реализует стандартные методы переходного и стационарного анализа непрерывно-временных цепей Маркова (НВЦМ) [37]. Возможен анализ только ОССП, т. е. допустимы только экспоненциальные и мгновенные переходы; на практике, чтобы преодолеть это ограничение, каждый неэкспоненциальный переход (т. е. переход с временной задержкой, заданной неэкспоненциальным распределением) может быть заменен (перед анализом) последовательностью экспоненциальных переходов, полученных с помощью аппроксимации фазового типа.

Метод ограничения готовности к срабатыванию переходов реализует переходный анализ для марковских регенерирующих процессов (МРП). Возможен анализ только НВССП, в каждом состоянии которых не более чем один неэкспоненциальный переход разрешен к срабатыванию.

Регенеративный метод реализует переходный и стационарный анализ МРП с несколькими неэкспоненциальными переходами, разрешенными к срабатыванию в состояниях. Для стационарного анализа требуется ограниченное число срабатываний переходов до тех пор, пока не произойдет регенерация — состояние, в котором все переходы будут вновь готовы к срабатыванию, тогда как для переходного анализа этого ограничения нет.

«Направленный» метод реализует переходный анализ НВССП без ограничений на наличие регенераций и при этом перебирает классы состояний в графе классов стохастических состояний, начиная с начального состояния, до тех пор, пока самое раннее время срабатывания переходов в каждой последовательности срабатываний переходов не превысит целевую временную границу анализа.

Недетерминированный метод анализа пространства состояний НВССП поддерживает проверку качественных свойств модели: например, проверку того, может ли быть достигнута конкретная разметка или содержит ли граф классов состояний регенерацию в каждом цикле (это позволяет проводить более точный регенеративный анализ).

На протяжении многих лет ORIS успешно используется в различных контекстах и областях применения, например, в качестве графического интерфейса (GUI) для оценки показателей работоспособности в системах железнодорожной сигнализации [38], в качестве интерфейса прикладного программирования для выполнения анализа технического обслуживания в газо- и водо-распределительных сетях [39–40], для оценки наполняемости вагонов и времени ожидания пассажиров трамвайной линии [41].

Инструмент ORIS находится в свободном доступе [42], а исходный код библиотеки SIRIO доступен по лицензии AGPL на GitHub [43].

3. PetriNuts. PetriNuts (разработан и продолжает развиваться в Бранденбургском техническом университете г. Котбус, Германия) — программный комплекс, включающий инструменты: Snoopy, Charlie, Marcie, Spike, MC2, которые поддерживают работу с различными классами моделей сетей Петри и их стохастическими расширениями.

Snoopy [44] — базовый инструмент комплекса, предназначенный для графического конструирования, редактирования и визуализации, а также симуляции (с возможной анимацией «игры с фишками») сетевых моделей. Инструмент поддерживает следующие классы моделей:

— обычные сети Петри (СП);

— сети Петри с ингибиторными (inhibitor)/читающими (read)/обнуляющими (reset)/зависящими от маркировки дугами;

— непрерывные (continuous) сети Петри (НСП) [45];

— дискретно- и непрерывно-временные сети Петри (ДВСП и НВСП);

— стохастические непрерывные сети Петри (СНСП);
— обобщенные стохастические дискретно-временные сети Петри (ОДВССП);
— нечеткие (fuzzy) расширения НСП и СНСП [46];
— вышеперечисленные сетевые модели с цветными фишками.

Помимо сетевых моделей названных классов в графическом интерфейсе Snoopy также можно строить и визуализировать и другие графы, в частности, графы достижимости, мульти-терминальные и интервальные бинарные диаграммы решений [47], деревья неисправностей.

Для обмена данными с другими инструментами комплекса и сторонними средствами анализа Snoopy предоставляет несколько форматов файлов представления моделей, в частности, стандартные форматы PNML и SBML (Systems Biology Mark-up Language) [48].

Charlie [49] — реализованный на Java инструмент анализа сетевых моделей. Он принимает на вход все поддерживаемые в Snoopy классы моделей, но временные и стохастические атрибуты игнорируются, рассматривается только структура моделей. Инструмент предоставляет простой графический интерфейс, в котором можно определять тип сети (конечный автомат, маркированный граф, сеть со свободным выбором и др.), выяснять ее структурные свойства, находить P/T инварианты, ловушки и тупики, строить и визуализировать покрывающий граф и граф достижимости. На основе этих графов анализируются такие поведенческие свойства, как ограниченность, живость, обратимость, наличие динамических конфликтов, а также осуществляется проверка моделей на выполнимость формул темпоральных логик LTL и CTL [22]. Набор анализаторов и форматов ввода анализируемых сетевых моделей может быть расширен с помощью подключаемых модулей.

Marcie [50] — инструмент анализа ССП и ОССП, в которых можно дополнительно задавать «вознаграждения». Инструмент позволяет производить как переходный, так и стационарный анализ стохастических процессов. В зависимости от класса сети и размера пространства состояний для анализа могут применяться различные внутренние вычислители: точный численный (только для ограниченных сетей), аппроксимирующий (при работе которого маловероятные состояния отбрасываются) и симуляционный. В каждом из вычислителей реализовано несколько алгоритмов, так что пользователь может подбирать наиболее подходящий для конкретной модели. Кроме того, с использованием этих вычислителей может выполняться проверка моделей на базе формул темпоральных логик CTL, $CSRL$ (Continuous Stochastic Reward Logic) [51] и $PLTLc$ (Probabilistic Linear-time Temporal Logic with numerical constraints) [52].

Spike [53] — инструмент командной строки для эффективной симуляции стохастических, непрерывных и стохастических непрерывных сетей Петри и их расширений цветными фишками. Инструмент предоставляет разнообразные алгоритмы симуляции, позволяет определять сложные сценарии сеансов симуляции с варьируемыми параметрами и точно воспроизводить сохраненные сценарии.

MC2 [54] — реализованный на Java инструмент проверки моделей, свойства которых заданы формулами темпоральных логик $LTLc$ и $PLTLc$ на множестве трасс, полученных в результате симуляции стохастических и стохастических непрерывных СП и их расширений цветными фишками. Такие трассы могут быть получены, например, с помощью инструментов Snoopy и Spike. Пример использования системы MC2 в моделировании роста биопленок можно найти в [55].

Все инструменты программного комплекса PetriNuts свободно доступны для некоммерческого использования. Особенно активно и успешно этот комплекс применяется для моделирования биологических систем (см., например, [56]).

4. TimeNET. TimeNET [57] (с 90-х годов разрабатывался в Берлинском Техническом Университете, а с 2008 года поддерживается и развивается в Техническом Университете г. Ильменау (Германия)) — это программный инструмент для моделирования и оценки производительности систем с использованием стохастических сетей Петри и цепей Маркова. Основными особенностями инструмента являются: возможность оценки стохастических моделей с неэкспоненциально распределенными задержками срабатывания переходов, работа со стохастическими сетями Петри с цветными фишками и эффективные методы симуляции для моделей с «редкими» (происходящими с низкой вероятностью) событиями.

В TimeNET поддерживаются следующие классы моделей:

— обобщенные стохастические сети Петри (ОССП);

— детерминированные ОССП (ДОССП), расширяющие ОССП переходами с фиксированными задержками, при ограничении, что в любой разметке сети не более чем один такой переход готов к срабатыванию[3];

— расширенные детерминированные стохастические сети Петри (РДССП), где допускаются переходы с неэкспоненциальным распределением задержек срабатывания. Многие известные распределения (равномерное, треугольное, усеченное экспоненциальное, конечно-дискретное) относятся к этому классу распределений;

— стохастические сети Петри с цветными фишками (ЦССП);

— диаграммы состояний UML-профиля для MARTE (Modeling and Analysis of Real-Time and Embedded systems) [58], которые транслируются в РДССП для последующего анализа.

TimeNET спроектирована как модульная и расширяемая система, центральной частью которой является графический интерфейс пользователя (модуль GUI), реализованный на Java. Он служит графическим редактором моделей, средством запуска и управления различными модулями анализа и симуляции, написанными на C++ и C, демонстратором результатов их работы, а также интерактивным визуализатором динамического поведения моделей. Модуль GUI является универсальным в том отношении, что поддерживается единообразная работа с различными классами моделей. Каждый класс описывается XML-схемой, которая не только неявно определяет формат файла модели, но задает и графический интерфейс, включая отрисовку элементов модели и иконки, показываемые пользователю. Конкретная модель хранится в виде XML-файла, проверяемого на соответствие схеме. Для взаимодействия остальных модулей системы с модулем GUI разработан специальный интерфейс, через который каждый модуль может сообщать конкретный класс моделей, с которым он работает, и дополнять меню GUI специфичными командами. Такой подход позволяет легко расширять систему, сохраняя унифицированный интерфейс пользователя. Модули анализа и симуляции могут также запускаться командной строкой и скриптами.

Для класса РДССП, включающего ССП и ОССП как подклассы, поддерживаются следующие возможности:

— автоматическая/интерактивная симуляция в модуле GUI;

[3]Ограничение применяется лишь к численным методам анализа, но не к симуляции.

— структурный анализ (нахождение P/T инвариантов, ловушек, тупиков, конфликтных множеств, оценка размера пространства состояний);

— визуализация графа достижимости;

— численный анализ стационарного поведения;

— численный анализ переходного поведения;

— симуляция стационарного поведения:

— с автоматическим определением начала стационарной фазы и остановкой при достижении заданной точности;

— со специальными методами для моделей с «редкими» событиями;

— с комбинированием с численным анализом;

— симуляция переходного поведения;

— построение сложных показателей производительности, включающих вероятности количества фишек в местах сети и срабатываний переходов;

— автоматический обсчет параметризованной модели с варьируемым параметром;

— импорт/экспорт модели в стандартном формате PNML.

Для класса ЦССП поддерживаются следующие возможности:

— автоматическая/интерактивная симуляция в модуле GUI;

— симуляция стационарного поведения с эвристической остановкой по точности и специальными методами для «редких» событий;

— симуляция переходного поведения с контролем точности;

— построение сложных показателей производительности, включающих вероятности количества фишек в местах сети и срабатываний переходов;

— хранение параметров сложных моделей в базе данных.

Система TimeNet применялась для моделирования различных программных, аппаратных и индустриальных систем, например, при проектировании архитектуры облачной инфраструктуры управления железнодорожным транспортом [59] и моделировании мониторинга с помощью дронов [60]. Система TimeNET доступна для установки на 32/64-бит Windows и Linux ОС при условии некоммерческого использования.

5. PIPE2. Инструмент PIPE2 (Platform-Independent Petri Net Editor 2) [61], изначально разработанный в Imperial College London, а затем усовершенствованный/адаптированный усилиями других организаций, а также отдельных программистов, реализован на Java для работы с моделями на основе обычных сетей Петри (СП), обобщенных стохастических сетей Петри (ОССП) и их расширений цветными фишками (ОЦССП). Инструмент предоставляет интуитивно ясный графический интерфейс для построения модели, ее интерактивной симуляции, сохранения и загрузки в стандартном формате PNML, а также для вызова модулей, выполняющих различные виды анализа. Модульная структура PIPE2, фиксированный интерфейс взаимодействия GUI с модулями анализа и открытость исходного кода предполагают возможность достаточно простого расширения функциональности инструмента.

С поставляемым набором модулей доступны следующие возможности:

— структурный анализ, включающий:

— определение типа сетевой модели (конечный автомат, маркированный граф, (расширенная) сеть со свободным выбором и (расширенная) простая сеть);

— вывод матрицы инцидентности сети Петри;

— нахождение P/T инвариантов, минимальных ловушек и тупиков;

— поведенческий анализ, включающий:

— определение свойств живости, безопасности и наличия тупиковых ситуаций;

— построение и визуализацию покрывающего графа и графа достижимости;

— интерактивная/автоматическая анимация;

— численный анализ стационарного поведения, включающий нахождение:

— множества существенных состояний (разметок, в которых разрешены только экспоненциальные переходы) и вероятностей этих состояний;

— времени пребывания в каждом состоянии;

— среднего количества фишек для каждого места сети;

— вероятности присутствия конкретного количества фишек для каждого места сети;

— среднего количества срабатываний за единицу времени для каждого экспоненциального перехода;

— симуляция Монте-Карло для вычисления среднего количества фишек в каждом месте;

— построение с помощью графического языка сложных показателей производительности, которые могут затем вычисляться сторонними средствами анализа.

Следует заметить, что работа поставляемых модулей анализа ОЦССП осуществляется путем игнорирования фишек всех цветов, кроме одного, базового.

Модульность PIPE2, фиксированный интерфейс взаимодействия модуля GUI с модулями анализа и открытость исходного кода дают возможность достаточно простого расширения функциональности инструмента.

Доступны две основные версии PIPE2: v4 [62] и v5 [63]. Первая из них получила широкое распространение, но развивалась довольно хаотично, в результате чего исходный код этой версии сложно понимать и модифицировать. Версия v5 переписана заново, архитектурно более правильно, с избавлением от некоторых ошибок, имеющихся в версии v4, однако большая часть модулей анализа в версию v5 не перенесена.

6. Сравнение инструментов. В табл. 1 приведены результаты сравнения рассмотренных выше инструментальных систем на основе пяти критериев.

По первому критерию происходит сравнение с точки зрения того, какие *стохастические модели* поддерживаются инструментом. Все инструменты, за исключением Charlie, позволяют работать с ССП и ОССП. В системе ORIS базовой моделью являются НВССП, тогда как в инструменте Snoopy предоставляется возможность моделирования ДВССП. Детерминированные переходы могут присутствовать в стохастических моделях систем GreatSPN, ORIS и TimeNet, так как поддерживаются модели ДССП и ДОССП. Кроме того, системы GreatSPN, ORIS и TimeNet наиболее привлекательны при моделировании и анализе производительности и надежности стохастических систем, описываемых немарковскими процессами, поскольку эти инструменты ориентированы на работу с РССП, т. е. пользователь может использовать переходы с неэкспоненциальными распределениями. Почти все инструменты, кроме ORIS, Marcie и Charlie, поддерживают цветные расширения стохастических моделей. Следует подчеркнуть, что модули комплекса PetriNuts позволяют работать со стохастическими вариантами широкого спектра модификаций сетей Петри (СП с ингибиторными/читающими/обнуляющими/зависящими от маркировки дугами, СП с элементами нечетких логик, СП, в которых условия запуска переходов формулируются на основе возможно нецелочисленных параметров).

Вторым критерием сравнения являются *(нестохастические) функции* инструментов. Большинство инструментов предоставляют графический редактор, позволяющий строить,

Таблица 1

Сравнение инструментов

		GreatSPN	Oris	PetriNuts					TimeNET	PIPE2
				SNOOPY	MARCIE	CHARLIE	Spike	MC2		
Модели ССП	СП			●	●	●				●
	ССП, ОССП	●	●	●	●		●	●	●	●
	ДВССП			●						
	НВССП		●							
	ДССП, ДОССП	●	●						●	
	РССП, РДССП	●	●						●	
	ЦССП, ОЦССП	●		●			●	●	●	●
	СНСП			●			●	●		
Функциональность	Графический редактор	●	●	●		●			●	●
	Анимация	●	●	●					●	●
	Структурный анализ	●				●			●	●
	Граф достижимости	●				●			●	●
	Абстракция графа достижимости	●	●							
	Проверка моделей	●			●	●		●		
	Поддержка PNML	●		●	●		○		●	●
	Расширяемость	○	○		●	●				○
Стохастика	Численные решатели	●			●				●	●
	Симуляция стохастических процессов	●	●		●				●	○
	Сложные показатели производительности	●	●		●				●	○
Платформы	Java		●			●		●		●
	Linux	●	●	●	●	●	●	●	●	●
	Windows	●	●	●	●	●	●	●	●	●
	Mac	●	●	●	●	●	●	●		●
Доступность		О	С/О	НК	НК	НК	НК	НК	НК	О

редактировать и визуализировать структуру сетевых моделей, и интерактивный/автоматический симулятор сетевого поведения, в котором, как правило, анимация «игры с фишками» может управляться пользователем (выбор срабатывающих из набора разрешенных переходов осуществляется в интерактивном режиме) или может быть делегирована инструменту (случайный выбор). Такие функции инструментов могут помочь обучающимся работе с сетевыми моделями лучше понять их особенности и достоинства/недостатки при анализе моделируемых систем.

Системы GreatSPN, Charlie, Time Net и PIPE2 имеют широкий спектр возможностей структурного анализа сетевых моделей для определения P/T-инвариантов/потоков, структурной ограниченности, структурных конфликтных множеств, тупиков, ловушек,

взаимоблокировок и т. д. В GreatSPN, в отличие от других инструментов, можно также осуществлять структурный анализ на основе формул темпоральных логик, представляющих сетевые свойства.

В системах GreatSPN, Charlie, TimeNET, PIPE2 строятся графы достижимых состояний моделей, с помощью которых устанавливаются качественные поведенческие свойства: ограниченность, живость, достижимость и т. д. Особенностью GreatSPN является то, что различные типы графов достижимости используются также и для стохастического анализа.

Хорошо известно, что при анализе реальных систем возникает проблема «взрыва пространства состояний», с которой можно бороться несколькими способами, среди которых есть методы, основанные на построении абстракций, эквивалентных графам достижимости относительно некоторых наборов свойств. В GreatSPN, используя симметрию в моделях ОЦССП, строится символьный граф достижимости, каждая вершина которого соответствует набору состояний, в которых разрешены к срабатыванию одни и те же переходы, тогда как в ORIS выполняется анализ НВССП с применением графа класса состояний, в котором каждой вершине соответствует набор состояний, имеющих одну и ту же разметку и схожие временные характеристики.

Две характеристики, которые уникальны для GreatSPN, — это наличие модулей символьной проверки (symbolic model checker) на базе темпоральной логики CTL^* и стохастической проверки (stochastic model checker) на базе непрерывно-временной стохастической темпоральной логики CSL^{TA}. В Charlie возможна проверка только качественных (безвременных) свойств сетевых моделей. С другой стороны, отличительной особенностью Charlie является выявление свойств сети на основе теорем теории сетей Петри, т. е. свойств, которые могут быть выведены из наличия/отсутствия уже выявленных свойств у конкретной сети и которые становятся известными без дополнительного анализа. В инструментах Marcie и MC2 особое внимание уделено анализу стохастических свойств (например, возможно учитывать «вознаграждения» и численные ограничения (constraints)), представленных формулами логик $CSRL$ и $PLTLc$.

К существенным функциям инструмента относятся поддержка стандартного формата PNML, присутствующая в GreatSPN, TimeNet, PIPE2 и частично в PetriNuts (в таблице обозначается символом ∘), а также возможность расширения функциональности инструмента пользователем, которая в Marcie и Charlie реализована в виде поддержки подключаемых модулей (plugins), а в GreatSPN, PIPE2 и Oris — за счет открытости исходного кода, что, впрочем, требует более серьезных усилий. Отметим, что в Oris исходный код открыт только для библиотеки Sirio (обозначается символом ∘).

Третий критерий сравнения — это *возможности стохастического анализа* сетевых моделей, предоставляемые инструментами. Не всегда легко подобрать подходящий решатель и способы его использования при наличии широкого выбора доступных решателей, поэтому мы кратко перечислим характерные особенности таких модулей в рассмотренных инструментах. Численные решатели стохастического поведения реализованы в системах GreatSPN, ORIS, Marcie, TimeNet, PIPE2. Отличительной особенностью ORIS является возможность анализа стохастических моделей, в которых в каждом состоянии может быть параллельно включено несколько таймеров с неэкспоненциальным распределением, а также есть возможность работы с немарковскими процессами. Это значительно расширяет класс моделей, поддающихся численному решению, но требует более высоких вычислительных затрат, и поэтому при работе с инструментом следует выбирать наиболее под-

ходящий для конкретных целей метод анализа. На данный момент ORIS предоставляет самые передовые решатели среди доступных. Следует отметить, что ORIS и Marcie позволяют осуществлять анализ моделей, в которых можно дополнительно задавать «вознаграждения». Возможности включения нескольких таймеров с неэкспоненциальным распределением нет в других инструментах, и только в TimeNet и GreatSPN реализованы методы для оценки моделей, в каждом состоянии которых может быть включен не более чем один такой таймер. Особенность GreatSPN — это эффективные методы численного решения, основанные на симметриях для ОЦССП. TimeNet сосредоточен на численном анализе моделей РДССП в стационарном и переходном режимах. Особенность системы PIPE2 состоит в том, что она очень компактна и проста в работе.

Симуляция стохастических процессов поддерживается в GreatSPN, Marcie, Spike, TimeNet, PIPE2. Симулятор Монте-Карло инструмента GreatSPN вычисляет показатели производительности в переходном/стационарном режиме. Отличительной особенностью инструмента TimeNet являются эффективные методы стохастической симуляции для моделей с «редкими» (происходящими с низкой вероятностью) событиями. Spike предоставляет различные алгоритмы стохастической симуляции с заданием и сохранением сложных симуляционных сценариев, выполняемых с переменными параметрами.

Построение сложных показателей производительности (например, комбинация присутствия конкретного количества фишек в местах сети и среднего количества срабатываний переходов) выполняется в GreatSPN, ORIS, TimeNET и частично в PIPE2 (в таблице обозначается символом ∘).

Следующий критерий — это *платформы*, поддерживаемые инструментом. Все рассмотренные инструменты либо универсальны (реализованы на Java), либо имеют версии для всех трех основных платформ: Windows, Linux и Mac, за исключением того, что у TimeNET нет версии для Mac.

Последний критерий — это *доступность*, т. е. условия, на которых можно использовать инструмент. Все компоненты PetriNuts и система TimeNET поставляются в бинарном виде и свободно доступны для некоммерческого использования (в таблице обозначается НК). Инструменты GreatSPN и PIPE2 имеют открытый исходный код (обозначается О), первый по лицензии GPLv2, а второй по лицензии MIT. На использование бинарной версии Oris никаких ограничений нет, т. е. пользователи получают свободный доступ к инструменту (обозначается С), а его Java библиотека Sirio имеет открытый исходный код (обозначается О) по лицензии AGPL.

Заключение. В статье дается обзор инструментальных средств GreatSPN, Oris, PetriNuts, TimeNet и PIPE2, которые предназначены для работы со стохастическими сетевыми моделями и которые многократно применялись для решения различных задач моделирования и анализа сложно организованных систем с вероятностными характеристиками. Для каждого из рассматриваемых инструментов приведены основные его характеристики и возможности. Дано сравнение инструментов по следующим критериям: поддерживаемые классы стохастических моделей, (нестохастические) функции, возможности стохастического анализа, платформы, доступность. Результаты сравнения сведены в таблицу. Статья может быть полезной для исследователей и инженеров при выборе подходящего инструмента для работы со стохастическими моделями.

Список литературы

1. Reisig W. Petri Nets: An Introduction. V. 4. Springer, 1985. (EATCS Monographs on Theoretical Computer Science), DOI: 10.1007/978-3-642-69968-9.

2. Boyer M., Roux O. On the Compared Expressiveness of Arc, Place and Transition Time Petri Nets // Fundamenta Informaticae. 2008. Jan. V. 88. P. 225–249.

3. Berthomieu B., Diaz M. Modeling and verification of time dependent systems using time Petri nets // IEEE Transactions on Software Engineering. 1991. Mar. V. 17, N 3. P. 259–273. DOI: 10.1109/32.75415.

4. Molloy M. Performance Analysis Using Stochastic Petri Nets // IEEE Trans. Computers. 1982. V. 31, N 9. P. 913–917. DOI: 10.1109/TC.1982.1676110.

5. Vicario E., Sassoli L., Carnevali L. Using Stochastic State Classes in Quantitative Evaluation of Dense-Time Reactive Systems // IEEE Trans. Software Eng. 2009. V. 35, N 5. P. 703–719. DOI: 10.1109/TSE.2009.36.

6. Wang J. Stochastic Timed Petri Nets and Stochastic Petri Nets // Timed Petri Nets: Theory and Application. Boston, MA : Springer US, 1998. P. 125–153. DOI: 10.1007/978-l-4615-5537-7_5.

7. Ajmone Marsan M. et al. An introduction to generalized stochastic Petri nets // Microelectronics Reliability. 1991. Jan. V. 31, N 4. P. 699–725. DOI: 10.1016/0026-2714(91)90010-5.

8. Ajmone Marsan M., Chiola G. On Petri nets with deterministic and exponentially distributed Bring times // Advances in Petri Nets 1987, covers the 7th European Workshop on Applications and Theory of Petri Nets, Oxford, UK, June 1986. V. 266 / ed. by G. Rozenberg. Springer, 1986. P. 132–145. (Lecture Notes in Computer Science), DOI: 10.1007/3-540-18086-9_23.

9. Dugan J. et al. Extended Stochastic Petri Nets: Applications and Analysis // Performance '84, Proceedings of the Tenth International Symposium on Computer Performance Modelling, Measurement and Evaluation / ed. by E. Gelenbe. North-Holland, 1984. P. 507–519.

10. Ajmone Marsan M. et al. The Effect of Execution Policies on the Semantics and Analysis of Stochastic Petri Nets // IEEE Trans. Software Eng. 1989. V. 15, N 7. P. 832–846. DOI: 10.1109/32.29483.

11. German R., Lindemann C. Analysis of stochastic Petri nets by the method of supplementary variables // Performance Evaluation. 1994. May. V. 20, N 1–3. P. 317–335. DOI: 10.1016/0166-5316(94)90020-5.

12. Тарасюк И. В. Стохастические сети Петри — формализм для моделирования и анализа производительности вычислительных процессов // Системная информатика. Новосибирск, 2004. С. 135–194.

13. German R. Performance analysis of communication systems — modelling with non- Markovian stochastic Petri nets : Modeling with Non-Markovian Stochastic Petri Nets. Wiley, 2000. P. 456.

14. Biagi M. et al. Exploiting Non-deterministic Analysis in the Integration of Transient Solution Techniques for Markov Regenerative Processes // Quantitative Evaluation of Systems. Springer International Publishing, 2017. P. 20–35. DOI: 10.1007/978-3-319-66335-7_2.

15. Martina S. et al. Performance Evaluation of Fischer's Protocol through SteadyState Analysis of Markov Regenerative Processes // IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). 09/2016. P. 355–360. DOI: 10.1109/MASCOTS.2016.72.

16. Horvath A. et al. Transient analysis of non-Markovian models using stochastic state classes // Performance Evaluation. 2012. V. 69, N 7/8. P. 315–335. DOI: 10.1016/j.peva.2011.11.002.

17. Amparore E. Stochastic Modelling and Evaluation Using GreatSPN // ACM- SIGMETRICS Performance Evaluation Review. New York, NY, USA, 2022. June. V. 49, N 4. P. 87–91. DOI: 10.1145/3543146.3543165.

18. Amparore E. et al. Years of GreatSPN // Principles of Performance and Reliability Modeling and Evaluation: Essays in Honor of Kishor Trivedi on his 70th Birthday / ed. by L. Fiondella, A. Puliafito. Cham : Springer International Publishing, 2016. P. 227–254. DOI: 10.1007/978-3-319-30599-8-9.

19. K. J., Kristensen L. Coloured Petri Nets. Springer Berlin Heidelberg, 2009. DOI: 10.1007/b95112.

20. ISO/IEC. Software and Systems Engineering - High-level Petri Nets, Part 2: Transfer Format, International Standard ISO/IEC 15909, February 2011.

21. Kindler E. The Petri Net Markup Language and ISO/IEC 15909-2: Concepts, Status, and Future Directions // Tagungsband Entwurf komplcxcr Automatisierungssysteme EKA. 2006. P. 35–55.

22. Clarke E., Emerson E. Design and synthesis of synchronization skeletons using branching time temporal logic // Logics of Programs / ed. by D. Kozen. Springer Berlin Heidelberg, 1982. P. 52–71.

23. Deharbe D. A Tutorial Introduction to Symbolic Model Checking // Logic for Concurrency and Synchronisation / ed. by R. de Queiroz. Dordrecht : Springer Netherlands, 2003. P. 215–237. DOI: 10.1007/0-306-48088-3_5.

24. Beccuti M., Franceschinis G., Haddad S. Markov Decision Petri Net and Markov Decision Well-Formed Net Formalisms // Petri Nets and Other Models of Concurrency — ICATPN 2007 / ed. by J. Kleijn, A. Yakovlev. Springer Berlin Heidelberg, 2007. P. 43–62. DOI: 10.1007/978-3-540-73094-1 _6.

25. Emerson E., Sistla A. Symmetry and model checking // Formal Methods in System Design. 1996. V. 9, N 1. P. 105–131. DOI: 10.1007/BF00625970.

26. Babar J. et al. GrcatSPN Enhanced with Decision Diagram Data Structures // Applications and Theory of Petri Nets / ed. by J. Lilius, W. Penczek. Springer Berlin Heidelberg, 2010. P. 308–317.

27. Chaki S., Gurfinkel A. BDD-Based Symbolic Model Checking // Handbook of Model Checking / ed. by E. Clarke et al. Springer, 2018. P. 219–245. DOI: 10.1007/978-3-319-10575-8_8.

28. R. R., S. B., Zimmermann A. An Evaluation Framework for Comparative Analysis of Generalized Stochastic Petri Net Simulation Techniques // IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2020. V. 50. P. 2834–2844. DOI: 10.1109/TSMC.2018.2837643.

29. Pernice S. et al. Multiple Sclerosis Disease: A Computational Approach for Investigating Its Drug Interactions // Computational Intelligence Methods for Bioinformatics and Biostatistics / ed. by P. Cazzaniga et al. Cham : Springer International Publishing, 2020. P. 299–308. DOI: 10.1007/978-3-030-63061-4_26.

30. Amparore E., Donatelli S., Landini E. Modelling and Evaluation of a Control Room Application // Application and Theory of Petri Nets and Concurrency / ed. by W. van der Aalst, E. Best. Cham: Springer International Publishing, 2017. P. 243–263.

31. Richard L. Performance Results for the CSMA/CD Protocol Using GrcatSPN // Journal of Systems and Software. 1997. V. 37, N 1. P. 75–90. DOI: 10.1016/S0164-1212(96)00041-6.

32. Amparore E., Donatelli S. GreatTeach: A Tool for Teaching (Stochastic) Petri Nets // Application nd Theory of Petri Nets and Concurrency. Springer International Publishing, 2018. P. 416–425. DOI: 10.1007/978-3-319-91268-4_24.

33. Castagno P. et al. A computational framework for modeling and studying pertussis epidemiology and vaccination // BMC Bioinformatics. 2020. V. 21, N 8. P. 344. DOI: 10.1186/S12859-020-03648-6.

34. The GrcatSPN Framework. [El. Res.]: `https://github.com/greatspn/SOURCES`. Accessed: 2024-03-15.

35. Paolieri M. et al. The ORIS Tool: Quantitative Evaluation of Non-Markovian Systems // IEEE Trans. Software Eng. 2021. V. 47, N 6. P. 1211–1225. DOI: 10.1109/TSE.2019.2917202.

36. Carnevali L., Paolieri M., Vicario E. The ORIS tool: app, library, and toolkit for quantitative evaluation of non-Markovian systems // ACM SIGMETRICS Performance Evaluation Review. 2022. V. 49, N 4. P. 81–86. DOI: 10.1145/3543146.3543164.

37. Stewart W. Introduction to the Numerical Solution of Markov Chains. Princeton University Press, 1995. DOI: 10.1515/9780691223384.

38. Carnevali L. et al. Non-Markovian Performability Evaluation of ERTMS/ETCS Level 3 // Computer Performance Engineering - 12th European Workshop, EPEW 2015. V. 9272 / ed. by M. Beltran, W. Knottenbelt, J. Bradley. Cham : Springer, 2015. P. 47–62. (Lecture Notes in Computer Science), DOI: 10.1007/978-3-319-23267-6_4.

39. Biagi M. et al. Model-Based Quantitative Evaluation of Repair Procedures in Gas Distribution Networks // ACM Trans. Cyber Phys. Syst. 2019. V. 3, N 2. 19:1–19:26. DOI: 10.1145/3284037.

40. Carnevali L., Tarani F., Vicario F. Performability Evaluation of Water Distribution Systems During Maintenance Procedures // IEEE Trans. Syst. Man Cybern. Syst. 2020. V. 50, N 5. P. 1704–1720. DOI: 10.1109/TSMC.2017.2783188.

41. Carnevali L. et al. Using the ORIS Tool and the SIRIO Library for Model- Driven Engineering of Quantitative Analytics // Computer Performance Engineering / ed. byK. Gilly, N. Thomas. Cham: Springer International Publishing, 2023. P. 200–215. DOI: 10.1007/978-3-031-25049-1_13.

42. ORIS Tool. [El. Res.]: `http://www.oris-tool.org`. Accessed: 2024-03-15.

43. ORIS Tool: The Sirio Library. [El. Res.]: `https://github.com/oris-tool/sirio`. Accessed: 2024-03-15.

44. Heiner M. et al. Snoopy — A Unifying Petri Net Tool // Application and Theory of Petri Nets. PETRI NETS 2012. V. 7347 / ed. by S. Haddad, L. Pomello. Springer Berlin Heidelberg, 2012. P. 398–407. (Lecture Notes in Computer Science), DOI: 10.1007/978-3-642-31131-4_22.

45. David R., Alla H. Discrete, Continuous, and Hybrid Petri Nets. Springer Berlin Heidelberg, 2010. P. 550. DOI: 10.1007/978-3-642-10669-9.

46. Liu F., Heiner M., Gilbert D. Fuzzy Petri nets for modelling of uncertain biological systems // Briefings in Bioinformatics. 2018. Dec. V. 21, N 1. P. 198–210. DOI: 10.1093/bib/bbyll8.

47. Fujita M., McGeer P., Yang J. Multi-Terminal Binary Decision Diagrams: An Efficient DataStructure for Matrix Representation // Form. Methods Syst. Des. USA, 1997. Apr. V. 10, N 2/3. P. 149–169. DOI: 10.1023/A: 1008647823331.

48. Hucka M. et al. Systems Biology Markup Language (SBML) Level 2 Version 5: Structures and Facilities for Model Definitions // Journal of Integrative Bioinformatics. 2015. V. 12. N 2. P. 731–901. DOI: 10.2390/biecoll-jib-2015-271.

49. Heiner M., Schwarick M., Wegener J.-T. Charlie — An Extensible Petri Net Analysis Tool // Application and Theory of Petri Nets and Concurrency / ed. by R. Devillers, A. Valmari. Cham : Springer International Publishing, 2015. P. 200–211. DOI: 10.1007/978-3-319-19488-2_10.

50. Heiner M., Rohr C., Schwarick M. MARGIE — Model Checking and Reachability Analysis Done Efficiently // Application and Theory of Petri Nets and Concurrency / ed. by J.-M. Colom, J. Desel. Springer Berlin Heidelberg, 2013. P. 389–399. DOI: 10.1007/978-3-642-38697-8_21.

51. Baier C. et al. Model Checking Continuous-Time Markov Chains by Transient Analysis // Computer Aided Verification / ed. by E. Emerson, A. Sistla. Springer Berlin Heidelberg, 2000. P. 358–372.

52. Donaldson R., Gilbert D. A Model Checking Approach to the Parameter Estimation of Biochemical Pathways // Computational Methods in Systems Biology / ed. by M. Heiner, A. M. Uhrmacher. Springer Berlin Heidelberg, 2008. P. 269–287.

53. Chodak J., Heiner M. Spike — Reproducible Simulation Experiments with Configuration File Branching // Computational Methods in Systems Biology. Springer International Publishing, 2019. P. 315–321. DOI: 10.1007/978-3-030-31304-3_19.

54. Gilbert D., Donaldson R. A Monte Carlo model checker for probabilistic LTL with numerical constraints : tech. reP. / Bioinformatics Research Centre, University of Glasgow. 01/2008.

55. Gilbert D. et al. Spatial quorum sensing modelling using coloured hybrid Petri nets and simulative model checking // BMC Bioinformatics. 2019. V. 20, supplement 4. DOI: 10.1186/sl2859-019-2690-z.

56. Hcrajy M. et al. Snoopy's hybrid simulator: a tool to construct and simulate hybrid biological models // BMC Systems Biology. 2017. July. V. 11, N 1. DOI: 10.1186/S12918-017-0449-6.

57. Zimmermann A. Modelling and Performance Evaluation with TimeNET 4.4 // Quantitative Evaluation of Systems - 14th International Conference, QEST 2017. V. 10503 / ed. by N. Bertrand, L. Bortolussi. Springer, 2017. P. 300–303. (Lecture Notes in Computer Science), DOI: 10.1007/978-3-319-66335-7_19.

58. Selic B. Modeling And Analysis Of Realtime And Embedded Systems With Uml And Marte Developing Cyberphysical Systems. Elsevier Science & Technology, 2014. DOI: 10.1016/C2012-0-13536-5.

59. Zimmermann A. et al. Analysis of Safety-Critical Cloud Architectures with MultiTrajectory Simulation // Annual Reliability and Maintainability Symposium (RAMS). 01/2022. P. 1–7. DOI: 10.1109/RAMS51457.2022.9893923.

60. Fedorova A., Beliautsou V., Zimmermann A. Colored Petri Net Modelling and Evaluation of Drone Inspection Methods for Distribution Networks // Sensors. 2022. V. 22, N 9. DOI: 10.3390/s22093418.

61. Dingle N., Knottenbelt W., Suto T. PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets // SIGMETRICS Performance Evaluation Review ACM. New York, NY, USA, 2009. Mar. V. 36, N 4. P. 34–39. DOI: 10.1145/1530873.1530881.

62. Platform Independent Petri Net Editor v4. [El. Res.]: `https://sourceforge.net/projects/pipe2`. Accessed: 2024-03-15.

63. PIPE 5. [El. Res.]: `https://github.com/sarahtattersall/PIPE`. Accessed: 2024-03-15.

Быстров Александр Васильевич — старший научный сотрудник Института систем информатики им. А. П. Ершова СО РАН, доцент кафедры «Системы информатики» Новосибирского государственного университета. E-mail: `avb@iis.nsk.su`. Кандидат физ.-мат. наук с 2008 г. Автор и соавтор более 40 научных работ. Основные научные интересы — спецификация и верификация параллельных систем и систем реального времени и соответствующие программные средства и языки программирования.

Bystrov Alexandr Vasilievich graduated Novosibirsk State University, Ph. D. (Candidate of science) degree since 2008, senior researcher at the A. P. Ershov Institute of Informatics Systems SB RAS and associate professor at the Novosibirsk State University. Author of more than 40 scientific papers. Research interests: specification and verification of concurrent/real-time systems, related software tools and languages.

Вирбицкайте Ирина Бонавентуровна — главный научный сотрудник, зав. лабораторией «Теории параллельных процессов» Института систем информатики им. А. П. Ершова СО РАН, профессор Новосибирского государственного университета. E-mail: `virb@iis.nsk.su`. Ученые степени кандидата (1990) и доктора (2002) физико-математических наук, ученые звания старшего научного сотрудника (1995), профессора (2008) по специальности «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей». Автор и соавтор более 150 научных трудов. Основные направления научной деятельности — теория параллельных систем и процессов, включая формальные модели параллелизма, спецификацию и верификацию параллельных систем и систем реального времени, автоматическое конструирование параллельных программ.

Virbitskaite Irina Bonaventurovna is a chief researcher, the head of the laboratory

"Theory of Concurrent Processes" at the A. P. Ershov Institute of Informatics Systems SB RAS and professor at Novosibirsk State University. Academic degrees of Candidate (1990) and Doctor (2002) of Physical and Mathematical Sciences, academic titles of senior researcher (1995), professor (2008), in the specialty "Mathematical and software support of computers, complexes and computer networks". Author and co-author of more than 150 scientific papers. Research interests: theory of concurrent systems and processes, including formal models for concurrency, specification and verification of concurrent and real–time systems, automatic design of parallel programs.

Елена Сергеевна Ошевская — младший научный сотрудник Института систем информатики им. А. П. Ершова (Новосибирск). E-mail: `eso@iis.nsk.su`. В 2005 г. закончила Механико-математический факультет Новосибирского государственного университета по направлению «Математика». В 2013 г. получила ученую степень кандидата физико-математических наук, защитив диссертацию на тему «Категорные методы исследования полукубических множеств и пространств как моделей параллельных процессов». Область научных интересов: спецификация и верификация параллельных систем и систем реального времени.

Elena Sergeevna Oshevskaya is a junior researcher at A. P. Ershov Institute of Informatics Systems (Novosibirsk city). Graduated from the Faculty of Mechanics and Mathematics of Novosibirsk State University in the field of Mathematics (2005), academic degree of Candidate of Physical and Mathematical Sciences, having defended her dissertation on the topic "Categorical methods in studying semicubic sets and spaces as models of concurrent processes". Research interests: specification and verification of concurrent and real-time systems.

Дата поступления — 19.03.2024

IMPLEMENTATION OF SEARCHING FOR THE MOST FREQUENT DNA SEQUENCES USING THE KOKKOS LIBRARY

M. Kozlov, E. Panova, I. Meyerov

Lobachevsky State University of Nizhny Novgorod,
603950, Nizhny Novgorod, Russia

DOI: 10.24412/2073-0667-2024-2-58-71
EDN: TGQKBV

Nowadays, the wide variety of existing architectures raises the problem of developing universal approaches to programming. Various frameworks enable single-source code creation for multiple devices, for example, CPU, GPU, FPGA. Such frameworks include OpenCL, OpenACC, Kokkos, Alpaka and others. However, the problem of efficiency and performance portability remains relevant. It is not always possible to create one code that works efficiently on different devices because of their specific architectures. This article discusses performance aspects in relation to the Kokkos library, a widely used framework for creating cross-platform code.

As a benchmark, we consider a bioinformatics problem to find the most frequent DNA sequences of certain length. It is assumed that important genetic information can be encrypted in such sequences. DNA sequence can be represented as a string consisting of four characters "A", "C", "G", "T", which denote corresponding nucleobases. Therefore, the problem reduces to counting fixed-length patterns in DNA and can be solved using existing string matching algorithms. Faro and Lecroq (2013) reviewed and classified exact string matching algorithms and experimentally evaluated them on different kinds of texts. Hakak et al. (2019) showed the latest advancements in the field of string matching algorithms and designated modern trends and challenges. They analyzed various classes of algorithms and drew conclusions about the limitations and effectiveness of different string matching algorithms for various applications. In this article, we have chosen two algorithms for consideration: the well-known Rabin-Karp algorithm and the Hash3 algorithm from the Hash$q$ family [Lecroq 2007]. The Hash3 algorithm is one of the most effective algorithms for short-length patterns of approximately 8 to 128 characters long. Both these algorithms are based on hashing and are well applicable for genome analysis. For verification and comparison, we also consider a simple naive algorithm based on sequential pattern matching.

The naive algorithm consists of character-by-character comparison of all fixed-length patterns. This algorithm is not effective enough, but has great potential for parallelization. We received an acceleration of up to 35 times when ported the parallel naive algorithm from CPU to GPU. The Rabin-Karp algorithm allows us to eliminate character-by-character comparisons effectively using hashing and shows better efficiency compared to the naive algorithm on both CPU and GPU. Our cross-platform parallel implementation of the Rabin-Karp algorithm is approximately 1.25 times faster than the naive algorithm on CPU and 2 times faster on GPU. The Hash3 algorithm cuts off character-by-character comparisons extremely efficiently. Because of this, the Hash3 algorithm is an order of magnitude faster than the naive algorithm. Due to the almost absence of character-by-character comparisons,

This work was funded by the Ministry of Science and Higher Education of the Russian Federation, project No. FSWR-2023-0034.

the algorithm is memory bound and has less potential for parallelization. The Hash3 algorithm was accelerated by 7 times on GPU relative to CPU.

We implement these algorithms for CPU and GPU using OpenMP, Cuda and Kokkos technologies. We demonstrate that when using Kokkos with a naive algorithm, the performance loss does not exceed 10 % relative to the OpenMP version. Losses are caused by the compiler making more efficient use of SIMD calculations in the OpenMP implementation when matching patterns. There is no performance loss for the Rabin-Karp and Hash3 algorithms when porting the OpenMP version to Kokkos. Speedup of all algorithms is about 14 times on 16 physical cores. It is worth noting that the Hash3 algorithm showed a noticeable improvement on the CPU when using hyper-threading, unlike other algorithms under consideration. This can be explained by more efficient memory management. Speedup on 32 threads and 16 physical cores for the naive algorithm and the Rabin-Karp algorithm is 16–17 times, while for the Hash3 algorithm it is 25 times.

Next, we run the developed code on the GPU and show that the Kokkos version of the Rabin-Karp algorithm loses to the Cuda version on the GPU by no more than 10 %. At the same time, the Kokkos versions of the naive and Rabin-Karp algorithms outperform our Cuda baseline version by 10–20 %. The authors did not set themselves the goal of optimizing the Cuda code. We believe that it is possible to optimize the Cuda code to match the performance of the Kokkos version. However, it is noteworthy that sometimes the baseline Kokkos version runs faster than the baseline Cuda version.

Overall, we demonstrate that in many cases the Kokkos version works as well as native OpenMP or Cuda code. In the worst case, the performance loss was no more than 10 %. We believe that paying this price is reasonable in order to run a single code on different devices.

Key words: Kokkos, single-source programming, cross-platform software, heterogeneous computing, program performance, string matching algorithms, bioinformatics.

References

1. Gaster B., Howes L., Kaeli D. R., Mistry P., and Schaa D. Heterogeneous computing with openCL: revised openCL. Newnes, 2012.

2. Farber R. Parallel programming with OpenACC. Newnes, 2016.

3. Kokkos 3: Programming model extensions for the exascale era / Trott C. R., Damien LG, Arndt D., Ciesko J., Dang V., Ellingwood N., Gayatri R., Harvey E., Hollman D. S., Ibanez D., et al. // IEEE Transactions on Paralleland Distributed Systems. 2021. V. 33, N 4. P. 805–817.

4. Alpaka — an abstraction library for parallel kernel acceleration / Zenker E., Worpitz B., Widera R., Huebl A., Juckeland G., Knupfer A., Nagel W. E., and Bussmann M. // 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) / IEEE. 2016. P. 631–640.

5. Reinders J. et al. Data parallel C++: mastering DPC++ for programming of heterogeneous systems using C++ and SYCL. Springer Nature, 2021. P. 548.

6. The Kokkos library. [Electron. Res.]: `https://github.com/kokkos/kokkos`. Date of access: 10.01.2024.

7. Subirana J. A., Messeguer X. The most frequent short sequences in non-coding DNA // Nucleic acids research. 2010. V. 38, N 4. P. 1172–1181.

8. Faro S., Lecroq T. The exact online string matching problem: A review of the most recent results // ACM Computing Surveys(CSUR). 2013. V. 45, N 2. P. 1–42.

9. Exact string matching algorithms: survey, issues, and future research directions / Hakak S. I., Kamsin A., Shivakumara P., Gilkar G. A., Khan W. Z., and Imran M. // IEEE access. 2019. V. 7. P. 69614–69637.

10. Stephen G. A. String searching algorithms. World Scientific, 1994.

11. Al-Khamaiseh K., Alshagarin S. A survey of string matching algorithms // Int. J. Eng. Res. Appl. 2014. V. 4, N 7. P. 144–156.

12. Karp R. M., Rabin M. O. Efficient randomized patternmatching algorithms // IBM Journal of Research and Development. 1987. V. 31, N 2. P. 249–260.

13. Lecroq T. Fast exact string matching algorithms // Information Processing Letters. 2007. V. 102, N 6. P. 229–235.

14. Galil Z. A constant-time optimal parallel string-matching algorithm // Journal of the ACM (JACM). 1995. V. 42, N 4. P. 908–918.

15. Park J. H., George K. M. Efficient parallel hardware algorithms for string matching // Microprocessors and Microsystems. 1999. V. 23,N 3. P. 155–168.

16. Accelerating string matching using multi-threaded algorithm on GPU / Lin C. H., Tsai S. Y., Liu C. H., Chang S. C., and Shyu J. M. // 2010 IEEE Global Telecommunications Conference GLOBECOM 2010 / IEEE. 2010. P. 1–5.

17. Kouzinopoulos C. S., Michailidis P. D., Margaritis K. G. Multiple string matching on a GPU using cudas //Scalable Computing: Practice and Experience. 2015. V. 16, N 2. P. 121–138.

18. Kozlov M. A., Panova E. A., Meerov I. B. Implementation of searching for the most frequent DNA sequences using the Kokkos library // Mathematical modeling and supercomputer technologies. Proceedings of the XXIII International Conference (N. Novgorod, November 13–16, 2023) / Ed. prof. D. V. Balandina. Nizhny Novgorod: Publishing House of Nizhny Novgorod State University, 2023. ISBN 978-5-91326-834-1. 2023. P. 73–78.

19. Benchmark source code. [Electron. Res.]: `https://github.com/Mishaizlesa/most_common_string_kokkos`. Date of access: 10.01.2024.

20. DNA Bank (National library of medicine). [Electron. Res.]: `https://www.ncbi.nlm.nih.gov/genbank`. Date of access: 10.01.2024.

РЕАЛИЗАЦИЯ ПОИСКА НАИБОЛЕЕ ЧАСТО ВСТРЕЧАЮЩИХСЯ ПОСЛЕДОВАТЕЛЬНОСТЕЙ ДНК С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ KOKKOS

М. А. Козлов, Е. А. Панова, И. Б. Мееров

Национальный исследовательский Нижегородский государственный университет им. Н. И. Лобачевского,
603950, Нижний Новгород, Россия

УДК 004.424, 575.112
DOI: 10.24412/2073-0667-2024-2-58-71
EDN: TGQKBV

Существующее на текущий день большое разнообразие архитектур ставит вопрос разработки универсального программного обеспечения. В связи с этим появляются и развиваются различные программные средства, позволяющие создавать единый кроссплатформенный код для запуска на CPU, GPU, FPGA и других архитектурах. Тем не менее, остается вопрос эффективности и переносимости производительности разработанного кода. В данной работе мы исследуем этот и другие аспекты применительно к библиотеке Kokkos, которая на сегодняшний день является одним из наиболее популярных средств для создания кроссплатформенного кода. В качестве бенчмарка мы рассматриваем задачу из области биоинформатики по поиску наиболее часто встречающихся последовательностей ДНК, которая решается с использованием строковых алгоритмов. Мы приводим несколько алгоритмов решения задачи, реализуем их с использованием технологий OpenMP, Cuda и Kokkos и демонстрируем, что потери производительности при использовании Kokkos не превышают 10 %, в то время как код может быть запущен как на CPU, так и на GPU.

Ключевые слова: Kokkos, кроссплатформенное ПО, гетерогенные вычисления, оптимизация программ, строковые алгоритмы, биоинформатика.

Введение. На сегодняшний день существует большое число аппаратных архитектур, требующих уникального подхода к разработке параллельного программного обеспечения. Как следствие, появляется множество реализаций одного и того же кода под различные архитектуры, заниматься одновременной поддержкой которых требует достаточно большого количества ресурсов. Требуются средства разработки универсального программного обеспечения, и такие средства существуют и активно развиваются в последнее десятилетие. Среди них можно отметить OpenCL [1], OpenACC [2], Kokkos [3], Alpaka [4], oneAPI [5] и др. Однако вопрос эффективности и переносимости производительности разработанного с помощью этих средств кода остается открытым.

В рамках данной работы рассматривается программный пакет Kokkos [3, 6], являющийся удобным C++ фреймворком для создания переносимого кода, который может быть

Работа выполнена при поддержке Министерства науки и высшего образования России, проект № FSWR-2023-0034.

запущен на различных аппаратных платформах (CPU, GPU). Мы исследуем некоторые аспекты написания переносимых производительных программ, демонстрируем особенности организации кода с использованием синтаксиса Kokkos и показываем, что Kokkos позволяет разрабатывать универсальное программное обеспечение для различных устройств без существенных потерь в производительности.

В качестве демонстрационного примера мы рассматриваем задачу из области биоинформатики по поиску наиболее часто встречающихся паттернов фиксированной длины во фрагменте ДНК [7]. Предполагается, что такие паттерны потенциально могут нести некоторую генетическую информацию. ДНК представляется в виде последовательности четырех азотистых оснований, традиционно обозначаемых символами «A», «C», «G», «T». Таким образом, задача сводится к поиску наиболее часто встречающихся подстрок небольшой фиксированной длины m ($4 \lesssim m \lesssim 100$) в последовательности ДНК достаточно большой длины n ($10^3 \lesssim n \lesssim 10^6$), состоящей из символов четырехбуквенного алфавита. В такой постановке задача решается с использованием строковых алгоритмов. Стоит отметить, что в биоинформатике имеет смысл учитывать генетические мутации и выполнять поиск последовательностей с не более чем k несоответствиями («k-mismatches problem»), однако в данной статье рассматриваются только точные совпадения.

Задача поиска подстроки встречается во многих областях, таких как обработка естественных языков, речи, обработка изображений, компьютерное зрение, распознавание образов, биоинформатика. На сегодняшний день существует множество алгоритмов, основанных на посимвольном сравнении, хешировании, побитовых операциях, конечных автоматах и т. д. [8–11]. Многие алгоритмы показывают очень хорошие результаты на данных, характерных для определенных областей приложений. Фаро и Лекрок [8] представляют классификацию существующих алгоритмов и делают сравнительный анализ их эффективности для текстов из различных областей приложений. Хакак и др. [9] демонстрируют последние достижения в области алгоритмов на строках, обозначают современные проблемы и тенденции и делают выводы касательно применимости тех или иных алгоритмов в различных приложениях.

Задачи биоинформатики имеют свою специфику, связанную с небольшим размером алфавита и необходимостью выполнять поиск множества различных паттернов в одном и том же тексте. Для таких задач хорошие результаты демонстрируют алгоритмы, основанные на хешировании, т. к. они дают возможность предобработки текста и легко расширяются на случай поиска множества паттернов. В данной работе мы рассматриваем два разных алгоритма из этого класса: известный алгоритм Рабина-Карпа [12] и алгоритм Hash3 из семейства Hash$q$ [13], который является одним из наиболее эффективных алгоритмов для решения данной задачи [8]. Для сравнения в качестве базового алгоритма рассматривается наивный алгоритм, основанный на последовательном посимвольном сравнении подстрок. Стоит отметить, что наивный алгоритм и алгоритм Рабина-Карпа требуют намного больше посимвольных сравнений, чем алгоритм Hash3, что накладывает свои условия при оптимизации кода.

Одним из важных вопросов при реализации строковых алгоритмов является возможность их адаптации под современные параллельные системы [14–15], в том числе под графические ускорители [16–17]. В данной работе мы предлагаем параллельные реализации перечисленных выше алгоритмов применительно к рассматриваемой задаче, исследуем их эффективность на CPU и GPU. В качестве базовой технологии распараллеливания мы используем OpenMP для CPU и Cuda для GPU. Также мы рассматриваем возможность ис-

пользования библиотеки Kokkos для написания кроссплатформенного кода, изучаем некоторые аспекты производительности и делаем вывод о возможности использования Kokkos для подобного рода задач. Мы демонстрируем, что потери производительности при использовании Kokkos не превышают 10 %, в то время как один и тот же код может быть запущен как на CPU, так и на GPU.

Работа является расширенным вариантом материалов доклада Козлова М. А., Пановой Е. А., Меерова И. Б. «Реализация поиска наиболее часто встречающихся последовательностей ДНК с использованием библиотеки Kokkos» [18] в рамках конференции «Математическое моделирование и суперкомпьютерные технологии 2023», рекомендована к публикации программным комитетом конференции «Математическое моделирование и суперкомпьютерные технологии».

1. Постановка задачи. Рассматривается задача поиска наиболее часто встречающихся паттернов в ДНК. Фрагмент ДНК представляется в виде последовательности символов четырехбуквенного алфавита «A», «C», «T», «G», где каждая буква соответствует одному из четырех азотистых оснований: аденин, цитозин, тимин, гуанин. С биологической точки зрения имеет смысл искать достаточно короткие паттерны, при этом точных границ не существует, рассматриваются как совсем короткие паттерны (от 1 до 6 символов), так и паттерны длиной 20–30 символов и более [7]. Как было ранее отмечено, выполняется поиск точных совпадений паттернов без учета мутаций. Таким образом, задача поиска наиболее часто встречающихся паттернов в ДНК может быть сведена к задаче вычисления числа вхождений каждой подстроки длины m ($1 \leqslant m \leqslant 1000$) в текст четырехбуквенного алфавита длины n ($10^3 \leqslant n \leqslant 10^6$). Общий псевдокод решения данной задачи демонстрирует листинг 1. Функция `patternCount` возвращает число вхождений паттерна в текст и представляет собой реализацию одного из рассматриваемых алгоритмов поиска подстроки. Подробнее данные алгоритмы описаны в разделе 2.

Листинг 1. Общий псевдокод программы. Входные данные: текст длины n и длина паттерна m. Выходные данные: массив частот для каждого паттерна.

```
int[] mostFrequentPatterns(text : string, m : int):
    int n = text.length
    frequencies : array [0..n − m + 1] of int
    for i from 0 to n − m + 1:  # parallel loop
        pattern = text[i..i + m − 1]
        frequencies[i] = patternCount(text, pattern, n, m)
    return frequencies
```

При решении задачи стоит учитывать, что один паттерн может встречаться в тексте много раз, и следует вычислять частоту вхождения паттерна однократно, исключая его из последующего рассмотрения. Однако множественные проверки не влияют на асимптотическую сложность и сравнительный анализ алгоритмов, поэтому мы не учитываем этот факт и рассматриваем псевдокод 1 как некоторый бенчмарк, основанный на сравнении строковых паттернов.

Поскольку в задаче требуется выполнять поиск многих паттернов, то имеет смысл выстраивать параллелизм именно по паттернам. Таким образом, параллельная схема не зависит от используемого алгоритма поиска. Специфика исходных данных позволяет утверждать, что параллельные подзадачи являются достаточно сбалансированными и будут выполняться за примерно одинаковое время.

2. Алгоритмы. 2.1. *Наивный алгоритм.* Представленный в данном разделе алгоритм, далее называемый «наивным», реализует очевидную идею поэлементного последовательного сравнения паттернов в цикле (листинг 2). Асимптотическая сложность данного алгоритма $O(mn)$, где n — длина исходного текста, а m — длина искомого паттерна. Поскольку для решения поставленной задачи наивный алгоритм применяется к каждому паттерну длины m исходного ДНК (листинг 1), то общая сложность всего алгоритма `mostFrequentPatterns` в худшем случае равна $O(mn^2)$. Данный алгоритм можно оптимизировать, например, обратить внимание на то, что в текущем варианте сравнение двух паттернов текста проводится минимум дважды, однако это не изменит асимптотику алгоритма. В данной работе наивный алгоритм используется для проверки корректности работы более сложных алгоритмов и в качестве точки отсчета при проведении анализа эффективности.

Листинг 2. Псевдокод наивного алгоритма.

```
int naive(text : string, pattern : string, n : int, m : int):
    int answer = 0
    for i = 0 to n - m + 1:
        bool flag = true
        for j = 0 to m - 1:
            if text[i + j] != pattern[j]:
                flag = false
                break
        if flag == true:
            answer = answer + 1
    return answer
```

2.2. *Алгоритм Рабина-Карпа.*

Алгоритм Рабина-Карпа [12] является одним из самых известных алгоритмов поиска подстроки в тексте. Алгоритм основан на идее хеширования подстрок. Основная идея алгоритма заключается в том, чтобы сравнивать не сами паттерны, а их значения хеш-функции. Псевдокод алгоритма Рабина-Карпа представлен листингом 3. Алгоритм Рабина-Карпа имеет ту же сложность в худшем случае, что и наивный алгоритм, но на практике обычно работает быстрее.

Листинг 3. Псевдокод алгоритма Рабина-Карпа.

```
int rabinKarp(text : string, pattern : string, n : int, m : int):
    int answer = 0
    int hashT = hash(text[0..m - 1])
    int hashP = hash(pattern)
    for i = 0 to n - m + 1:
        if hashT == hashP:
            answer = answer + 1
        hashT = hash(text[i..i + m - 1])
    return answer
```

Для того чтобы алгоритм Рабина-Карпа работал быстрее наивного алгоритма, требуется, чтобы хеш-функция подстроки вычислялась достаточно быстро. Один из вариантов – предварительно вычислить хеш-функции всех подстрок текста. Другая, более оптималь-

ная опция — воспользоваться тем фактом, что две соседние подстроки длины m в тексте имеют общую подпоследовательность длиной $m-1$. Использование кольцевого хеша позволяет учесть эту особенность и вычислять значение хеш-функции за время, не зависящее от длины паттерна. В данной работе в качестве кольцевого хеша использовался полиномиальный хеш:

$$h(s_0 s_1 ... s_{m-1}) = (s_0 p^{m-1} + s_1 p^{m-2} + ... + s_{m-1} p^0) \bmod q, \quad (1)$$

где $s_0 s_1 ... s_{m-1}$ — подстрока длины m, p и q – параметры, mod – операция взятия остатка. Таким образом:

$$h(s_{i+1} s_{i+2} ... s_{i+m}) = (h(s_i s_{i+1} ... s_{i+m-1}) p - s_i p^m + s_{i+m}) \bmod q. \quad (2)$$

Для того чтобы операции и умножения и взятия остатка можно было заменить на побитовые операции и получить преимущество в скорости, в качестве параметров были выбраны значения $p = 2$ и $q = 2^{31} - 1$.

2.3. *Алгоритм Hash3.* Алгоритм Hash3 [13] продолжает идею алгоритма Рабина-Карпа по отсечению как можно большего количества посимвольных сравнений строк. Для искомого паттерна предварительно вычисляется расстояние от каждой его трехбуквенной подстроки до конца паттерна. Это расстояние записывается в хеш-таблицу `shift`, ключом в которой является значение хеш-функции трехбуквенной подстроки. Для вычисления хеш-функции трехбуквенной подстроки используется полиномиальный хеш (формула 1). Алгоритм Hash3 на каждой итерации вычисляет хеш-функцию текущей трехбуквенной подстроки текста и производит сдвиг на значение, сохраненное в хеш-таблице. Таким образом, алгоритм перемещается по памяти непоследовательно, постоянно пытаясь перейти к концу предполагаемого вхождения паттерна в исходный текст. Это значительно уменьшает требуемое количество посимвольных сравнений.

Алгоритм в исходном виде, представленный в работе [13], был оптимизирован, исходя из специфики рассматриваемой задачи (листинг 4). Поскольку алфавит состоит всего из 4 символов, и всевозможных комбинаций из трех букв существует только 64, то вместо хеш-таблицы `shift` используется массив с индексами от 0 до 63. Для оптимизации работы на GPU данный массив выделяется в локальной памяти ядра.

Листинг 4. Псевдокод алгоритма Hash3.

```
int Hash3(text: string, pattern : string, n : int, m : int):
    int shift_size = 64
    shift : array [0 .. shift_size - 1] of int
    for ind = 0 to shift_size - 1:
        shift[ind] = m
    for j = 2 to (m - 1):
        shift[hash(patern[j - 2 : j])] = m - 1 - j
    int answer = 0
    int j = m - 1
    while j < n:
        int temp_shift = 1
        while temp_shift != 0 and j < size:
            temp_shift = shift[hash(text[j - 2 : j])]
            j += temp_shift
```

```
        if j < n and text[j - m + 1 : j] == pattern:
            answer += 1
            j += 1
    return answer
```

С точки зрения производительности данный алгоритм имеет некоторые особенности по сравнению с рассмотренными ранее наивным алгоритмом и алгоритмом Рабина-Карпа. Во-первых, большое количество ветвлений может ухудшать работу на GPU. Это связано с тем, что графические ускорители основаны на параллельной архитектуре, где большое количество ядер одновременно выполняет одну и ту же инструкцию на разных данных. В случае ветвления кода те ядра, для которых не срабатывает условие, простаивают, что снижает эффективность параллельного выполнения. Во-вторых, непоследовательное перемещение по памяти приводит к тому, что производительность кода сильно ограничена пропускной способностью памяти как на CPU, так и на GPU.

3. Программная реализация. Для каждого из рассмотренных в разделе 2 алгоритмов было подготовлено три программных реализации. Одна из них является нативной для CPU и использует технологию OpenMP для организации параллелизма. Вторая предназначена для запусков только на GPU и написана на языке Cuda. Третья реализация, разработанная с использованием Kokkos, является универсальной и может быть запущена как на CPU, так и на GPU.

Параллелизм в версии OpenMP для CPU выполняется тривиальным образом: задачи распределяются по итерациям внешнего цикла (листинг 1) согласно статическому расписанию, т. е. используется стандартная OpenMP директива `#pragma omp parallel for`. Ядро в версии Cuda реализует один из алгоритмов поиска количества вхождений паттерна в текст (функция `patternCount` листинга 1). В качестве аргументов ядро принимает массив частот, исходный текст, длину текста и искомого паттерна. Алгоритмы реализованы тривиальным образом, специальных оптимизаций под GPU произведено не было.

Реализация алгоритмов на Kokkos универсальна и может работать как на CPU, так и на GPU. Также Kokkos выполняет ряд оптимизаций под архитектуру, на которой производится запуск. Для CPU выполняется автоматическая векторизация, выбирается оптимальное число потоков. На GPU автоматически определяется число блоков Cuda и потоков в блоке. Кроме того, Kokkos может определять оптимальную структуру хранения многомерного массива в зависимости от архитектуры. Например, в связи с особенностями устройства памяти матрицы на CPU обычно выгодно хранить по строкам, а на GPU по столбцам, и Kokkos это учитывает. Для инкапсуляции подобного рода деталей Kokkos предоставляет для хранения многомерных статических или динамических массивов специальную структуру данных `View`, которая при объявлении принимает в качестве аргумента шаблона информацию об аппаратном окружении.

Для написания версии Kokkos применительно к рассматриваемой задаче требовалось представить массив частот и исходный текст как два объекта типа `View`. Для организации параллелизма используется синтаксис Kokkos `parallel_for` (листинг 5). Ядро параллельной программы на Kokkos практически идентично ядру в версии Cuda, за исключением некоторых незначительных синтаксических особенностей.

Листинг 5. Фрагмент программы на Kokkos.

```
Kokkos::View<int*,Kokkos::SharedSpace> freq("frequency", size);
Kokkos::View<const char*,Kokkos::SharedSpace> data("txt", size);
```

```
Kokkos::parallel_for("pattern_search", size-len+1,
    KOKKOS_LAMBDA (int i) { // algorithm kernel }
);
```

Программная реализация находится в открытом доступе [19].

4. Эксперименты. 4.1. *Тестовая конфигурация.* Все запуски проводились на кластере ННГУ «Лобачевский». Запуски на CPU производились на следующей конфигурации:

— 2 процессора Intel Sandy Bridge E5-2660 2.2 GHz (8 ядер);
— 64 ГБ оперативной памяти;
— Компилятор Intel DPC++ Compiler 2023.0.0;
— OpenMP 5.0;
— Kokkos 4.0.1.

Запуски на GPU производились на видеокарте NVidia A100:

— 40 ГБ видеопамяти (пропускная способность 1555 ГБ/с);
— Частота ГП 1410 МГц;
— Версия Cuda 11.3;
— Kokkos 4.0.1.

На вход алгоритмам подается представленный в виде строки фрагмент незакодированной ДНК длиной $n \sim 5 \cdot 10^5$, а также длина искомого паттерна m ($4 \lesssim m \lesssim 1024$). В качестве тестовых данных выступают реальные фрагменты ДНК, взятые из открытых источников [20].

4.2. *Результаты экспериментов.* В процессе работы было разработано несколько версий кода: реализация рассматриваемых алгоритмов для CPU на C++ с использованием OpenMP и векторных вычислений; реализация для GPU на языке Cuda; кроссплатформенная реализация с использованием библиотеки Kokkos. На рис. 1, 2 представлены результаты замеров времени для каждой версии кода. Можно видеть, что производительность версии Kokkos отличается от производительности версии OpenMP на CPU или версии Cuda на GPU не более чем на 5–10 %.

На рис. 1 представлен анализ производительности на CPU. Можно заметить, что алгоритм Hash3 работает гораздо быстрее алгоритмов Рабина-Карпа и наивного алгоритма. Алгоритм Рабина-Карпа и Hash3 не понесли потери производительности при переходе на Kokkos. Наивный алгоритм при переходе на Kokkos стал работать медленнее примерно на 10 %. Это связано с тем, что в реализации OpenMP компилятор более эффективно задействовал векторные вычисления при посимвольном сравнении паттернов.

Все алгоритмы ускорились примерно в 14 раз при запуске на 16 потоках, что является достаточно хорошим результатом. Однако можно заметить, что алгоритм Hash3, в отличие от алгоритма Рабина-Карпа и наивного алгоритма, получил также существенный прирост производительности при задействовании гипертрединга. Наивный алгоритм и алгоритм Рабина-Карпа на 16 физических ядрах и 32 потоках демонстрируют ускорение 16–17 раз, в то время как ускорение алгоритма Hash3 составляет примерно 25 раз. Это связано с тем, что выделение большего количества потоков позволяет создавать большее число запросов к памяти, в результате чего шина ОЗУ начинает использоваться активнее.

Поскольку в алгоритме Hash3 практически не выполняются посимвольные сравнения и все основное время занимает подгрузка данных из памяти, то, как и ожидалось, при переходе на GPU алгоритм Рабина-Карпа и наивный алгоритм получили значительно большее ускорение (в 51 и 34 раза соответственно), чем Hash3 (примерно в 7 раз). Мож-

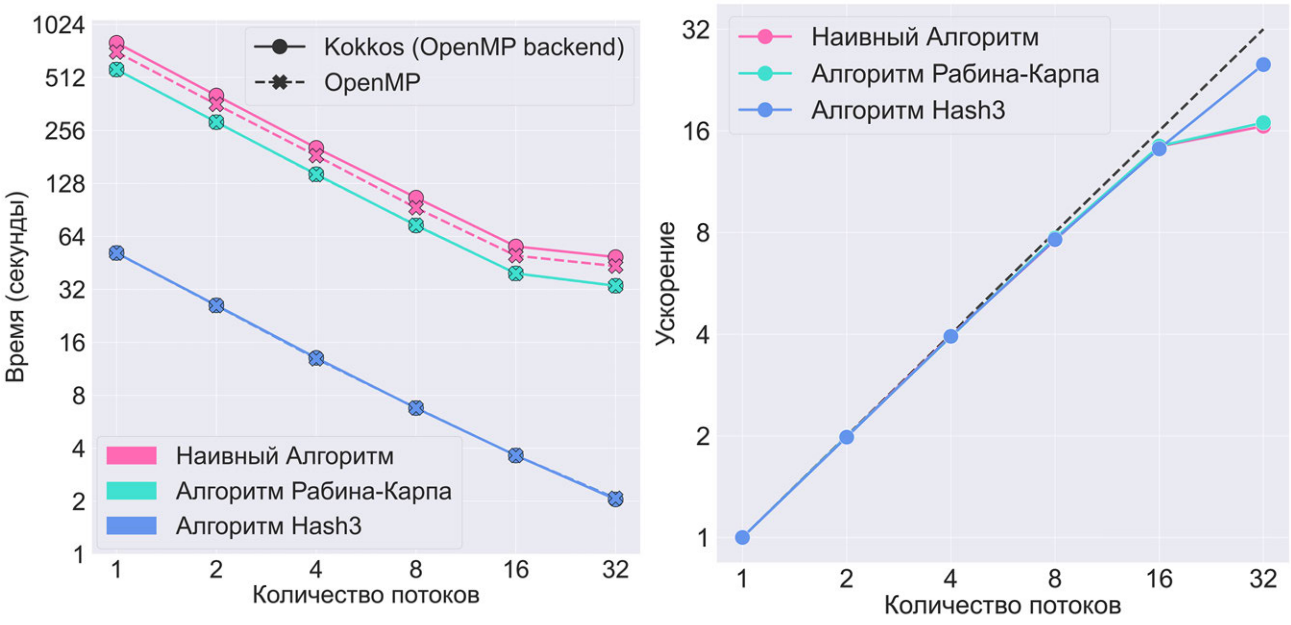

Рис. 1. Результаты тестирования алгоритмов на CPU (m=64). (а) Время работы в зависимости от числа потоков версий OpenMP и Kokkos с бэкендом OpenMP. (б) Ускорение для версии Kokkos. На числе потоков до 16 включительно один поток соответствует одному физическому ядру; 32 потока соответствуют 16 физическим ядрам с включенным гипертредингом

но заметить, что производительность алгоритма Hash3 зависит от длины искомой строки (рис. 2). На коротких паттернах не удается достичь отсечения большого количества поэлементных сравнений строк, поэтому на малых длинах паттернов алгоритм показывает меньшую эффективность, чем на больших длинах паттернов. В то же время производительность наивного алгоритма и алгоритма Рабина-Карпа не зависит от длины паттерна, поэтому они показывают более хорошую производительность на коротких паттернах, чем Hash3.

Результаты тестирования на GPU также показали, что реализации рассматриваемых алгоритмов, написанные с помощью Kokkos, не теряют в производительности относительно нативных решений. Более того, версия программы с использованием Kokkos иногда работает на GPU быстрее, чем базовая реализация того же кода на Cuda (рис. 2). Вероятно, Kokkos распределяет задачи между ядрами GPU более эффективно, чем это было сделано авторами вручную при написании кода на Cuda. Авторы уверены, что код на CUDA можно оптимизировать так, чтобы версия Cuda работала не медленнее, чем версия Kokkos, однако это требует отдельного исследования, и авторы не ставили себе подобной задачи. Тем не менее, заслуживает внимания тот факт, что Kokkos выполняет ряд оптимизаций для GPU самостоятельно без участия программиста.

Заключение. На основе задачи из биоинформатики по поиску наиболее часто встречающихся паттернов небольшой длины во фрагменте ДНК был разработан бенчмарк, демонстрирующий работу рассматриваемых в работе алгоритмов на CPU и GPU с использованием технологий OpenMP и Cuda, а также библиотеки Kokkos. Было продемонстрировано, что во многих случаях версия Kokkos работает не хуже, чем версия

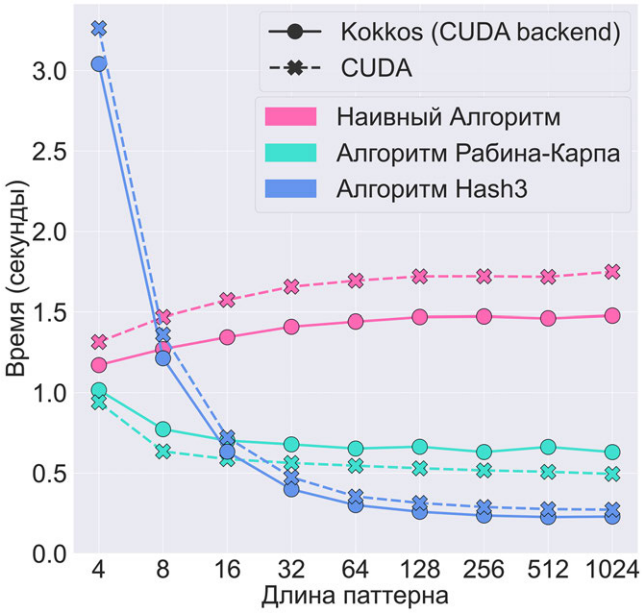

Рис. 2. Результаты тестирования алгоритмов на GPU. Зависимость времени работы алгоритмов от длины паттерна m для версий CUDA и Kokkos с бэкендом CUDA

кода при использовании OpenMP или Cuda, при этом версия с использованием Kokkos может быть запущена на устройствах различной архитектуры. В худшем случае потери производительности составили не более 10 %. Код бенчмарка находится в открытом доступе на платформе GitHub [19].

Список литературы

1. Gaster B., Howes L., Kaeli D. R., Mistry P., and Schaa D. Heterogeneous computing with openCL: revised openCL. Newnes, 2012.

2. Farber R. Parallel programming with OpenACC. Newnes, 2016.

3. Kokkos 3: Programming model extensions for the exascale era / Trott C. R., Damien LG, Arndt D., Ciesko J., Dang V., Ellingwood N., Gayatri R., Harvey E., Hollman D. S., Ibanez D., et al. // IEEE Transactions on Paralleland Distributed Systems. 2021. V. 33, N 4. P. 805–817.

4. Alpaka — an abstraction library for parallel kernel acceleration / Zenker E., Worpitz B., Widera R., Huebl A., Juckeland G., Knupfer A., Nagel W. E., and Bussmann M. // 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) / IEEE. 2016. P. 631–640.

5. Reinders J. et al. Data parallel C++: mastering DPC++ for programming of heterogeneous systems using C++ and SYCL. Springer Nature, 2021. P. 548.

6. Библиотека Kokkos. [Электрон. рес.]: `https://github.com/kokkos/kokkos`. Дата доступа: 10.01.2024.

7. Subirana J. A., Messeguer X. The most frequent short sequences in non-coding DNA // Nucleic acids research. 2010. V. 38, N 4. P. 1172–1181.

8. Faro S., Lecroq T. The exact online string matching problem: A review of the most recent results // ACM Computing Surveys(CSUR). 2013. V. 45, N 2. P. 1–42.

9. Exact string matching algorithms: survey, issues, and future research directions / Hakak S. I., Kamsin A., Shivakumara P., Gilkar G. A., Khan W. Z., and Imran M. // IEEE access. 2019. V. 7. P. 69614–69637.

10. Stephen G. A. String searching algorithms. World Scientific, 1994.

11. Al-Khamaiseh K., Alshagarin S. A survey of string matching algorithms // Int. J. Eng. Res. Appl. 2014. V. 4, N 7. P. 144–156.

12. Karp R. M., Rabin M. O. Efficient randomized patternmatching algorithms // IBM Journal of Research and Development. 1987. V. 31, N 2. P. 249–260.

13. Lecroq T. Fast exact string matching algorithms // Information Processing Letters. 2007. V. 102, N 6. P. 229–235.

14. Galil Z. A constant-time optimal parallel string-matching algorithm // Journal of the ACM (JACM). 1995. V. 42, N 4. P. 908–918.

15. Park J. H., George K. M. Efficient parallel hardware algorithms for string matching // Microprocessors and Microsystems. 1999. V. 23,N 3. P. 155–168.

16. Accelerating string matching using multi-threaded algorithm on GPU / Lin C. H., Tsai S. Y., Liu C. H., Chang S. C., and Shyu J. M. // 2010 IEEE Global Telecommunications Conference GLOBECOM 2010 / IEEE. 2010. P. 1–5.

17. Kouzinopoulos C. S., Michailidis P. D., Margaritis K. G. Multiple string matching on a GPU using cudas //Scalable Computing: Practice and Experience. 2015. V. 16, N 2. P. 121–138.

18. Козлов М. А., Панова Е. А., Мееров И. Б. Реализация поиска наиболее часто встречающихся последовательностей ДНК с использованием библиотеки Kokkos // Математическое моделирование и суперкомпьютерные технологии. Труды XXIII Международной конференции (Н. Новгород, 13–16 ноября 2023 г.) / Под ред. проф. Д. В. Баландина. Нижний Новгород: Изд-во Нижегородского госуниверситета, 2023. ISBN 978-5-91326-834-1. 2023. С. 73–78.

19. Открытый исходный код бенчмарка. [Электрон. рес.]: `https://github.com/Mishaizlesa/most_common_string_kokkos`. Дата доступа: 10.01.2024.

20. DNA Bank (National library of medicine). [Электрон. рес.]: `https://www.ncbi.nlm.nih.gov/genbank`. Дата доступа: 10.01.2024.

Козлов Михаил Андреевич — студент 3 курса кафедры Высокопроизводительных вычислений и системного программирования Института информационных технологий, математики и механики ННГУ им. Н. И. Лобачевского. E-mail: `misha11ru@gmail.com`.

Область научных интересов: высокопроизводительные вычисления, строковые алгоритмы.

Kozlov Mikhail — 3rd year student at the Department of HPC and System Programming at the Lobachevsky State University of Nizhny Novgorod. Research interests: HPC, string matching algorithms.

Панова Елена Анатольевна — аспирантка кафедры Высокопроизводительных вычислений и системного программирования Института информационных технологий, математики и механики ННГУ им. Н. И. Лобачевского. E-mail: `elena.panova@itmm.unn.ru`.

Ее исследовательские интересы включают разработку научного программного обеспечения, высокопроизводительные вычисления, анализ производительности, оптимизацию программ.

Elena Panova — Ph. D. student at the Department of HPC and System Programming at the Lobachevsky State University of Nizhny Novgorod. Her research interests include scientific software development, HPC, performance analysis, optimization.

Мееров Иосиф Борисович — канд. техн. наук, доцент, заведующий кафедрой Высокопроизводительных вычислений и системного программирования Института информационных технологий, математики и механики ННГУ

ГУ им. Н. И. Лобачевского. E-mail: meerov@vmk.unn.ru.

Научные интересы включают высокопроизводительные вычисления, анализ производительности и оптимизацию программ, вычислительные науки, разработку научного программного обеспечения.

Iosif Meyerov — Ph. D., head of Department of HPC and System programming at the Lobachevsky State University of Nizhny Novgorod. Research interests include HPC, performance analysis and optimization, computational science, scientific software development.

Дата поступления — 31.01.2024

Правила представления и подготовки рукописей для публикации в журнале „ПРОБЛЕМЫ ИНФОРМАТИКИ“

Общие требования.

Редакция принимает к рассмотрению статьи в электронном виде (*исходный файл в LaTeX* и файл PDF, с приложением оригиналов рисунков в формате тех программ, в которых они были сделаны, отдельными файлами).

Файлы, содержащие текст статьи, иллюстрации и дополнительные материалы, можно пересылать на электронный адрес редакции: problem-info@sscc.ru.

Принимаются файлы, архивированные архиваторами ZIP/7Z или RAR; применение самораспаковывающихся архивов не допускается.

При повторной отправке материалов, а также при внесении в исходный текст дополнений или исправлений необходимо сообщить об этом в редакцию в тексте электронного письма.

Направляя статью в редакцию журнала, автор (соавторы) на безвозмездной основе передает(ют) издателю на срок действия авторского права по действующему законодательству РФ исключительное право на использование статьи или отдельной ее части (в случае принятия редколлегией Журнала статьи к опубликованию) на территории всех государств, где авторские права в силу международных договоров Российской Федерации являются охраняемыми, в том числе следующие права: на воспроизведение, на распространение, на публичный показ, на доведение до всеобщего сведения, на перевод на иностранные языки и переработку (и исключительное право на использование переведенного и (или) переработанного произведения вышеуказанными способами), на предоставление всех вышеперечисленных прав другим лицам.

Журнал „Проблемы информатики“является некоммерческим изданием. Плата с авторов за публикацию статей не взимается.

К статье должны быть приложены:

— **разрешение на публикацию** от экспертного совета организации, в которой выполнена работа (для авторов из России);

— **оригинал рецензии;**

портретные фотографии авторов разрешением не менее 300 dpi.

— Блоки информации и на русском, и на английском языках просьба присылать отдельными файлами:

— **Название** статьи;

— **Инициалы и фамилии** авторов;

— **Места работы** авторов: полное наименование организации, почтовый индекс, город, страна;

— **Код(ы) классификации УДК**;

— **Аннотации**, содержащие краткую постановку задачи и описание метода решения: на русском языке объемом не более 1000 знаков, на английском языке расширенную, объемом от 4000 до 8000 знаков, что соответствует требованиям ВАК и Scopus.

— **Ключевые слова**;

— **Списки используемой литературы** в соответствии с ГОСТ Р 7.0.5—2008 (в английской версии необходимо выполнить **транслитерацию** неанглоязычных элементов списка литературы в соответствии с ГОСТ Р 7.0.34-2014) — составляются по ходу упоминания источников в тексте;

— **Краткие биографии (БИО)** авторов с указанием ключевых научных достижений (включая ученую степень, ученое звание — при наличии; основные области научных интересов и формулировку основных результатов, место работы, занимаемую должность, контактные данные — почтовый адрес с индексом, адрес электронной почты, контактный телефон).

Подготовка статьи.

1. Материал статьи должен быть изложен в следующей последовательности:

1.1. название статьи на английском языке;

1.2. инициалы и фамилия автора(ов) на английском языке;

1.3. место работы автора(ов) (на английском языке): полное наименование организации, индекс, город, страна;

1.4. англоязычная аннотация;

1.5. ключевые слова на английском языке;

1.6. references+транслитерация неанглоязычных элементов списка литературы;

1.7. название статьи на русском языке;

1.8. инициалы и фамилии и авторов;

1.9. место работы авторов: полное наименование организации, почтовый индекс, город, страна;

1.10. индекс УДК;

1.11. аннотация на русском языке;

1.12. ключевые слова (не более 8);

1.13. текст статьи;

1.14. список литературы, оформленный в соответствии с требованиями ГОСТ;

1.15. краткие биографии авторов на английском и русском языках с указанием ключевых научных достижений (ученую степень, ученое звание — при наличии; место работы, занимаемую должность, контактные данные — почтовый адрес с индексом, адрес электронной почты, контактный телефон, основные области научных интересов и формулировка основных результатов).

2. Требования к формулам:

— Нумерация формул сквозная, выносные формулы центрируются, номер выровнен по правому краю.

3. Требования к рисункам:

— Файлы с рисунками присылаются отдельно в формате программ, в которых они были выполнены: в формате MS Excel (для графиков и диаграмм), eps, pdf, png, tiff, bmp или jpeg (с максимальным качеством).

— Рисунки с подрисуночными подписями заверстываются в текст статьи.

— Тексты, являющиеся частью рисунка, выполняются шрифтом TimesNewRoman.

— Фотографии должны иметь разрешение не менее 300 dpi.

4. Дополнительные требования:

— В текст статьи необходимо включать ссылки на рисунки и таблицы, а также подрисуночные подписи и заголовки таблиц. Все буквенные обозначения, приведенные на рисунках, необходимо пояснить в основном тексте или в подрисуночных подписях.

— Сокращения слов не допускаются (кроме общепринятых).

— Векторные переменные обозначаются полужирным шрифтом без курсива.

— Таблицы не должны быть громоздкими. Значения физических величин в таблицах, на графиках и в тексте должны указываться в единицах измерения СИ.

— Графики, если их на рисунке несколько, а также отдельные детали на чертежах, узлы и линии на схемах следует обозначать цифрами, набранными курсивом.

— Нумеровать следует только те формулы и уравнения, на которые имеются ссылки в тексте, нумерация сквозная.

— Ссылки на источники в тексте заключаются в квадратные скобки.

— Иностранные источники приводятся на языке оригинала. Ссылки на неопубликованные работы не допускаются.

Все статьи, опубликованные в журнале «Проблемы информатики», доступны на сайте `https://elibrary.ru/title_about.asp?id=30275` и на сайте журнала `http://problem-info.sscc.ru` спустя год после опубликования.

Пример оформления статей можно посмотреть на сайте журнала `http://problem-info.sscc.ru`.