ISSN 1818-1015
eISSN 2313-5417

2022 Volume 29 No 1

MODELING AND ANALYSIS OF INFORMATION SYSTEMS

SCIENTIFIC JOURNAL

Start date of publication — 1999
Published quarterly

FOUNDER
P.G. Demidov Yaroslavl State University

EDITORIAL OFFICE
14 Sovetskaya str., Yaroslavl 150003, Russian Federation

Website: http://mais-journal.ru
E-mail: mais@uniyar.ac.ru
Phone: +7 (4852) 79-77-73

2022 Том 29 № 1

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ

НАУЧНЫЙ ЖУРНАЛ

Издается с 1999 года
Выходит 4 раза в год

УЧРЕДИТЕЛЬ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Ярославский государственный университет им. П. Г. Демидова»

РЕДАКЦИЯ
ул. Советская, 14, Ярославль, 150003, Российская Федерация
Website: http://mais-journal.ru
E-mail: mais@uniyar.ac.ru
Телефон: +7 (4852) 79-77-73

Свидетельство о регистрации СМИ ПИ № ФС 77–66186 от 20.06.2016 выдано Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций. Подписной индекс — 31907 в Объединенном каталоге «Пресса России». Технический редактор, компьютерная вёрстка – М. С. Каряева. Корректор английского текста – М. С. Комар. Подписано в печать 16.03.2021. Дата выхода в свет 31.03.2022. Формат 200×265 мм. Объем 72 с. Тираж 40 экз. Свободная цена. Заказ 008/022. Адрес типографии: ул. Советская, 14, оф. 109, Ярославль, 150003, Россия. Адрес издателя: Ярославский государственный университет им. П. Г. Демидова, ул. Советская, 14, Ярославль, 150003, Россия.
Содержание предназначено для детей старше 12 лет.

Contents

Содержание

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 29, NO. 1, 2022
journal homepage: www.mais-journal.ru

ALGORITHMS

The Algorithm of Angular Superresolution Using the Cholesky Decomposition and its Implementation Based on Parallel Computing Technology

S. E. Mishchenko[1], N. V. Shatskiy[2]

DOI: 10.18255/1818-1015-2022-1-6-19

[1]Rostov-on-Don Institute of Radiocommunications, 130 Nansena av., Rostov-on-Don 344010, Russia.
[2]Academician A. L. Mints Radiotechnical Institute, 8 March st., 10/1, Moscow 127083, Russia.

MSC2020: 78A50 78M50 68W10
Research article
Full text in Russian

Received February 3, 2022
After revision March 14, 2022
Accepted March 16, 2022

An algorithm of angular superresolution based on the Cholesky decomposition, which is a modification of the Capon algorithm, is proposed. It is shown that the proposed algorithm makes it possible to abandon the inversion of the covariance matrix of input signals. The proposed algorithm is compared with the Capon algorithm by the number of operations. It is established that the proposed algorithm, with a large dimension of the problem, provides some gain both when implemented on a single-threaded and multithreaded computer. Numerical estimates of the performance of the proposed and original algorithm using parallel computing technology CUDA NVidia are obtained. It is established that the proposed algorithm saves GPU computing resources and is able to solve the problem of constructing a spatial spectrum with an increase in the dimension of the covariance matrix of input signals by almost two times.

Keywords: digital array antennas; Capon super-resolution algorithm; Cholesky decomposition,bordering method; parallel computing.

INFORMATION ABOUT THE AUTHORS

Sergey E. Mishchenko, correspondence author — orcid.org/0000-0002-3210-1485. E-mail: mihome@yandex.ru
Leading researcher, Doctor of Engineering Sciences, Professor.

Nikolay V. Shatskiy — orcid.org/0000-0002-6365-7734. E-mail: shteiz@mail.ru
Head of the Integrated Department, PhD in Engineering Sciences, Associate Professor.

For citation: S. E. Mishchenko and N. V. Shatskiy, "The Algorithm of Angular Superresolution Using the Cholesky Decomposition and its Implementation Based on Parallel Computing Technology", *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 6-19, 2022.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 29, № 1, 2022
сайт журнала: www.mais-journal.ru

ALGORITHMS

Алгоритм углового сверхразрешения с использованием разложения Холецкого и его реализация на основе технологии параллельных вычислений

С. Е. Мищенко[1], Н. В. Шацкий[2]

DOI: 10.18255/1818-1015-2022-1-6-19

[1]ФГУП "Ростовский научно-исследовательский институт радиосвязи", ул. Нансена, д. 130, г. Ростов-на-Дону, 344010 Россия.
[2]Радиотехнический институт имени академика А. Л. Минца, ул. 8 Марта, д. 10, стр. 1, г. Москва, 127083 Россия.

УДК 621.396.677
Научная статья
Полный текст на русском языке

Получена 3 февраля 2022 г.
После доработки 14 марта 2022 г.
Принята к публикации 16 марта 2022 г.

Предложен алгоритм углового сверхразрешения на основе разложения Холецкого, представляющий собой модификацию алгоритма Кейпона. Показано, что предложенный алгоритм позволяет отказаться от обращения ковариационной матрицы входных сигналов. Проведено сравнение предложенного алгоритма с алгоритмом Кейпона по числу операций. Установлено, что предложенный алгоритм при большой размерности задачи обеспечивает некоторый выигрыш как при реализации на однопоточном, так и на многопоточном вычислителе. Получены численные оценки быстродействия предложенного и исходного алгоритма с использованием технологии параллельных вычислений CUDA NVIDIA. Установлено, что предложенный алгоритм обеспечивает экономию вычислительных ресурсов GPU и способен решать задачу построения пространственного спектра при увеличении размерности ковариационной матрицы входных сигналов почти в два раза.

Ключевые слова: цифровые антенные решетки; алгоритм сверхразрешения Кейпона; разложение Холецкого; метод окаймления; параллельные вычисления

ИНФОРМАЦИЯ ОБ АВТОРАХ

Сергей Евгеньевич Мищенко автор для корреспонденции	orcid.org/0000-0002-3210-1485. E-mail: mihome@yandex.ru ведущий научный сотрудник, доктор технических наук, профессор.
Николай Витальевич Шацкий	orcid.org/0000-0002-6365-7734. E-mail: shteiz@mail.ru начальник комплексного отдела, кандидат технических наук, доцент.

Для цитирования: S. E. Mishchenko and N. V. Shatskiy, "The Algorithm of Angular Superresolution Using the Cholesky Decomposition and its Implementation Based on Parallel Computing Technology", *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 6-19, 2022.

Введение

Алгоритмы пространственно-временной адаптивной обработки сигналов находят применение в радарах с цифровыми антенными решетками (ЦАР) для обнаружения источников радиоизлучения и определения их параметров: угловых координат — в пространственной области, радиальной скорости — в частотной области [1—4]. При построении пространственного спектра размерность задачи оптимальной обработки связана с числом приемных каналов ЦАР, в частотной области размерность задачи обусловлена числом зондирующих импульсов. В многоэлементных ЦАР основная сложность реализации алгоритмов адаптивной обработки сигналов связана с необходимостью формирования ковариационной матрицы входных сигналов (КМВС) большой размерности и обращения данной матрицы [5]. Эти операции желательно осуществлять в реальном масштабе времени.

Необходимость обработки больших объемов информации заставляет совершенствовать существующие алгоритмы пространственно-временной обработки и использовать технологии параллельных вычислений. В настоящее время в области параллельных вычислений широкое применение находит технология CUDA (Compute Unified Device Architecture) – программно-аппаратная архитектура параллельных вычислений. Применение параллельных вычислений позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы NVIDIA Corporation. Обычно выигрыш в производительности от использования технологий параллельной обработки оценивают применительно к простейшим алгоритмам векторных вычислений. Например, в работе [6] было показано, что использование технологии CUDA позволяет сократить время формирования КМВС почти в 30 раз для ЦАР, содержащей $M = 1024$ приемных каналов. Если же устройство обработки поддерживает технологию CUDA Direct Access [7], то формирование КМВС теоретически осуществимо в реальном масштабе времени, поскольку потребует для каждого нового временного среза выполнить последовательно M^2 комплексных умножений новых данных и M^2 комплексных сложений результатов умножения с элементами ранее сформированной матрицы. Последовательности операций умножения и сложения являются независимыми и могут выполняться в параллельных потоках вычислений.

В связи с этим дальнейшее повышение быстродействия алгоритмов пространственно-временной адаптивной обработки неразрывно связано с их реализацией на основе технологии параллельных вычислений.

Цель работы состоит в разработке алгоритма адаптивной обработки сигналов не требующего обращения КМВС, и оценке эффективности реализации данного алгоритма с использованием технологии параллельных вычислений.

1. Обоснование алгоритма и аналитическая оценка его сложности при использовании технологии параллельных вычислений

Пусть в процессе функционирования M-элементной ЦАР сформирована КМВС $\mathbf{R}$. В соответствии с алгоритмом Кейпона для формирования пространственного спектра заранее формируют набор векторов гипотез $\mathbf{V}(\theta_i, \varphi_i)$, где $i = 1, 2, ..., N_i$, после формирования КМВС выполняют ее обращение одним из известных алгоритмов и строят пространственный спектр по формуле

$$I(\theta_i, \varphi_i) = \frac{1}{\mathbf{V}(\theta_i, \varphi_i) \cdot \mathbf{R}^{-1} \cdot \mathbf{V}^H(\theta_i, \varphi_i)}, \tag{1}$$

Здесь символ H обозначает операцию сопряжения Эрмита, а вектор гипотеза $\mathbf{V}(\theta_i, \varphi_i)$ обеспечивает фазирование ЦАР в одном из N_i направлений (θ_i, φ_i).

Отсюда следует, что процесс обработки сигналов по формуле (1) состоит в выполнении двух этапов:

- обращение корреляционной матрицы $\mathbf{R}$;

- циклический перебор N_i векторов-гипотез, для каждой из которых сначала выполняют умножение вектора на матрицу, а затем, находят скалярное произведение результирующего вектора и вектора-гипотезы.

В известной литературе рассматриваются различные методы решения систем линейных уравнений и обращения матриц. В монографии [8] кроме алгоритмов линейной алгебры приводятся оценки числа операций, необходимые для их реализации. Одним из наиболее эффективных методов обращения матриц, позволяющим учесть эрмитовы свойства матрицы $\mathbf{R}$, является метод окаймления [8—10]. Вычислительной схеме метода окаймления соответствуют выражения:

$$\left(\mathbf{R}_1^{-1}\right)_{1,1} = \frac{R_{1,1}}{R_{1,1}R_{2,2} - \left|R_{1,2}^2\right|};\ \left(\mathbf{R}_1^{-1}\right)_{1,2} = -\frac{R_{2,1}}{R_{1,1}R_{2,2} - \left|R_{1,2}^2\right|}; \tag{2}$$

$$\left(\mathbf{R}_1^{-1}\right)_{2,1} = -\frac{R_{1,2}}{R_{1,1}R_{2,2} - \left|R_{1,2}^2\right|} = \left(\mathbf{R}_1^{-1}\right)^*_{1,2};\ \left(\mathbf{R}_1^{-1}\right)_{2,2} = \frac{R_{2,2}}{R_{1,1}R_{2,2} - \left|R_{1,2}^2\right|}; \tag{3}$$

$$\mathbf{R}_m = \begin{pmatrix} \mathbf{R}_{m-1} & \mathbf{u}_m \\ \mathbf{u}_m^H & R_{m,m} \end{pmatrix}; \tag{4}$$

$$\mathbf{R}_m^{-1} = \begin{pmatrix} \mathbf{R}_{m-1}^{-1} + \frac{1}{\rho_{m,m}}\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)^H & -\frac{1}{\rho_{m,m}}\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right) \\ -\frac{1}{\rho_{m,m}}\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)^H & \frac{1}{\rho_{m,m}} \end{pmatrix}; \tag{5}$$

$$\rho_{m,m} = R_{m,m} - \mathbf{u}_m^H\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right). \tag{6}$$

В выражении (3) символ $*$ обозначает операцию комплексного сопряжения.

На первом этапе при реализации метода окаймления выполняют обращение матрицы размером 2×2 с использованием выражений (2) и (3).

При реализации на многопоточном вычислителе метода окаймления число последовательных однотипных операций в этом случае составит: $\lceil 2/P \rceil + \lceil 3/P \rceil$ умножений; 1 сложение и 1 деление. Здесь и далее $\lceil x \rceil$ обозначает операцию округления x до ближайшего старшего целого, а P – число параллельных потоков вычислений.

После этого необходимо постепенно увеличивать ранг обратной матрицы до M, т. е. выполнить еще $M - 1$ циклов, в ходе каждого из которых выполняют операции в соответствии с (4)-(6). Формулы (4)-(6) включают операции: умножение матрицы на вектор, скалярное произведение.

Умножение матрицы размером $m \times m$ на вектор-столбец потребует вычислить: $\lceil m^2/P \rceil$ умножений и $\lceil m(m-1)/P \rceil$ сложений, а скалярное произведение – $\lceil m/P \rceil$ умножений и $\lceil (m-1)/P \rceil$ сложений (вычитаний).

Каждый m-ый из $M - 1$ циклов алгоритма обращения матрицы методом окаймления требует выполнить последовательность действий, приведенных в таблице 1. С учетом данных таблицы 1

Table 1. The computing difficulty by (4)-(6) for preassigned m

Описание действия	Число операций ($m = 3, ..., M$)
произведение матрицы на вектор $\mathbf{R}_{m-1}^{-1}\mathbf{u}_m$	умножений $\lceil (m-1)^2/P \rceil$; сложений $\lceil (m-1)(m-2)/P \rceil$
скалярное произведение векторов $\mathbf{u}_m^H\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)$	умножений $\lceil (m-1)/P \rceil$; сложений $\lceil (m-2)/P \rceil$
диагональный элемент $1/\rho_{m,m}$	1 сложение; 1 деление
элементы матрицы $\frac{1}{\rho_{m,m}}\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)^H$	умножений $2\lceil (m-1)^2/P \rceil$
умножение $\frac{1}{\rho_{m,m}}\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)$	умножений $\lceil (m-1)/P \rceil$

Таблица 1. Сложность вычислений согласно (4)-(6) при заданном m

получим следующие оценки сложности алгоритма обращения матрицы методом окаймления на многопоточном вычислителе. Число последовательно выполняемых операций умножения $O_{\times}^{BM}(M,P)$, сложения $O_{+}^{BM}(M,P)$ и деления $O_{/}^{BM}(M,P)$ равно:

$$O_{\times}^{BM}(M,P) = \lceil 2/P \rceil + \lceil 3/P \rceil \sum_{m=1}^{M} \left(3\lceil (m-1)^2/P \rceil + 2\lceil (m-1)/P \rceil \right); \tag{7}$$

$$O_{+}^{BM}(M,P) = 1 + \sum_{m=1}^{M} \left(\lceil (m-1)(m-1)/P \rceil + \lceil (m-2)/P \rceil + 1 + \lceil (m-1)^2/P \rceil \right); \tag{8}$$

$$O_{/}^{BM}(M) = M - 2. \tag{9}$$

В монографии [8] приведены следующие оценки вычислительных затрат при реализации метода окаймления для обращения матрицы:

- $0,5\left(M^3 + M^2 - 2M\right)$ комплексных умножений;
- $0,5\left(M^3 - M^2\right)$ комплексных сложений;
- M комплексных делений.

Сопоставление выражений (7) и (8) с соответствующими оценками из [8] позволяет заключить, что выигрыш от распараллеливания может быть очень существенным по операциям комплексного умножения и сложения.

Дальнейшая реализация алгоритма Кейпона состоит в том, что многократно выполняют умножение вектора на обратную матрицу. Это соответствует:

- $N_i M^2$ комплексных умножений;
- $N_i M\,(M-1)$ комплексных сложений.

После этого рассчитывают скалярное произведение двух векторов, первый из которых получен в результате умножения обратной матрицы на вектор-гипотезу и на исходный вектор гипотезу. На данном этапе имеем:

- $N_i M$ комплексных умножений;
- $N_i\,(M-1)$ комплексных сложений.

При переходе к многопоточному вычислителю суммарные затраты на реализацию алгоритма Кейпона составляют:

$$O_{\times}^{Capon}(M,P,N_i) = O_{\times}^{BM}(M,P) + N_i\lceil M/P \rceil^2 P + N_i\lceil M/P \rceil; \tag{10}$$

$$O_{+}^{Capon}(M,P,N_i) = O_{+}^{BM}(M,P) + N_i\lceil M(M-1)/P \rceil + N_i\lceil (M-1)/P \rceil; \tag{11}$$

$$O_{/}^{Capon}(M) = O_{/}^{BM}(M) = M - 2. \tag{12}$$

Можно также получить суммарную оценку сложности, если учесть, что операции действительной арифметики при комплексном умножении распараллеливаются и сводятся к одному умножению и сложению. Операцию деления будем рассматривать как совокупность трех операций: двух умножений и сложения. В результате получим:

$$O_{\Sigma}^{Capon}(M,P,N_i) = 2O_{\times}^{Capon}(M,P,N_i) + O_{+}^{Capon}(M,P,N_i) + 3O_{/}^{Capon}(M). \tag{13}$$

В качестве альтернативного варианта реализации алгоритма Кейпона предложим алгоритм, в котором пространственный спектр представляется выражением

$$I(\theta_i,\varphi_i) = \frac{1}{\mathbf{V}(\theta_i,\varphi_i)\cdot\mathbf{X}(\theta_i,\varphi_i)}, \tag{14}$$

где вектор-столбец $\mathbf{X}(\theta_i, \varphi_i)$ для каждой из гипотез соответствует решению системы уравнений

$$\mathbf{R} \cdot \mathbf{X}(\theta_i, \varphi_i) = \mathbf{V}^H(\theta_i, \varphi_i). \tag{15}$$

Существуют различные методы решения систем уравнений без обращения матрицы коэффициентов при неизвестных. Применим для решения системы уравнений (15) разложение Холецкого (метод квадратного корня в [8]), в соответствии с которым

$$\mathbf{R} = \mathbf{L} \cdot \mathbf{L}^H, \tag{16}$$

где $\mathbf{L}$ – нижнетреугольная матрица, элементы которой определяют по формулам [8]:

$$L_{1,1} = \sqrt{R_{1,1}};\ L_{m,1} = \frac{R_{m,1}}{L_{1,1}}; \tag{17}$$

$$L_{m,m} = \sqrt{R_{m,m} - \sum_{p=1}^{m-1} L^*_{m,p} L_{m,p}};\ m = 2, ..., M; \tag{18}$$

$$L_{m,n} = \frac{1}{L_{n,n}} \left(R_{m,n} - \sum_{p=1}^{n-1} L^*_{n,p} L_{m,p} \right);\ n = 2, ..., M-1;\ m = n+1, ..., M. \tag{19}$$

Для определения искомого вектора $\mathbf{X}(\theta_i, \varphi_i)$ требуется решить две системы уравнений методом исключения [11]:

$$\mathbf{L}^H \cdot \mathbf{X}(\theta_i, \varphi_i) = \mathbf{V}'(\theta_i, \varphi_i);\ \mathbf{L} \cdot \mathbf{V}'(\theta_i, \varphi_i) = \mathbf{V}^H(\theta_i, \varphi_i). \tag{20}$$

Отсюда следует, что алгоритм сверхразрешения, основанный на разложении Холецкого также состоит из двух этапов, первый из которых состоит в формировании нижнетреугольной матрицы $\mathbf{L}$ по формулам (17)–(19), а второй состоит в том, что для различных наборов векторов-гипотез решают системы уравнений (20), а затем вычисляют скалярное произведение текущего вектора-гипотезы и вектора, соответствующего решению систем линейных уравнений (15).

При реализации данного алгоритма система линейных уравнений решается с использованием меньшего числа операций, однако отсутствие обратной матрицы приводит к тому, что разложение Холецкого необходимо выполнить один раз, а решение систем уравнений (20), в которых матрица коэффициентов при неизвестных является треугольной – многократно.

В соответствии с оценками [8] для решения системы линейных уравнений методом квадратного корня необходимо выполнить:

- $\frac{1}{6}\left(M^3 + 9M^2 + 2M\right)$ комплексных умножений;
- $\frac{1}{6}\left(M^3 + 6M^2 - 7M\right)$ комплексных сложений;
- M делений;
- M операций извлечения квадратного корня.

В настоящее время операцию извлечения квадратного корня выполняют с использованием итерационного метода Ньютона [12—14]. При этом существуют эффективные в вычислительном отношении процедуры, которые позволяют ограничить число итераций за счет очень близкой начальной оценки [13]. Примем, что для отыскания квадратного корня методом Ньютона достаточно 4 итерации. Поскольку при вычислении квадратного корня оперируют только с действительными числами, то можно считать, что одна итерация соизмерима с одним произведением двух комплексных чисел, а наличие операции извлечения корня по вычислительным затратам требует только $4M$ операций комплексного умножения, которые не могут быть распараллелены.

Число операций, необходимое для решения двух систем уравнений с треугольными матрицами коэффициентов несложно оценить на простых численных примерах, из которых следует, что они

определяются выражениями для суммы членов арифметической прогрессии от 1 до M с шагом 1. Для двух систем уравнений число умножений и сложений будет одинаковым и равно $M(M-1)$. Эти операции следует исключить из приведенных выше оценок, для того, чтобы оценить сложность выполнения разложения Холецкого.

Рассмотрим теперь схему реализации разложения Холецкого на многопоточном вычислителе. Реализацию алгоритма начинают с заполнения первого диагонального элемента и внедиагональных элементов в первом столбце.

Для этого шага необходимо вычислить квадратный корень, выполнить одну операцию деления и $\lceil (M-1)/P \rceil$ операций умножения.

Далее последующие этапы состоят в заполнении элементов столбцов, в которых число внедиагональных элементов уменьшается от $M-2$ до 0. Описания соответствующих шагов и сложность их оценки приведены в таблице 2.

Table 2. The realization difficulty of the Cholesky decomposition for one iteration of m on a multithreaded computer

Таблица 2. Сложность реализации разложения Холецкого для одной итерации m на многопоточном вычислителе

Описание действия	Число операций
вычисление диагонального элемента m-го столбца $(1 < m \le M)$	умножений $\lceil (m-1)2/P \rceil$; сложений $2\lceil (m-1)/P \rceil$; 1 квадратный корень
вычисление внедиагональных элементов	умножений $2\lceil (m-1)/P \rceil$; сложений $2\lceil (m-2)/P \rceil$; 1 деление

На основании таблицы 2 и операций для заполнения первого этапа получим число последовательно выполняемых операций умножения:

$$O_{\times}^{Cholesky}(M,P) = \lceil (M-1)/P \rceil + 3\sum_{m=2}^{M-1} \lceil (m-1)/P \rceil; \tag{21}$$

сложения:

$$O_{+}^{Cholesky}(M,P) = 2\sum_{m=2}^{M-1} \left(\lceil (m-1)/P \rceil + \lceil (m-2)/P \rceil \right); \tag{22}$$

$O_{/}^{Cholesky}(M) = M$ делений; $O_{\sqrt{}}^{Cholesky}(M) = M$ операций извлечения квадратного корня. Полученные соотношения демонстрируют не только существенное сокращение числа операций при использовании параллельных вычислений при реализации разложения Холецкого по сравнению с методом окаймления. Как видно, элементы треугольной матрицы, определяемой при реализации алгоритма могут записываться на месте элементов исходной матрицы. Это позволяет сохранить ресурсы памяти вычислителя для работы. Этот вывод согласуется и с результатами анализа алгоритма в [8].

Теперь оценим вычислительные затраты, необходимые при многократном вычислении значений пространственного спектра в различных направлениях для предлагаемого алгоритма.

Алгоритм сверхразрешения, основанный на разложении Холецкого, требует решения систем уравнений (20) и вычисления скалярного произведения векторов.

Каждая из систем уравнений (20) решается методом исключения. Этот метод несложно реализовать на однопоточном вычислителе. Оценка его сложности в случае многопоточного вычислителя может быть выполнена следующим образом.

Каждая из треугольных матриц при решении систем уравнений (20) содержит $0,5\left(\lceil M/P \rceil^2 - \lceil M/P \rceil\right)$ полностью заполненных блоков коэффициентов и $\lceil M/P \rceil$ блоков, которые

представляют собой треугольные матрицы. При этом алгоритм решения системы уравнений методом исключения при распараллеливании можно разбить на следующие этапы:

- решение системы P уравнений с треугольной матрицей коэффициентов методом исключения;
- переход к следующим P уравнениям, приведение их к виду системы уравнений с треугольной матрицей коэффициентов.

Выполнение первого этапа включает P операций деления. Всего для всей исходной системы уравнений потребуется M операций деления. Эти операции не могут выполняться параллельно.

Алгоритм решения системы P уравнений с треугольными коэффициентами начинается с вычисления первого компонента вектора решения. Для его вычисления необходима одна операция деления. Далее необходимо откорректировать последующие $P-1$ уравнений, исключив из них первый столбец коэффициентов. При выполнении этих операций одновременно для всех уравнений потребуется одна операция умножения и одна операция сложения. Продолжая эти действия можно заключить, что для решения рассматриваемой системы P уравнений достаточно выполнить $P-1$ операций комплексного умножения и столько же операций сложения (вычитания). Поскольку общее число блоков подобных систем уравнений равно $\lceil M/P \rceil$, то число умножений и сложений будет равно $\lceil M/P \rceil (P-1)$ операций соответственно.

Приведение следующих P строк исходной системы уравнений к системе уравнений с треугольной матрицей коэффициентов требует вычисления P^2 умножений и P^2 вычитаний (сложений). При выполнении параллельных вычислений эти оценки следует уменьшить в P раз, т.е. число умножений и сложений станет равным P.

Для произвольного по счету блока уравнений из P строк число удаляемых коэффициентов для приведения матрицы при неизвестных к треугольному виду будет изменяться дискретно, а общее число таких коррекций будет соответствовать числу блоков заполненных коэффициентов. Отсюда следует, что выполнение данного этапа для всей системы уравнений потребует $0,5\left(\lceil M/P \rceil^2 - \lceil M/P \rceil\right) P$ последовательностей операций комплексного умножения и столько же операций сложения.

Отсюда следует, что число операций с комплексными числами для реализации предлагаемого алгоритма на многопоточном вычислителе равно:

$$O_{\times}^{Sgstd}(M,P,N_i) = O_{\times}^{Cholesky}(M,P) + N_i\left(\lceil M/P \rceil^2 - \lceil M/P \rceil\right) P + N_i \lceil M/P \rceil (P-1); \tag{23}$$

$$O_{+}^{Sgstd}(M,P,N_i) = O_{+}^{Cholesky}(M,P) + N_i\left(\lceil M/P \rceil^2 - \lceil M/P \rceil\right) P + N_i \lceil M/P \rceil (P-1); \tag{24}$$

$$O_{/}^{Sgstd}(M,N_i) = O_{/}^{Cholesky}(M) + N_i M;\ O_{\sqrt{}}^{Sgstd}(M) = O_{\sqrt{}}^{Cholesky}(M) = M. \tag{25}$$

Аналогично выражению (13) запишем суммарную оценку числа операций при реализации предложенного алгоритма в виде

$$O_{\Sigma}^{Sgstd}(M,P,N_i) = 2O_{\times}^{Sgstd}(M,P,N_i) + O_{+}^{Sgstd}(M,P,N_i) + 3O_{/}^{Sgstd}(M,P,N_i) + 4M. \tag{26}$$

Таким образом, предложенный алгоритм построения пространственного спектра отличается от алгоритма Кейпона тем, что не требует операции обращения КМВС. Это приводит к тому, что при построении пространственного спектра в каждом направлении приходится решать две системы линейных уравнений с треугольными коэффициентами. Такое усложнение алгоритма может приводить к существенному замедлению решения задачи на однопоточном вычислителе. Однако полученные аналитические выражения показывают, что использование технологии параллельных вычислений может существенно изменить вычислительную эффективность предложенного алгоритма сверхразрешения.

В связи с этим ниже были получены оценки сложности классического алгоритма Кейпона и предлагаемого алгоритма углового сверхразрешения при реализации на многопоточном вычислителе.

2. Результаты численных исследований

На рисунке 1 представлены результаты сопоставления сложности алгоритма Кейпона, при реализации которого используется обращение матрицы, и предложенного алгоритма, основанного на использовании разложения Холецкого. Данные результаты получены при $N_i = 1801$. Две верхние кривые соответствуют зависимостям сложности алгоритмов при их реализации на однопоточном вычислителе ($P = 1$). Данные зависимости демонстрируют сопоставимость сложности обоих алгоритмов. Нижние кривые получены при $P = 32$. Из их сравнения следует, что при увеличении числа потоков с ростом размерности задачи возрастает эффективность предложенного алгоритма сверхразрешения. При $M = 4000$ сложность предложенного алгоритма, реализованного на многопоточном вычислителе при $P = 32$, по сравнению с оценкой для $P = 1$ уменьшилась в 43 раза. Число операций при реализации классического алгоритма Кейпона с увеличением числа потоков до 32 сократилось в 22 раза.

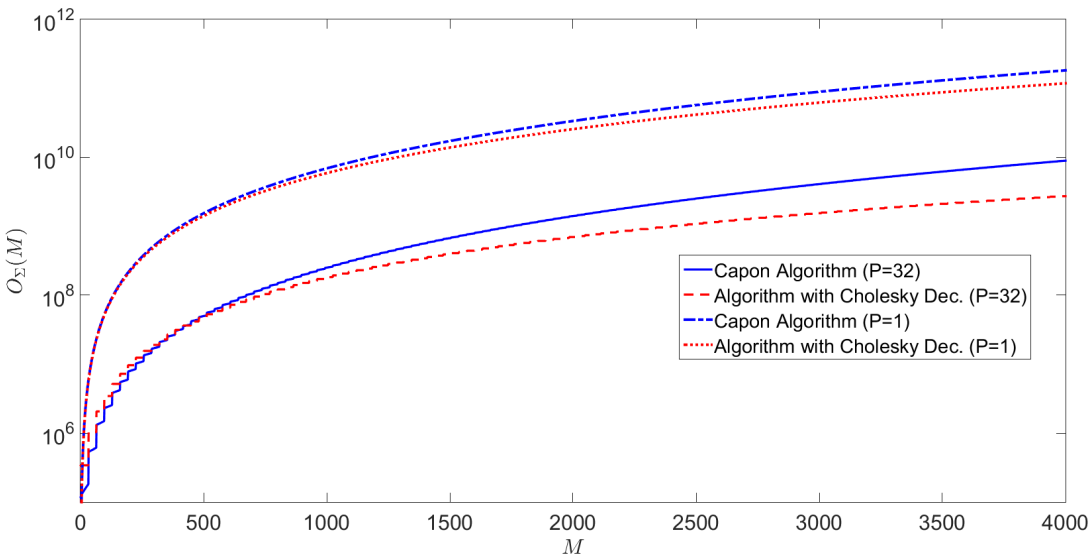

Fig. 1. Theoretical comparing of difficulty of suggested algorithm and Capon's algorithm

Рис. 1. Теоретическое сравнение сложности предложенного алгоритма и алгоритма Кейпона

Для подтверждения полученных аналитически результатов была написана программа на языке Python, содержащая процедуры: разложения Холецкого, обращения эрмитовой матрицы по методу окаймления, умножения комплексного вектора на эрмитову матрицу и скалярного произведения векторов. При реализации алгоритмов сверхразрешения при вычислениях на CPU использовались функции линейной алгебры модуля Numpy Python.

Ниже представлен фрагмент кода программы, реализующей рассматриваемые алгоритмы сверхразрешения при условии, что КМВС уже сформирована и загружена в память CPU. В тексте фрагмента программы не приведен код для формирования КМВС и векторов-гипотез.

```
import numpy as np                  # Инициализация функций модуля Numpy Python
tria(v1,L):                         # Процедура решения систем уравнений (20)
    M = v1.size                     # v1 – Вектор-гипотеза, L – Нижнетреугольная матрица
    v1[0] = v1[0]/L[0,0]
    for i in range(1,M):
        tmp = 0
```

```
        for j in range(i):
            tmp += L[i,j]*v1[j]
        v1[i] = (v1[i]-tmp)/L[i,i]          # Результат решения первой системы уравнений из (20)
    v1[M-1] = v1[M-1]/L[M-1,M-1]
    for i in range(M-2,-1,-1):
        tmp = 0
        for j in range(i+1,M):
            tmp += L[j,i].conjugate()*v1[j]
        v1[i] = (v1[i]-tmp)/L[i,i]
    return v1                               # Возвращает вектор решения обеих систем уравнений
V1 = np.dot(np.linalg.inv(R),V)             # Умножение обратной матрицы на вектор-гипотезу V
ICPU=1/abs(np.dot(V.transpose().conjugate(),V1))        # Результат расчета в соответствии с (1)
L = np.linalg.cholesky(R)                               # Разложение Холецкого КМВС R
V1 = tria(V,L)                                          # Решение систем уравнений (20)
ICPU-Chol = 1/abs(np.dot(V1.transpose().conjugate(),V)) # Результат расчета в соответствии с (14)
```

Параллельные вычисления осуществлялись с использованием функций CUDA модуля Numba Python. Далее представлен фрагмент текста программы, в котором оформлены процедуры, реализующие основные этапы рассматриваемых алгоритмов сверхразрешения с использованием технологии CUDA. Распределение ресурсов вычислителя для реализации параллельных вычислений обеспечивается на этапе компиляции процедур с декоратором @cuda.jit.

```
from numba import cuda                          # Инициализация функций модуля Numba Python
@cuda.jit('void(complex128[:,:],complex128[:,:])')  # Декорирование функции
def cuda_cholesky(R, L):                        # Процедура разложения Холецкого матрицы R
    L[0,0] = cmath.sqrt(abs(R[0,0]))            # Первый диагональный элемент
    M = L[0,:].size
    abs(cmath.sqrt(L.size))
    for i in range(1,M):
        L[i,0] = R[i,0]/abs(L[0,0])             # Заполнение первого столбца
    for i in range(1,M):
        tmp = 0
        for j2 in range(i):
            tmp += L[i,j2]*L[i,j2].conjugate()
            L[i,i] = cmath.sqrt(R[i,i]-tmp)     # Заполнение диагонального элемента
        for j2 in range(i+1,M):
            tmp = 0
            for j1 in range(i):
            tmp += L[j2,j1]*L[i,j1].conjugate()
        L[j2,i] = (R[j2,i] - tmp)/L[i,i]        # Заполнение внедиагональных элементов

@cuda.jit('void(complex128[:],complex128[:],complex128[:])')
def cuda_vect_mul(v1,v2,C):                     # Процедура скалярного произведения векторов
    M = v1.size
    for i in range(M):
        C[0] += v1[i]*v2[i].conjugate()

@cuda.jit('void(complex128[:],complex128[:,:],complex128[:])')
```

```
def mul_VxM(V,L,V1):                          # Процедура умножения матрицы на вектор
    M = V1.size
    for i in range(M):
        V1[i] = 0
        for j in range(M):
            V1[i] += V[j]*L[j,i]

@cuda.jit('void(complex128[:,:],complex128[:,:],complex128[:],complex128[:],complex128[:])')
def okaimlen(R,L,V1,V2):              # Процедура обращения матрицы методом окаймления
    tmp = R[0,0]*R[1,1]-abs(R[0,1])**2
    L[0,0] = R[1,1]/tmp; L[0,1] = -R[1,0].conjugate()/tmp
    L[1,0] = L[0,1].conjugate(); L[1,1] = R[0,0]/tmp; M = V2.size
    for i in range(2,M):
        for j in range(i):
            V2[j] = R[i,j]
        for i1 in range(i):
            V1[i] = 0
            for j in range(i):
                V1[i1] += V2[j]*L[j,i1]
        tmp = 0
        for i1 in range(i):
            tmp += V1[i1]*V2[i1].conjugate()
        L[i,i] = 1/(R[i,i] - tmp)
        for j in range(i):
            L[j,i] = -(V1[j]*L[i,i]).conjugate(); L[i,j] = L[j,i].conjugate()
            for k in range(i):
                L[j,k] = L[j,k] + L[i,i]*V1[j].conjugate()*V1[k]
```

Помимо приведенных процедур также использовалась процедура cuda_tria(v1,L), которая отличалась от процедуры tria(v1,L) только использованием декоратора:
@cuda.jit('void(complex128[:],complex128[:,:])').

Наконец, ниже приведен фрагмент текста программы, в которой рассматривалась реализация алгоритма Кейпона с использованием технологии CUDA и реализованных процедур.

```
device = cuda.get_current_device()          # Инициализация видеоадаптера для вычислений
tpb = [16,16]; Nmax = 108
for N in range(3,Nmax):                     # Индекс N определяет размерность КМВС R
    bpg = int(np.ceil(N/16))
    d_R = cuda.to_device(R)                 # Загрузка данных в память видеоадаптера
    d_V1 = cuda.to_device(V1); d_V2 = cuda.to_device(V2)
    d_V0 = cuda.to_device(V0); d_C = cuda.to_device(C)
    d_L = cuda.to_device(L)
    okaimlen[bpg, tpb](d_R,d_L,d_V1,d_V2)   # Обращение КМВС методом окаймления
    mul_VxM[bpg, tpb](d_V0,d_L,d_V1)        # Умножение обратной матрицы на вектор-гипотезу
    cuda_vect_mul[bpg, tpb](d_V0,d_V1,C)    # Скалярное произведение векторов
    C = d_C.copy_to_host()                  # Извлечение результата из памяти видеоадаптера
```

При реализации предложенного алгоритма в тексте программы вместо строк, в которых выполнялось обращение матрицы и умножение обратной матрицы на вектор исполльзовались команды

```
cuda_cholesky[bpg, tpb](d_R,d_L) # Разложение Холецкого
cuda_tria[bpg, tpb](d_V1,d_L)    # Решение систем уравнений (20)
```

Приведенные фрагменты программ не содержат функций, связанных с оценкой временных затрат на выполнение тех или иных этапов алгоритмов и выводом результатов этих оценок. Кроме того, в текстах программ отсутствует код, обеспечивающий формирование КМВС **R** и векторов гипотез **V**. При проведении исследований учитывалось, что используемый видеоадаптер имеет ограниченный объем памяти 4 Гб, что не позволяло загружать весь массив векторов-гипотез в память GPU. В связи с этим временные оценки соответствовали одному вектору-гипотезе, а затем умножались на заданное значение N_i. При этом формула для оценки быстродействия имела вид

$$T(M) = T_1(M) + N_i \cdot T_2(M), \tag{27}$$

где $T_1(M)$ – время на обращение матрицы **R** для исходного алгоритма или время выполнения разложения Холецкого этой же матрицы для предложенного алгоритма; $T_2(M)$ – время на проверку одного вектора-гипотезы.

При проведении численных исследований использовалась ЭВМ со следующими характеристиками:

- процессор – Intel® Core™ i7-2600K CPU @ 3.40 GHz 3.40 GHz;
- объем ОЗУ– 8 ГБ;
- видеоадаптер – NVIDIA GeForce GTX 550 Ti 4ГБ ОЗУ.

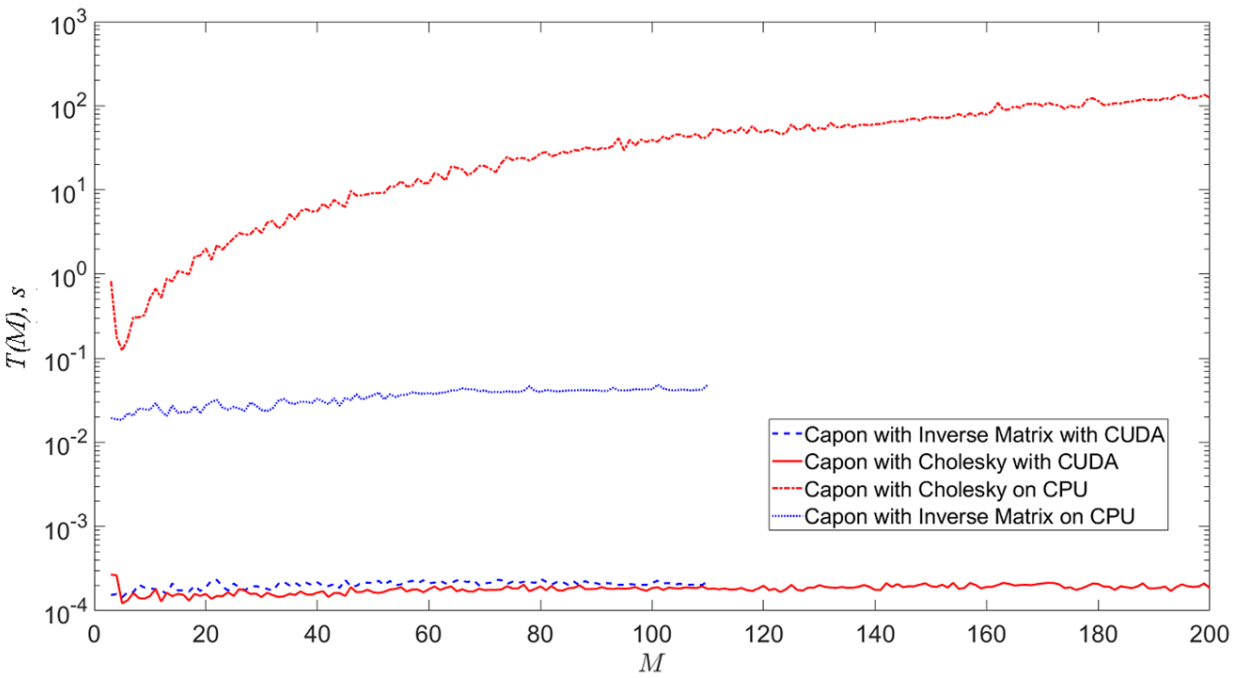

Fig. 2. The performance of the proposed algorithm and the Capon algorithm when implemented on CPU and GPU

Рис. 2. Быстродействие предложенного алгоритма и алгоритма Кейпона при реализации на CPU и GPU

Полученные результаты приведены на рисунке 2. Верхняя кривая отражает оценку быстродействия предложенного алгоритма без использования вычислений на GPU. Как видно, эта зависимость монотонно возрастает, что обусловлено необходимостью многократного повторения решения системы уравнений (20) с использованием процедуры tria. Код этой процедуры не претендует на оптимальность по быстродействию и не относится к библиотеке стандартных функций модуля Numpy Python. Вторая сверху кривая получена при реализации исходного алгоритма Кейпона с использованием библиотечных функций. Две нижние кривые получены при реализации алгоритмов с использованием вычислений на GPU. При этом штриховая кривая получена при реализации исходного алгоритма Кейпона, а сплошная нижняя кривая – с использованием предложенного алгоритма. Видно, что предложенный алгоритм ни в чем не уступает, а по эффективности использования памяти видеоадаптера превосходит исходный алгоритм Кейпона.

Оценки быстродействия алгоритмов сверхразрешения, использующих вычисления на GPU не учитывают временные затраты на загрузку данных для вычислений в память видеоадаптера. Максимальное время загрузки данных при построении пространственного спектра не превышало 40 мс.

Заключение

Предложенный алгоритм углового сверхразрешения, востребованный при обработке сигналов в цифровых антенных решетках, отличается от известного алгоритма Кейпона исключением операции обращения матрицы и использованием разложения Холецкого.

Показано, что предложенный алгоритм при построении пеленгационного рельефа требует многократного решения систем уравнений вида (20). Это приводит к тому, что данный алгоритм при реализации на CPU существенно уступает обычному алгоритму Кейпона.

Установлено, что при использовании параллельных вычислений предложенный алгоритм не уступает исходному алгоритму Кейпона с обращением КМВС по быстродействию, а также при реализации более эффективно использует память GPU. При этом допустимая размерность задачи увеличивается почти в два раза.

References

[1] R. Klemm, *Principles of Space-Time Adaptive Processing*. London: IEE, 2002.

[2] M. Parker, *Radar Basics-Part 4: Space-time adaptive processing*, 2011. [Online]. Available: http://www.eetimes.com/design/programmable-logic/4217308/Radar-Basics-Part-4-Space-time-adaptive-processing.

[3] W. Bürger, *Space-Time Adaptive Processing: Algorithms*, 2006. [Online]. Available: http://ftp.rta.nato.int/public//PubFullText/RTO/EN/RTO-EN-SET-086///EN-SET-086-07.pdf.

[4] S. Nefedov, I. Kruchkov, M. Noniashvili, G. Lesnikov, and N. Soloviev, "A Review of the Signal Space-Time Adaptive Processing (STAP) Basic Techniques in Space-Based Radars with Synthetic Aperture)", *Vestnik MGTU im N.E. Baumana. Ser. "Priborostroenie"*, no. 8, pp. 251–258, 2012.

[5] M. Ratynskij, *Adaptation and Superresolution in Array Antennas*. Moscow: Radio i svyaz, 2003.

[6] S. Tulenev, A. Savinkov, and V. Vereitin, "Application of CUDA Parallel Programming Technology to Increase the Speed Of Calculation of Superresolution Algorithms in Direction Finding", *Vestnik Voronezhskogo instituta MVD Rossii*, no. 2, pp. 196–203, 2021.

[7] *Developing a Linux Kernel Module using GPUDirect RDMA*. [Online]. Available: https://docs.nvidia.com/cuda/gpudirect-rdma/index.html.

[8] V. Voevodin, *Numerical Methods of Algebra*. Moscow: Nauka, 1966.

[9] A. Novikov, D. Gabriel'yan, E. Novikova, and N. Shatskiy, *Device to Covariance Matrix of Interference Sygnals Inversion*, Rus. Patent 2 562 389, Sept. 2015.

[10] M. Zvezdina, O. Komova, N. Shatskiy, and A. Shokov, "Hermitian matrix inversion algorithm", *Vestnik Donskogo gosudarstvennogo universiteta*, no. 2(81), pp. 78–84, 2015.

[11] F. Gantmacher, *The Theory of Matrices*. New York: Chelsea Publishing Company, 1959.

[12] T. Shoup, *A Practical Guide to Computer Methods for Engineers*. New York: Prentice-Hall, Inc., Englewood Cliffs, 1979.

[13] C. Lomont, "Fast Inverse Square Root", vol. 32, 2003. [Online]. Available: http://www.lomont.org/papers/2003/InvSqrt.pdf.

[14] N. Shilov, D. Kondratyev, I. Anureev, E. Bodin, and A. Promsky, "Platform-independent Specification and Verification of the Standard Mathematical Square Root Function", *Modeling and Analysis of Information Systems*, vol. 25, no. 6, pp. 637–666, 2018. DOI: 10.18255/1818-1015-2018-6-637-666.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 29, NO. 1, 2022
journal homepage: www.mais-journal.ru

DISCRETE MATHEMATICS IN RELATION TO COMPUTER SCIENCE

Numerical Modeling Tools and S-derivatives

A. N. Morozov[1]

DOI: 10.18255/1818-1015-2022-1-20-29

[1]P. G. Demidov Yaroslavl State University, 14 Sovetskaya str., Yaroslavl 150003, Russia.

MSC2020: 41A35, 41A45, 65D25
Research article
Full text in Russian

Received January 15, 2022
After revision February 28, 2022
Accepted March 9, 2022

Numerical study of various processes leads to the need of clarification (extensions) of the limits of applicability of computational constructs and modeling tools. For dynamical systems, this question may be related with a generalization of the concept of a derivative, which keeps the used constructions relevant. In this article we introduce the concept of weak local differentiability in a space of Lebesgue integrable functions and consider the consistency of this concept with such fundamental computational constructions as the Taylor expansion and finite differences, as well as properties of functions with a given type of differentiability on a segment.

The function f from $L_1[a;b]$ is called S-differentiable at the point x_0 from $(a;b)$, if there are coefficients c and q, for which $\int_{x_0}^{x_0+h}\big(f(x)-c-q\cdot(x-x_0)\big)\,dx = o(h^2)$. Formulas are found for calculating the coefficients c and q, coefficients c and q, which are conveniently denoted $f_S(x_0)$ and $f_S^{'}(x_0)$ respectively. It is shown that if the function f belongs to $W_1^{n-1}[a;b]$, n is greater than 1, and the function $f^{(n-1)}$ is S-differentiable at the point x_0 from $(a;b)$, then f is approximated by a Taylor polynomial with accuracy $o\big((x-x_0)^n\big)$, and the ratio of $\Delta_h^n(f,x_0)$ to h^n tends to $f_S^{(n)}(x_0)$ as h tends to 0. Based on the quotient $\Delta_h^n(f,\cdot)$ and h^n, a sequence is built $\big\{\Lambda_m^n[f]\big\}$ piecewise constant functions subordinate to partitions segment $[a;b]$ into m equal parts. It is shown that for the function f from $W_1^{n-1}[a;b]$, for which the value is defined $f_S^{(n)}(x_0)$, $\big\{\Lambda_m^n[f](x_0)\big\}$ converges to $f_S^{(n)}(x_0)$ as m tends to infinity, and for f from $W_p^n[a;b]$ the sequence $\big\{\Lambda_m^n[f]\big\}$ converges to $f^{(n)}$ in the norm of the space $L_p[I]$. The place of S-differentiability in practical and theoretical terms is determined by its bilateral relations with ordinary differentiability. It is proved that if f belongs to $W_1^{n-1}[I]$ and the function $f^{(n-1)}$ is uniformly S-differentiable on I, then f belongs to $C^n[I]$. The constructions are algorithmic in nature and can be applied in numerically computer research of various relevant models.

Keywords: Difference Expressions; the Taylor Polynomial; S-derivative; Numerical Simulation; Numerical Finding of Derivatives on a Computer; the Spreading of the Differentiation Operator

INFORMATION ABOUT THE AUTHORS

Anatoly Nikolaevich Morozov
correspondence author
orcid.org/0000-0001-9940-159X. E-mail: moroz@uniyar.ac.ru
PhD.

Funding: The work was carried out within the framework of the initiative research work of the YarGU. P.G. Demidov No. VIP-016.

For citation: A. N. Morozov, "Numerical Modeling Tools and S-derivatives", *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 20-29, 2022.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 29, № 1, 2022
сайт журнала: www.mais-journal.ru

DISCRETE MATHEMATICS IN RELATION TO COMPUTER SCIENCE

Инструменты численного моделирования и S-производные

А. Н. Морозов[1]

DOI: 10.18255/1818-1015-2022-1-20-29

[1]Ярославский государственный университет им. П. Г. Демидова, ул. Советская, д. 14, г. Ярославль, 150003 Россия.

УДК 519.65
Научная статья
Полный текст на русском языке

Получена 15 января 2022 г.
После доработки 28 февраля 2022 г.
Принята к публикации 9 марта 2022 г.

Численное исследование различных процессов приводит к необходимости уточнения (расширения) границ применимости вычислительных конструкций и инструментов моделирования. Для динамических систем данный вопрос может быть связан с обобщением понятия производной, сохраняющим актуальными применяемые конструкции. В настоящей статье вводится понятие слабой локальной дифференцируемости в пространстве интегрируемых по Лебегу функций и рассматриваются согласованность этого понятия с такими основополагающими вычислительными построениями как разложение Тейлора и конечные разности, а также свойства функций, обладающих данного вида дифференцируемостью на отрезке.

Функцию f из $L_1[a;b]$ назовём S-дифференцируемой в точке x_0 из $(a;b)$, если существуют коэффициенты c и q, при которых выполняется $\int_{x_0}^{x_0+h}\big(f(x)-c-q\cdot(x-x_0)\big)\,dx = o(h^2)$. Найдены формулы для вычисления коэффициентов c и q, которые удобно обозначить $f_S(x_0)$ и $f_S'(x_0)$ соответственно. Показано, что если функция f принадлежит $W_1^{n-1}[a;b]$, n больше 1, и функция $f^{(n-1)}$ является S-дифференцируемой в точке x_0 из $(a;b)$, то f приближается тейлоровским многочленом с точностью $o\big((x-x_0)^n\big)$, а отношение $\Delta_h^n(f,x_0)$ к h^n стремится к $f_S^{(n)}(x_0)$ при стремлении h к 0. На основе частного $\Delta_h^n(f,\cdot)$ и h^n строится последовательность $\big\{\Lambda_m^n[f]\big\}$ кусочно-постоянных функций, подчинённых разбиениям отрезка $[a;b]$ на m равных частей. Показано, что для функции f из $W_1^{n-1}[a;b]$, для которой определено значение $f_S^{(n)}(x_0)$, $\big\{\Lambda_m^n[f](x_0)\big\}$ сходится к $f_S^{(n)}(x_0)$ при стремлении m к бесконечности, а для $f\in W_p^n[a;b]$ последовательность $\big\{\Lambda_m^n[f]\big\}$ сходится к $f^{(n)}$ по норме пространства $L_p[I]$. Место S-дифференцируемости в практическом и теоретическом плане определяется её двусторонними соотношениями с обычной дифференцируемостью. Доказан факт, что если f принадлежит $W_1^{n-1}[I]$, и функция $f^{(n-1)}$ является равномерно S-дифференцируемой на I, то f принадлежит $C^n[I]$. Рассмотренные построения имеют алгоритмический характер, и могут быть применены в численном исследовании на ЭВМ соответствующих моделей.

Ключевые слова: разностные выражения; многочлен Тейлора; S-производная; численное моделирование; численное нахождение производных; распространение оператора дифференцирования

ИНФОРМАЦИЯ ОБ АВТОРАХ

Анатолий Николаевич Морозов
автор для корреспонденции

orcid.org/0000-0001-9940-159X. E-mail: moroz@uniyar.ac.ru
канд. физ.-мат. наук, доцент.

Финансирование: Работа выполнена в рамках инициативной НИР ЯрГУ им. П. Г. Демидова № VIP-016.

Для цитирования: A. N. Morozov, “Numerical Modeling Tools and S-derivatives”, *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 20-29, 2022.

Введение и основные обозначения

Как обычно, $L_p[I]$ обозначает пространство действительных измеримых функций, интегрируемых в степени p ($0 < p < \infty$) по Лебегу на отрезке $I = [a; b]$,

$$\|f\|_{L_p[I]} = \left(\int\limits_I |f(x)|^p \, dx \right)^{\frac{1}{p}};$$

при $p = \infty$ всюду ниже рассматривается $B[I]$ — пространство измеримых ограниченных на отрезке I функций, —

$$\|f\|_{B[I]} = \sup_{x \in I} |f(x)|.$$

Когда неясность исключена, сокращаем обозначения до L_p и $\|f\|_p$ или соответственно до B и $\|f\|_\infty$. Длину I обозначаем $|I|$.

Также используются ($k \in \mathbf{N}$) $C^k = C^k[I]$ — пространство k раз непрерывно дифференцируемых на отрезке I функций — и ($1 \leq p < \infty$)

$W_p^k = W_p^k[I] = \left\{ f : f^{(k-1)} \text{ абсолютно непрерывна на отрезке } I, f^{(k)} \in L_p[I] \right\}$

с нормами $\|f\|_p + \|f^{(k)}\|_p$.

Численное исследование различных процессов приводит к необходимости уточнения (расширения) границ применимости вычислительных конструкций и инструментов моделирования. Для динамических систем данный вопрос может быть связан с обобщением понятия производной, сохраняющим актуальными применяемые конструкции. Эффективность такого подхода была продемонстрирована, например, в работе А. П. Кальдерона и А. Зигмунда ([1]), где были даны приложения обобщённого локального понятия производной к изучению локальных свойств решений дифференциальных уравнений ($1 \leq p \leq \infty$). В одномерном случае их формулировка приводит к следующему определению.

Определение 1. *Функция $f \in L_p[I]$ называется (k, p)-дифференцируемой в точке $x \in I$, если существует алгебраический многочлен P степени не больше k, для которого выполняется*

$\|f - P\|_{L_p[J_{x,h}]} = o(h^{k+\frac{1}{p}})$, *при* $h \longrightarrow 0$, *где* $J_{x,h} = [x - h; x + h] \cap I$.

Такой многочлен может быть только один, его часто называют тейлоровским. Здесь есть возможность ещё уменьшить требования к взаимоотношению функции и приближающего её многочлена. При $k = 1$ это фактически ведёт к обновлению классической операции дифференцирования.

Определение 2. *Функцию $f \in L_1[a; b]$ назовём S-дифференцируемой в точке $x_0 \in (a; b)$, если существует алгебраический многочлен P степени не больше* 1, *для которого выполняется*

$$\int_{x_0}^{x_0+h} (f - P)(x) \, dx = o(h^2) \quad \textit{при } h \longrightarrow 0. \tag{1}$$

Односторонняя S-дифференцируемость определяется стандартным образом, в частности, применительно к точке a требуется выполнение условия (1) при $h \longrightarrow 0+$, а к точке b — при $h \longrightarrow 0-$.

Покажем единственность многочлена, рассматриваемого в определении 2. Его удобно представить в виде $P(x) = c + q \cdot (x - x_0)$.

Предложение 1. *Если $f \in L_1[a; b]$, $a < x_0 < b$ и*

$$\int_{x_0}^{x_0+h} \big(f(x) - c - q \cdot (x - x_0) \big) \, dx = o(h^2) \quad \textit{при } h \longrightarrow 0,$$

то

$$i) \qquad c = \lim_{h \to 0} \frac{1}{h} \int_{x_0}^{x_0+h} f(x) \, dx;$$

$$ii) \qquad q = \lim_{h \to 0} \frac{\Delta_h(f, x_0)}{h},$$

где

$$\Delta_h(f, x_0) \stackrel{\text{def}}{=} \frac{1}{h}\int_{x_0+h}^{x_0+2h} f(x)dx - \frac{1}{h}\int_{x_0}^{x_0+h} f(x)dx.$$

При односторонней S-дифференцируемости в точке рассматриваются соответствующие односторонние пределы.

Доказательство. Из условия сразу получается, что имеет место

$$\int_{x_0}^{x_0+h} \big(f(x) - c\big)\, dx = o(h) \quad \text{при } h \to 0,$$

откуда следует пункт *i*).

Пусть $F(x) = \int_a^x f(t)dt$. Тогда по условию справедливо разложение

$$F(x) = F(x_0) + c\cdot(x-x_0) + \frac{q}{2}\cdot(x-x_0)^2 + o\big((x-x_0)^2\big).$$

Учитывая, что

$$\Delta_h(f, x_0) = \frac{1}{h}\cdot\big(F(x_0+2h) - 2F(x_0+h) + F(x_0)\big) = \frac{\Delta_h^2(F, x_0)}{h},$$

получаем

$$\lim_{h\to 0}\frac{\Delta_h(f, x_0)}{h} = \lim_{h\to 0}\frac{q\cdot h^2 + \Delta_h^2\big(o\big((x-x_0)^2\big), x_0\big)}{h^2} = q.$$

Предложение доказано.

Ясно, что из обычной дифференцируемости функции в точке (определяемой в пространстве B) следует её S-дифференцируемость в этой точке. Получается, что там, где существуют и не равны между собой односторонние (обычные) производные или какая-то из них равна бесконечности, невозможна S-дифференцируемость. Так функция $f(x) = |x|^\alpha$ является S-дифференцируемой в нуле только при $\alpha > 1$. Рассмотрим примеры, отражающие основные черты обсуждаемого понятия.

Пример 1. *Пусть для определённости f – чётная функция, задаваемая при $x \geq 0$ формулой*

$$f(x) = \sum_{k=1}^{\infty} f_k(x),$$

где

$$f_k(x) = \begin{cases} 1, & \text{если } x \in J_k = \left[\frac{1}{2^k} - \frac{1}{2^{3k}}; \frac{1}{2^k}\right], \\ 0, & \text{если } x \notin J_k, \end{cases}$$

тогда f S-дифференцируема в нуле.

Доказательство. Для h, удовлетворяющего условию $0 < \big|h\big| \leq \frac{1}{2}$, пусть $m \in \mathbf{N}$ таково, что $\frac{1}{2^{m+1}} < \big|h\big| \leq \frac{1}{2^m}$, тогда

$$\Big|\int_0^h f(x)\, dx\Big| \leq \sum_{k=m}^{\infty}\big|J_k\big| = \sum_{k=m}^{\infty}\frac{1}{2^{3k}} = \frac{64}{7}\cdot\Big(\frac{1}{2^{m+1}}\Big)^3 < \frac{64}{7}\cdot h^3.$$

Из чего следует доказываемое утверждение.

Отметим, что пример 1 указывает на взаимосвязь S-дифференцируемости с фрактальными свойствами.

Рассмотрим случай, представляющий интерес с точки зрения моделирования «непрерывных» процессов.

Пример 2. *Функция*

$$f(x) = \begin{cases} x\cdot \sin(\frac{1}{x}), & x \neq 0, \\ 0, & x = 0; \end{cases}$$

является S-дифференцируемой в каждой точке.

Доказательство. В силу чётности функции f достаточно доказать её S-дифференцируемость справа в точке 0.

При $x > 0$ точки $x_k = 1/k\pi$, $k \in \mathbf{N}$, являются нулями данной функции.

Для каждого $0 < h < 1/\pi$ пусть $k > 2$ таково, что $\frac{1}{k\pi} \le h < \frac{1}{(k-2)\pi}$, тогда

$$\left|\int_0^h f(x)\,dx\right| \le \left|\int_0^{\frac{1}{k\pi}} f(x)\,dx\right| + \left|\int_{\frac{1}{k\pi}}^{\frac{1}{(k-2)\pi}} f(x)\,dx\right|.$$

Представим

$$\left(0; \frac{1}{k\pi}\right] = \cdots \cup \left(\frac{1}{(k+2l+2)\pi}; \frac{1}{(k+2l)\pi}\right] \cup \cdots \cup \left(\frac{1}{(k+2)\pi}; \frac{1}{k\pi}\right], \quad l \in \mathbf{N}.$$

Рассмотрим некоторый полуинтервал $\left(\frac{1}{(m+2)\pi}; \frac{1}{m\pi}\right]$ из этого набора и соответствующий интеграл. Применяя замену $t = 1/x$ получаем:

$$\left|\int_{\frac{1}{(m+2)\pi}}^{\frac{1}{m\pi}} x\cdot \sin\left(\frac{1}{x}\right)\,dx\right| = \left|\int_{m\pi}^{(m+2)\pi} \frac{\sin t}{t^3}\,dt\right| = \left|\int_{m\pi}^{(m+1)\pi} \left(\frac{\sin t}{t^3} + \frac{\sin(t+\pi)}{(t+\pi)^3}\right)\,dt\right| \le$$

$$\le \int_{m\pi}^{(m+1)\pi} \left(\frac{1}{t^3} - \frac{1}{(t+\pi)^3}\right)\,dt \le 3\pi \int_{m\pi}^{(m+1)\pi} \frac{(t+\pi)^2}{t^3\cdot(t+\pi)^3}\,dt \le 3\pi \int_{m\pi}^{(m+1)\pi} \frac{dt}{t^4} \le \frac{1}{\pi}\cdot\frac{1}{m^4}.$$

Следовательно,

$$\left|\int_0^{\frac{1}{k\pi}} f(x)dx\right| \le \frac{1}{\pi}\cdot\left(\frac{1}{k^4} + \cdots + \frac{1}{(k+2l)^4} + \cdots\right) \le \frac{1}{\pi}\int_{-1}^{\infty} \frac{dx}{(k+2x)^4} = \frac{1}{6\pi}\cdot\frac{1}{(k-2)^3},$$

$$\left|\int_{\frac{1}{k\pi}}^{\frac{1}{(k-2)\pi}} f(x)\,dx\right| \le \int_{(k-2)\pi}^{k\pi} \frac{dt}{t^3} \le \frac{2}{\pi^2}\cdot\frac{1}{(k-2)^3}.$$

Имеем

$$\left|\int_0^h f(x)\,dx\right| \le \frac{1}{\pi}\cdot\frac{1}{(k-2)^3} \le \frac{9}{\pi}\cdot h^3,$$

из чего вытекает S-дифференцируемость функции f в нуле.

1. Инструменты численного моделирования и производные в пространстве L_1

Для S-дифференцируемой в точке x_0 функции f положим

$$f_S(x_0) \stackrel{\text{def}}{=} \lim_{h\to 0} \frac{1}{h}\int_{x_0}^{x_0+h} f(x)\,dx,$$

$$f_S'(x_0) \stackrel{\text{def}}{=} \lim_{h\to 0} \frac{\Delta_h(f, x_0)}{h}.$$

Второе число назовём S-производной функции f в точке x_0. Как уже отмечалось, из существования $f'(x_0)$ следует существование $f_S'(x_0)$ и их равенство.

Рассмотрим связь некоторых классических конструкций с понятием S-дифференцируемости. Определим обычным образом k-ю разность функции f в точке x :

$$\Delta_h^k(f, x) = \sum_{j=0}^{k} (-1)^{k-j}\cdot\binom{k}{j}\cdot f(x+jh).$$

Теорема 1. *Пусть* $f \in W_1^{n-1}[a;b]$, $n \geq 2$, *и функция* $f^{(n-1)}$ *является S-дифференцируемой в точке* $x_0 \in [a;b]$, *тогда справедливо разложение*

$$f(x) = \sum_{k=0}^{n} \frac{f_S^{(k)}(x_0)}{k!} \cdot (x-x_0)^k + o\big((x-x_0)^n\big) \quad \textit{при } x \longrightarrow x_0,$$

и существуют

$$\lim_{h\to 0} \frac{\Delta_h^k(f,x_0)}{h^k} = f_S^{(k)}(x_0), \quad k = 1,\cdots,\ n,$$

где $f_S^{(n-1)}(x_0) = \big(f^{(n-1)}\big)_S(x_0)$, $f_S^{(n)}(x_0) = \big(f^{(n-1)}\big)_S'(x_0)$. *В точках a и b подразумеваются односторонние разложения и пределы.*

Доказательство. Для функции $f \in W_1^{n-1}[a;b]$, $n \geq 2$, и точки $x_0 \in (a;b)$ запишем начальное разложение Тейлора с остаточным членом в интегральной форме:

$$f(x) = \sum_{k=0}^{n-2} \frac{f^{(k)}(x_0)}{k!} \cdot (x-x_0)^k + \int_{x_0}^{x} \frac{(x-t)^{n-2} \cdot f^{(n-1)}(t)\,dt}{(n-2)!}.$$

По условию $f^{(n-1)}$ представима в виде

$$f^{(n-1)}(t) = f_S^{(n-1)}(x_0) + f_S^{(n)}(x_0)\cdot(t-x_0) + o_L\big((t-x_0)^2\big),$$

где $o_L\big((t-x_0)^2\big)$ обозначает функцию, интеграл от которой по отрезку $[x_0;x]$ даёт $o\big((x-x_0)^2\big)$ при $x \longrightarrow x_0$. Следовательно,

$$\int_{x_0}^{x} \frac{(x-t)^{n-2}\cdot f^{(n-1)}(t)\,dt}{(n-2)!} = \frac{f_S^{(n-1)}(x_0)}{(n-2)!}\cdot \int_{x_0}^{x} (x-t)^{n-2}\,dt +$$

$$+ \frac{f_S^{(n)}(x_0)}{(n-2)!}\cdot \int_{x_0}^{x} (x-t)^{n-2}\cdot(t-x_0)\,dt + \int_{x_0}^{x} \frac{(x-t)^{n-2}}{(n-2)!}\cdot o_L\big((t-x_0)^2\big)\,dt.$$

Выполнив интегрирование в правой части, получаем в итоге

$$\int_{x_0}^{x} \frac{(x-t)^{n-2}\cdot f^{(n-1)}(t)\,dt}{(n-2)!} = \frac{f_S^{(n-1)}(x_0)}{(n-1)!}\cdot(x-x_0)^{n-1} + \frac{f_S^{(n)}(x_0)}{n!}\cdot(x-x_0)^n + o\big((x-x_0)^n\big),$$

что доказывает первое утверждение теоремы.

Для k-ой разности функции хорошо известно (см., например, [2], с.159, или [3], с.54), что ($h > 0$) :

$$\Delta_h^k(x^j,\cdot) = k!\cdot h^k\cdot \delta_{j,k}, \quad j = 0,\ 1,\ \ldots,\ k,$$

где $\delta_{j,i}$ — символ Кронекера. Также имеет место простая оценка:

$$\|\Delta_h^k(f)\|_B \overset{\text{def}}{=} \|\Delta_h^k(f,\cdot)\|_{B[a;\,b-kh]} \leq 2^k\cdot\|f\|_B.$$

Поэтому из

$$f(x) = \sum_{j=0}^{k} \frac{a_j}{j!}\cdot(x-x_0)^j + o\big((x-x_0)^k\big)$$

следует

$$\lim_{h\to 0+} \frac{\Delta_h^k(f,x_0)}{h^k} = \lim_{h\to 0+} \frac{a_k\cdot h^k + \Delta_h^k\big(o\big((x-x_0)^k\big),x_0\big)}{h^k} = a_k.$$

При $h < 0$ имеем $\Delta_h^k(f, x_0) = (-1)^k \cdot \Delta_{|h|}^k(f,\ x_0 - k\cdot|h|)$, значит,

$$\lim_{h\to 0-} \frac{\Delta_h^k(f, x_0)}{h^k} = a_k.$$

Теорема доказана.

Отметим, что при $n = 2$ легко проверяется наличие обратного соотношения в первом утверждении теоремы: если $f \in W_1^1[a; b]$ и в точке $x_0 \in (a; b)$ справедливо разложение

$$f(x) = a_0 + a_1 \cdot (x - x_0) + a_2 \cdot (x - x_0)^2 + o\big((x - x_0)^2\big) \quad \text{при } x \longrightarrow x_0,$$

то функция f' является S-дифференцируемой в точке x_0,

В качестве «глобального» аналога «разделённой конечной разности» можно взять следующую конструкцию.

По определённой на отрезке $[a; b]$ функции f и разбиению $\tau_m = \{[t_{i-1}; t_i)\}_{i=1}^m$ /$a = t_0 < t_1 < \cdots < t_{m-1} < t_m = b$/ полуинтервала $[a; b)$ на равные полуинтервалы построим ступенчатую функцию, задаваемую формулой

$$\Lambda_m^k[f](x) = \frac{\Delta_h^k(f,\ t_{i-1})}{h^k} \quad \text{при } x \in [t_{i-1}; t_i)$$

на каждом интервале из τ_m /заключительный справа полуинтервал замкнём/, где $h = \frac{b-a}{m\cdot k}$.

Теорема 2. *Пусть $f \in W_1^{n-1}[a; b]$, $n \geq 2$, и функция $f^{(n-1)}$ является S-дифференцируемой в точке $x_0 \in [a; b]$, тогда в этой точке последовательность*
$\{\Lambda_m^k[f]\}$ сходится к $f_S^{(k)}(x_0)$ при $m \longrightarrow \infty$, $k = 1, \ldots, n$,
где $f_S^{(n-1)}(x_0) = \big(f^{(n-1)}\big)_S(x_0)$, $f_S^{(n)}(x_0) = \big(f^{(n-1)}\big)_S'(x_0)$.
В точках a и b подразумевается односторонняя S-дифференцируемость.

Доказательство. По теореме 1 для $1 \leq k \leq n$ из условия имеем

$$f(x) = \sum_{j=0}^{k} \frac{f_S^{(j)}(x_0)}{j!} \cdot (x - x_0)^j + o\big((x - x_0)^k\big).$$

Рассмотрим некоторое разбиение τ_m отрезка $[a; b]$. Пусть точка x_0 содержится в полуинтервале $[t_{i-1}; t_i)$, $1 \leq i \leq m$, из τ_m /при $i=m$ – в отрезке $[t_{m-1}; t_m]$/. Получаем

$$\left|\Lambda_m^k[f](x_0) - f_S^{(k)}(x_0)\right| = \left|\frac{\Delta_h^k\big(\frac{f_S^{(k)}(x_0)}{k!} \cdot (x - x_0)^k + o\big((x - x_0)^k\big),\ t_{i-1}\big)}{h^k} - f_S^{(k)}(x_0)\right| =$$

$$= \left|h^{-k}\cdot\Delta_h^k\big(o\big((x - x_0)^k\big),\ t_{i-1}\big)\right| \longrightarrow 0 \quad \text{при } m \longrightarrow \infty.$$

Здесь снова использованы формулы, применённые в доказательстве теоремы 1.

2. Исследование сходимости вычислительной конструкции $\Lambda_m^k[\,\cdot\,]$ и условий S-дифференцируемости на отрезке

Важной стороной использования обсуждаемых производных в вычислительных целях являются свойства функций, S-дифференцируемых на множестве. В теоретическом плане место таких производных определяется двусторонними соотношениями с обычными производными.

S-дифференцируемая в конкретной точке функция может изучаться как определённого вида предел непрерывно дифференцируемых функций. По предложению 1 и теоремам 1-2 в такого вида точках сохраняется сходимость выражений, основанных на «разделённых конечных разностях». Поэтому актуально рассматривать распространение оператора дифференцирования в метриках, построенных при помощи таких конструкций.

Определим обычные L_p-модули гладкости ($0 < p < \infty$):

$$\omega_k(f, t)_p = \sup_{0<h<t} \left\| \Delta_h^k(f) \right\|_{L_p[a;\, b-kh]}.$$

Как известно, на пространстве W_p^k имеет место (см. [4]) равенство:

$$\sup_{t>0} t^{-k} \omega_k(f, t)_p = \limsup_{h \to 0} \left\| h^{-k} \Delta_h^k(f) \right\|_{L_p[a;\, b-kh]} = \left\| f^{(k)} \right\|_p,$$

т. е.

$$\| \cdot \|_p + \sup_{t>0} t^{-k} \omega_k(\cdot, t)_p - \quad \text{норма.}$$

Таким образом, пространство W_p^k является пополнением C^k по данной норме, а построение k-ой производной в W_p^k равносильно распространению оператора k-кратного дифференцирования, определяемому этим пополнением.

Данное рассуждение можно дополнить описанием поведения последовательностей $\left\{ \Lambda_m^k[\,\cdot\,] \right\}$ на обсуждаемых пространствах.

Если $f \in C^k[x_0; x_0 + k \cdot h]$, то

$$\frac{\Delta_h^k(f,\ x_0)}{h^k} = f^{(k)}(\xi), \quad x_0 \le \xi \le x_0 + k \cdot h\,,$$

(см. более общую формулу в доказательстве теоремы 3), поэтому на пространстве C^k последовательности $\left\{ \Lambda_m^k[\,\cdot\,] \right\}$ сходятся равномерно.

Теорема 3. *Если $f \in W_p^k$, то $\left\{ \Lambda_m^k[f] \right\}$ сходится к $f^{(k)}$ по норме пространства L_p.*

Доказательство. Для заданной функции $f \in W_p^k$ и $\varepsilon > 0$ найдётся функция $f_\varepsilon \in C^k$ такая, что $\left\| f^{(k)} - f_\varepsilon^{(k)} \right\|_p < \varepsilon$. Получается, что в неравенстве

$$\left\| \Lambda_m^k[f] - f^{(k)} \right\|_p \le \left\| \Lambda_m^k[f] - \Lambda_m^k[f_\varepsilon] \right\|_p + \left\| \Lambda_m^k[f_\varepsilon] - f_\varepsilon^{(k)} \right\|_p + \left\| f^{(k)} - f_\varepsilon^{(k)} \right\|_p$$

третье слагаемое в правой части может быть сделано сколь угодно малым за счёт выбора функции f_ε, а второе слагаемое станет меньше любой требуемой величины, начиная с некоторого соответственно большого номера m. Нужно оценить первое слагаемое.

В условиях теоремы при произвольном значении $m \in \mathbf{N}$ по определению имеет место

$$\left\| \Lambda_m^k[f] \right\|_p = \left(\sum_{i=1}^{m} \left| \frac{\Delta_h^k(f,\ t_{i-1})}{h^k} \right|^p k \cdot h \right)^{\frac{1}{p}}.$$

Для k-й разности функции из W_1^k выполняется (см. [3], с. 137)

$$\Delta_h^k(f, x) = h^k \int_0^{k \cdot h} \frac{N_k(t/h)}{h}\, f^{(k)}(x + t)\, dt,$$

где N_k – нормализованный B-сплайн порядка k с узлами в точках $0,\ 1, \cdots,\ k$. Т. е. N_k – неотрицательная кусочно-полиномиальная функция степени $k-1$, принадлежащая пространству $C^{k-2}(-\infty; \infty)$, имеющая носитель $(0;\ k)$ с $k+1$ равноотстоящими узлами на нём (включая конечные точки интервала) и обладающая свойством $\int_0^k N_k(t)\, dt = 1$ ([3], с.128, формула (4.40)). Кроме того, $\| N_k \|_{B[0;\, k]} \le 1$ ([3], с. 125, формула (4.32)).

Рассмотрим отдельно случай $p = 1$. Очевидно

$$\left\|\Lambda_n^k[f]\right\|_1 = k\cdot \sum_{i=1}^{m} \left| \int_0^{k\cdot h} N_k(t/h)\, f^{(k)}(t_{i-1}+t)\, dt \right| \le k\cdot \left\|f^{(k)}\right\|_1.$$

Пусть $1 < p < \infty$. Тогда по интегральному неравенству Гёльдера

$$\left\|\Lambda_m^k[f]\right\|_p \le k^{\frac{1}{p}} \left(\sum_{i=1}^{m} \left(\int_0^{k\cdot h} \left(\frac{N_k(t/h)}{h} \right)^{\frac{p}{p-1}} dt \right)^{p-1} \left(\int_0^{k\cdot h} \left| f^{(k)}(t_{i-1}+t) \right|^p dt \right) h \right)^{\frac{1}{p}} =$$

$$= k^{\frac{1}{p}} \left(\int_0^{k\cdot h} \left(N_k(t/h) \right)^{\frac{p}{p-1}} d(t/h) \right)^{\frac{p-1}{p}} \left(\sum_{i=1}^{m} \int_{t_{i-1}}^{t_i} \left| f^{(k)}(t) \right|^p dt \right)^{\frac{1}{p}} \le$$

$$\le k^{\frac{1}{p}} \cdot \left\|f^{(k)}\right\|_p. \tag{2}$$

Заключительное неравенство в преобразованиях основано на том, что из свойств $\|N_k\|_{B[0;\,k]} \le 1$ и $\|N_k\|_{L_1[0;\,k]} = 1$ вытекает $\|N_k\|_{L_q[0;\,k]} \le 1$ при любом $1 < q < \infty$.

Формула (2) показывает, что $\left\|\Lambda_m^k[f] - \Lambda_m^k[f_\varepsilon]\right\|_p$ оценивается через $\left\|f^{(k)} - f_\varepsilon^{(k)}\right\|_p$, из чего следует утверждение теоремы.

Теорема доказана.

Дальнейшее распространение оператора дифференцирования может быть осуществлено с помощью аналогичного построения.

Пусть $(0 < p < \infty)$

$$H_p^k = \left\{ f \in L_p \,:\, \sup_{t>0} t^{-k} \omega_k(f,t)_p < \infty \right\}.$$

Очевидно, что $H_{p_1}^k \supset H_{p_2}^k$, если $p_1 < p_2$. Чтобы избежать постоянных оговорок, для значений индекса, меньших единицы, в дальнейшем будет использоваться буква r, т. е. всюду ниже $0 < r < 1$.

Определим для каждого r пространство Y_r^k как пополнение C^k в метрике, порождаемой квазинормой пространства H_r^k (см. [5]). Метрику определяет функционал

$$\left\| \cdot \right\|_r^r + \left(\sup_{t>0} t^{-k} \omega_k(\,\cdot\,, t)_r \right)^r.$$

Для функции $f \in C^k$ применим обозначение:

$$\Lambda^k[f] \stackrel{\text{def}}{=} \lim_{n\to\infty} \Lambda_m^k[f].$$

Так как на C^k выполняется

$$\left\|f^{(k)}\right\|_r = \lim_{t\to 0}\; t^{-k} \omega_k(f,t)_r,$$

то имеем следующий факт: для $f_1, f_2 \in C^k$

$$\left\|\Lambda^k[f_1] - \Lambda^k[f_2]\right\|_r^r = \left\|(f_1 - f_2)^{(k)}\right\|_r^r = \left(\lim_{t\to 0}\; t^{-k} \omega_k(f_1 - f_2, t)_r \right)^r \le \left(\sup_{t>0}\; t^{-k} \omega_k(f_1 - f_2, t)_r \right)^r.$$

Следовательно, действие Λ^k выходит за пределы пространства W_1^k.

Теорема 4. ([5]) Оператор Λ^k при каждом r имеет единственное линейное непрерывное распространение до оператора из Y_r^k в L_r. Это распространение обладает свойствами:

$$i)\ \lim_{t\to 0} t^{-k}\omega_k(f,t)_r = \left\|\Lambda^k[f]\right\|_r,$$

$$ii)\ \text{если}\ f\in W_1^k[J],\ J\subseteq I,\ \text{то}\ \Lambda^k[f]\Big|_J = f^{(k)}.$$

Обратное соотношение между S-дифференцируемостью и дифференцируемостью (в пространстве B) может быть выражено, например, при помощи равномерного существования первой конструкции. S-дифференцируемая во всех точках отрезка I функция f называется равномерно S-дифференцируемой на I, если для любого числа $\varepsilon > 0$ найдется число $\delta > 0$ такое, что при $|h| < \delta$ для каждой точки $x_0 \in I$ выполняется

$$\frac{1}{h^2}\left|\int_{x_0}^{x_0+h}(f-P)(x)\,dx\right| < \varepsilon \quad (x_0+h\in I),$$

где P – многочлен из условия S-дифференцируемости в точке x_0.

Для единой формулировки положим $W_1^0 = L_1$.

Теорема 5. *Пусть $f\in W_1^{n-1}[I]$ $(n\in\mathbf{N})$ такова, что функция $f^{(n-1)}$ является равномерно S-дифференцируемой на I, тогда $f\in C^n[I]$.*

Доказательство. Из условия теоремы получаем, что равномерно по точкам x_0 из I выполняется

$$f^{(n-2)}(x) = f^{(n-2)}(x_0) + f_S^{(n-1)}(x_0)\cdot(x-x_0) + \frac{f_S^{(n)}(x_0)}{2}\cdot(x-x_0)^2 + o\big((x-x_0)^2\big)\ \text{ при } x\longrightarrow x_0,$$

/для $n = 1$ под $f^{(n-2)}$ подразумевается F – первообразная функции f/. Следовательно, равномерно на I есть соотношение

$$\frac{\Delta_h^2\big(f^{(n-2)},x_0\big)}{h^2} = f_S^{(n)}(x_0) + \frac{\Delta_h^2\big(o\big((x-x_0)^2\big),x_0\big)}{h^2} = f_S^{(n)}(x_0) + o\big(h^0\big)\ \text{ при } h\longrightarrow 0. \tag{3}$$

Убедимся, что $f_S^{(n)}\in C[I]$. Из равномерного выполнения на I соотношения (3) вытекает равномерная сходимость $\left\{\Lambda_m^2[f^{(n-2)}]\right\}$ к $f_S^{(n)}$. Разбиения отрезка на равные части обладают свойством «смешения узлов» (см., например,[3], с.241), поэтому из равномерной сходимости последовательности ступенчатых функций, подчинённых таким разбиениям, следует непрерывность предельной функции (предположив противоположное, тот час же получим противоречие).

На основе условия $f_S^{(n)}\in C[I]$ получаем из (2), что

$$\sup_{t>0} t^{-2}\omega_2(f^{(n-2)},t)_p = \limsup_{h\to 0}\|h^{-2}\Delta_h^2(f^{(n-2)})\|_{L_p[a;\,b-2h]} = \|f_S^{(n)}\|_p < \infty.$$

Это даёт принадлежность функции $f^{(n-2)}$ пространству W_p^2 при каждом $1\le p<\infty$, из чего следует существование n-ой производной и равенство $f^{(n)} = f_S^{(n)}$ в пространствах L_p.

References

[1] A. P. Calderon and A. Zygmund, “Local Properties of Solution of Elliptic Partial Differential Equation”, *Studia Mathematica*, vol. 20, no. 2, pp. 171–225, 1961.

[2] V. K. Dzyadyk, *Introduction to the Theory of Uniform Approximation of Functions by Polynomials*. Nauka, 1977.

[3] L. L. Schumaker, *Spline Functions: Basic Theory*. Wiley, New York, 1981.

[4] Y. A. Brudnyi, “Criteria for the Existence of Derivatives in Lp”, *Mathematics of the USSR-Sbornik*, vol. 2, no. 1, pp. 35–55, 1967.

[5] A. N. Morozov, “Local Approximations of Differentiable Functions”, *Mathematical Notes*, vol. 100, no. 2, pp. 256–262, 2016.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 29, NO. 1, 2022
journal homepage: www.mais-journal.ru

SOFTWARE

Recursive-Parallel Algorithm for Solving the Graph-Subgraph Isomorphism Problem

V. V. Vasilchikov[1]

DOI: 10.18255/1818-1015-2022-1-30-43

[1]P. G. Demidov Yaroslavl State University, 14 Sovetskaya str., Yaroslavl 150003, Russia.

MSC2020: 68W10
Research article
Full text in Russian

Received November 29, 2021
After revision February 28, 2022
Accepted March 9, 2022

The paper proposes a parallel algorithm for solving the Graph-Subgraph Isomorphism Problem and makes an experimental study of its efficiency. The problem is one of the most famous NP-complete problems. Its solution may be required when solving many practical problems associated with the study of complex structures. We solve the problem in a formulation that requires finding all existing isomorphic substitutions or proving their absence. In view of the high complexity of the problem, it is natural to want to speed up its solution by parallelizing the algorithm.
We used the RPM_ParLib library, developed by the author, as the main tool to program the algorithm. This library allows us to develop effective applications for parallel computing on a local network under the control of the runtime environment .NET Framework. Thanks to this library, applications have the ability to generate parallel branches of computation directly during program execution and dynamically redistribute work between computing modules. Any language with support of the .NET Framework can be used as a programming language in conjunction with this library. For the numerical experiment, an open database from the Internet was used, which was specially developed to study algorithms for searching for isomorphic substitutions. Also, the author has developed a special application in C# for generating additional sets of initial data with specified characteristics. The aim of the experiment was to study the speedup achieved due to the recursively parallel organization of computations. The paper provides a detailed description of the proposed algorithm, as well as the results obtained during the experiment.

Keywords: graph-subgraph isomorphism problem; parallel algorithm; recursion; .NET

INFORMATION ABOUT THE AUTHORS

Vladimir V. Vasilchikov correspondence author	orcid.org/0000-0001-7882-8906. E-mail: vvv193@mail.ru PhD.

Funding: This work was supported by initiative program VIP-016.

For citation: V. V. Vasilchikov, “Recursive-Parallel Algorithm for Solving the Graph-Subgraph Isomorphism Problem”, *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 30-43, 2022.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 29, № 1, 2022
сайт журнала: www.mais-journal.ru

SOFTWARE

Рекурсивно-параллельный алгоритм решения задачи об изоморфизме граф-подграф

В. В. Васильчиков[1]

DOI: 10.18255/1818-1015-2022-1-30-43

[1]Ярославский государственный университет им. П. Г. Демидова, ул. Советская, д. 14, г. Ярославль, 150003 Россия.

УДК 519.688: 519.85
Научная статья
Полный текст на русском языке

Получена 29 ноября 2021 г.
После доработки 28 февраля 2022 г.
Принята к публикации 9 марта 2022 г.

В работе предложен параллельный алгоритм решения задачи об изоморфизме граф-подграф и произведено экспериментальное исследование его эффективности. Задача является одной из самых известных NP-полных задач. Ее решение может потребоваться при решении многих практических задач, связанных с исследованием сложных структур. Мы решаем ее в постановке, в которой требуется найти все возможные изоморфизмы или доказать отсутствие таковых. Ввиду высокой трудоемкости задачи естественным является желание ускорить ее решение за счет распараллеливания алгоритма.

Для организации параллельных вычислений автором использовалась библиотека RPM_ParLib, которая позволяет создавать параллельные приложения, работающие в локальной вычислительной сети под управлением среды исполнения .NET Framework. Библиотека поддерживает рекурсивно-параллельный стиль программирования и обеспечивает эффективное распределение работы и динамическую балансировку загрузки вычислительных модулей в процессе исполнения программы. Она может быть использована для приложений, написанных на любом языке программирования, поддерживаемом .NET Framework. Для проведения численного эксперимента использовалась открытая база данных из сети Интернет, специально разработанная для исследования алгоритмов поиска изоморфизмов. Также автором было разработано специальное приложение на языке C# для генерации собственных дополнительных наборов исходных данных с заданными характеристиками. Целью эксперимента было исследование ускорения, достигаемого за счет рекурсивно-параллельной организации вычислений. В работе приводится подробное описание предлагаемого алгоритма, а также полученных в ходе эксперимента результатов.

Ключевые слова: изоморфизм граф-подграф; параллельный алгоритм; рекурсия; .NET

ИНФОРМАЦИЯ ОБ АВТОРАХ

Владимир Васильевич Васильчиков
автор для корреспонденции

orcid.org/0000-0001-7882-8906. E-mail: vvv193@mail.ru
канд. техн. наук, зав. кафедрой вычислительных и программных систем.

Финансирование: Работа выполнена при поддержке инициативного проекта VIP-016.

Для цитирования: V. V. Vasilchikov, “Recursive-Parallel Algorithm for Solving the Graph-Subgraph Isomorphism Problem”, *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 30-43, 2022.

Введение

Задача об изоморфизме граф-подграф является одной из классических NP-полных задач дискретной оптимизации [1]. Потребность в решении этой задачи возникает в самых разных предметных областях, где требуется установление идентичности структур тех или иных сложных систем. В качестве примеров можно назвать транспортные, энергетические системы, системы связи, электронные схемы, задачи сравнения молекулярных структур, системы распознавания образов, а также задачи математической химии, исследование социальных сетей и многие другие.

Для решения данной задачи в разное время было разработано и исследовано множество алгоритмов. В числе самых известных можно назвать алгоритм Ульмана [2], алгоритм VF, первая версия которого была предложена Корделлой, Фогиа, Самсоне и Венто в 1997 году [3] и решала задачу быстрее, а также его усовершенствованный вариант VF2 [4]. Позднее были предложены различные модификации этих алгоритмов, например, VF3 [5], а также алгоритмы, построенные на других подходах: предварительной оценке совместимости вершин графов [6, 7], бит-векторные алгоритмы [8].

Ввиду высокой трудоемкости данной задачи возникает естественный интерес к разработке и исследованию параллельных алгоритмов ее решения. Существует ряд публикаций, посвященных именно таким алгоритмам, основанным на самых разных подходах к распараллеливанию. Например, в [9] был предложен алгоритм решения задачи средствами MPI. В [10] произведено описание алгоритма VF3P – параллельной версии VF3 и исследована его эффективность. VF3P в основном базируется на динамическом назначении задач на обработку с использованием глобального и, возможно, локальных стеков для хранения подзадач, что на наш взгляд, может ограничить возможности масштабирования.

При подготовке данной работы автор использовал специальные программные инструменты для организации параллельных вычислений в соответствии концепцией рекурсивно-параллельного (РП) программирования. В [11] изложены основные принципы организации таких вычислений, алгоритмы и механизмы поддержки рекурсивно-параллельного стиля программирования. Ранее разработанные автором библиотеки [12, 13] позволяют относительно легко создавать, отлаживать и эксплуатировать РП-приложения в среде .NET Framework. Функциональные возможности упомянутых библиотек подробно описаны в [14]. Библиотеки успешно применялись при разработке и исследовании параллельных алгоритмов решения задачи о клике [14], задачи коммивояжера [15], задачи о рюкзаке [16], а также задачи об изоморфизме неориентированных графов [17]. При этом стратегия разделения задачи на параллельно решаемые подзадачи в целях наилучшего использования вычислительных ресурсов каждый раз выстраивалась по-новому.

Постановка задачи

Напомним формулировку задачи в постановке из [1]. Пусть есть два графа, заданных своими множествами вершин и дуг: $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$. Существует ли подмножество $V \subseteq V_1, E \subseteq E_1$ и взаимно-однозначная функция $f(V_2 \longrightarrow V)$ такие, что $|V| = |V_2|, |E| = |E_2|, \{u, v\} \in E_2 \iff \{f(u), f(v)\} \in E$. Упомянутая функция, очевидно, задает некоторую изоморфную подстановку. Мы будем решать задачу в несколько более широкой и более востребованной формулировке, которая требует найти все такие подстановки.

Последовательный алгоритм решения задачи

Как мы отметили выше, разными авторами было предложено много различных алгоритмов решения задачи, основанных на принципиально разных подходах. Нашей целью является исследование возможностей распараллеливания такого широко используемого алгоритма, как VF2.

Сначала опишем упомянутый последовательный алгоритм, поскольку именно он является основой для дальнейшего распараллеливания решения задачи. В основном наш вариант алгорит-

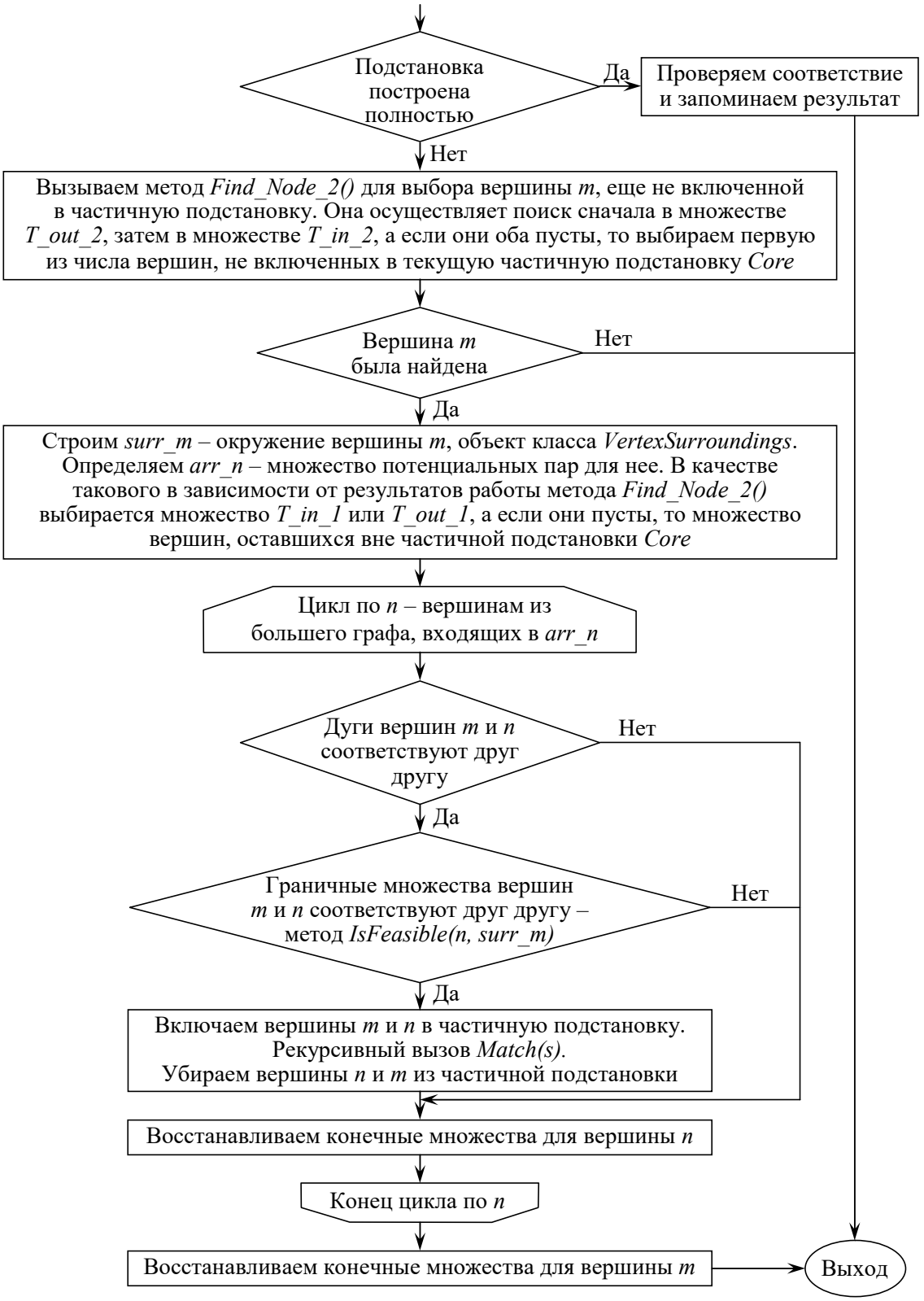

Fig. 1. Block diagram of serial variant of the method *Match(s)*

Рис. 1. Блок-схема последовательного варианта метода *Match(s)*

ма будет соответствовать его описанию, изложенному в [4]. Блок-схема алгоритма представлена на Рис. 1.

Пусть $N = |V1|$ – количество вершин в первом (большем) графе, а $M = |V2|$ – во втором. Основной структурой данных является строящаяся частичная подстановка $Core$, в которой хранятся соответствующие друг другу вершины $n \in V_1$ и $m \in V_2$. Когда ее размер достигает M, это означает, что очередная искомая подстановка построена полностью.

Строительством подстановки занимается рекурсивный метод $Match(s)$, где s – текущее состояние. На каждом уровне рекурсии он пытается найти подходящую пару (n, m) и добавить ее в $Core$. Для этого он сначала вызывает метод $Find\_Node\_2()$ для выбора m, его описание мы приведем чуть ниже. Далее он собирает информацию об окружении вершины m, сохраняет ее в специальном объекте (у нас он назван $surr\_m$), а затем последовательно перебирает вершины-кандидаты на роль n, оценивая их пригодность через вызов метода $IsFeasible(n, surr\_m)$.

Упомянутые методы используют понятие так называемых терминальных множеств, а именно: $T_1^{in}(s)$ – множество вершин, еще не включенных в $Core$, дуги из которых ведут в вершины первого графа, включенные строящуюся частичную подстановку. Аналогичным образом определяется $T_1^{out}(s)$ – множество вершин для исходящих дуг, а также множества $T_2^{in}(s)$ и $T_2^{out}(s)$ для второго графа. Отметим здесь, что в процессе сбора информации об окружении вершины m для того, чтобы метод $IsFeasible(n, surr\_m)$ быстрее отсекал недопустимые варианты, собирается информация о количестве исходящих и входящих дуг для вершин, включенных в концевые множества. В нашем варианте алгоритма мы дополнительно учитываем и количество двунаправленных дуг.

Метод $Find\_Node\_2()$ для выбора m осуществляет сначала поиск вершины в множестве $T_2^{out}(s)$, затем при отсутствии таковой в множестве $T_2^{in}(s)$ и, наконец, если ничего найти не удалось, среди оставшихся вершин, не включенных в $Core$. Метод $IsFeasible(n, surr\_m)$ считает вершину n неподходящим кандидатом на включение в $Core$, если хотя бы одна количественная характеристика о связях с вершинами из $Core$ у нее меньше, чем у вершины m. К таковым характеристикам мы отнесли:

- количество дуг из $Core$ в вершину;
- количество дуг из вершины в $Core$;
- количество двунаправленных дуг между $Core$ и вершиной;
- количество входящих, выходящих и двунаправленных дуг между вершиной и соответствующим множеством T^{in};
- количество входящих, выходящих и двунаправленных дуг между вершиной и соответствующим множеством T^{out}.

Мы также экспериментировали с дополнительными проверками на совместимость, учитывающими количество всех дуг без учета того, куда они ведут. Эти проверки никакого выигрыша не дали, поэтому мы от них отказались.

Если вершина n не была отвергнута, мы включаем пару (n, m) в $Core$, состояние s тем самым меняется, и рекурсивно вызываем $Match(s)$. После этого исключаем пару (n, m) из $Core$, восстанавливаем конечные множества в состояние до добавления n и продолжаем цикл по перебору вершин-кандидатов из первого графа.

Распараллеливание алгоритма

Описанный выше алгоритм представляет собой обход дерева вариантов совмещения вершин с постоянной проверкой возможности получения на очередной ветви корректной подстановки и отсечением неудачных вариантов. Ветви порождаются в процессе вычислений, и их трудоемкость не представляется возможным оценить заранее. Концепция рекурсивного распараллеливания [11] была разработана специально для таких задач и позволяет относительно легко справиться с возникающими при выстраивании параллельной стратегии проблемами.

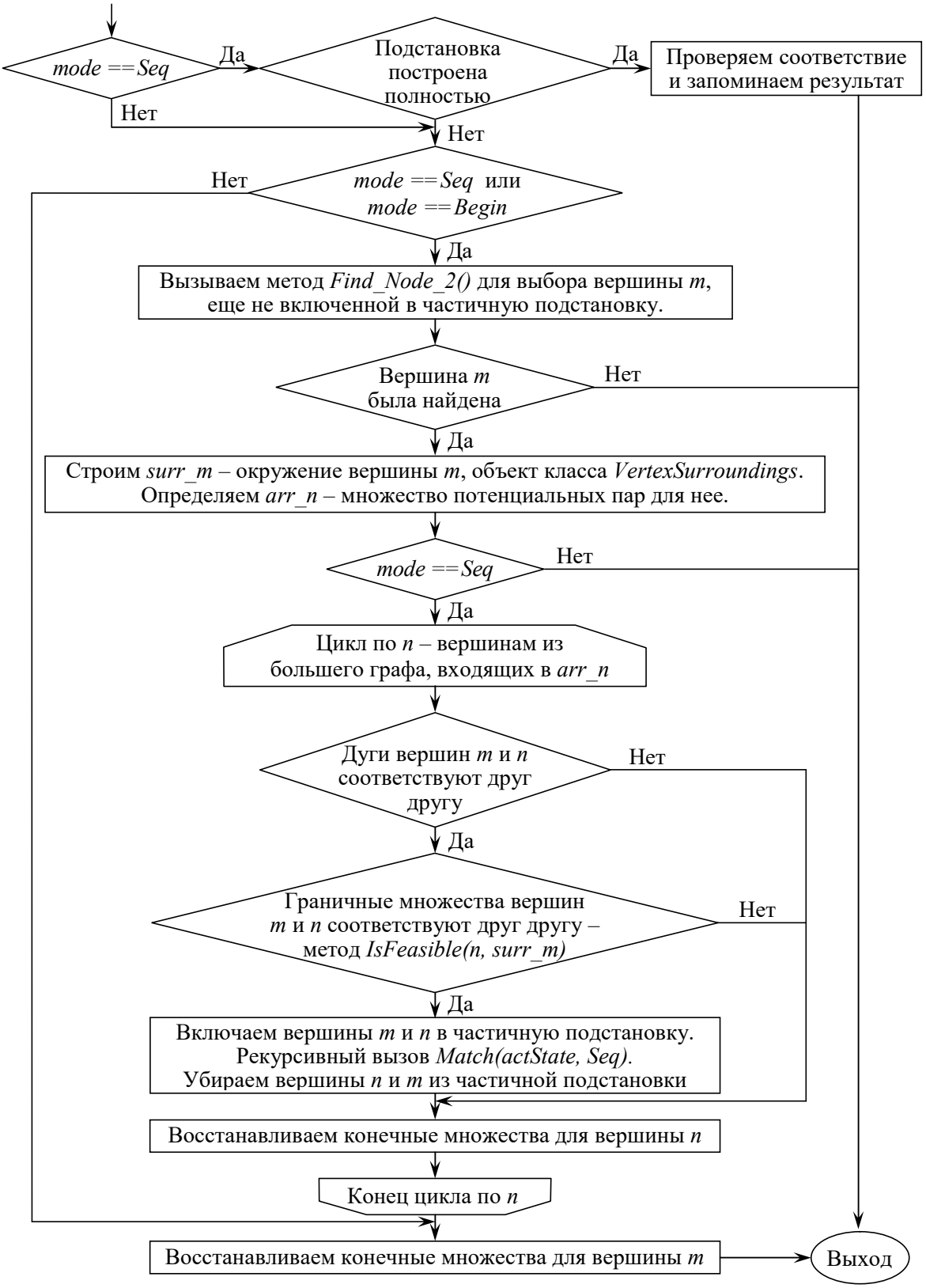

Fig. 2. Block diagram of parallel variant of the method *Match(actState, mode)*

Рис. 2. Блок-схема параллельного варианта метода *Match(actState, mode)*

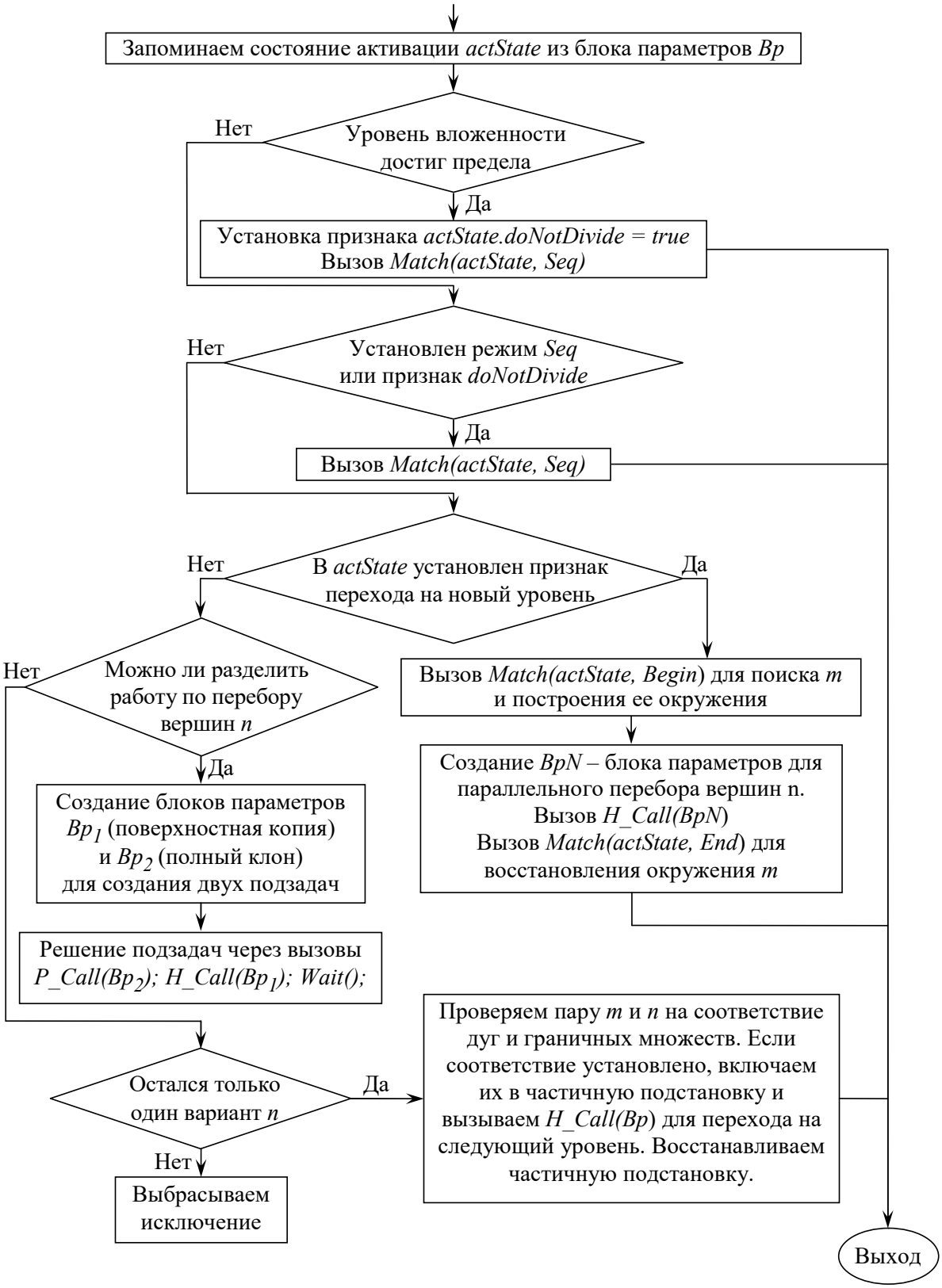

Fig. 3. Block diagram of the method *Parallel_VF2(Bp)*

Рис. 3. Блок-схема метода *Parallel_VF2(Bp)*

В нашем случае основным претендентом на распараллеливание представляется цикл по перебору n – вершин из большего графа внутри метода $Match(s)$. Оно легко может быть осуществлено в соответствии со стандартной РП-схемой [11], то есть рекурсивным делением пополам списка претендентов на роль n. При этом подзадача для обработки одной половины претендентов остается для решения на текущем процессорном модуле (ПМ), другая оформляется, как потенциально мигрирующая подзадача (ПМП) и при необходимости может быть передана для исполнения на другом ПМ. То, что порождаемые при этом подзадачи имеют, очевидно, очень разную и непредсказуемую трудоемкость, при применении РП-подхода не является проблемой. Библиотека поддержки рекурсивно-параллельного стиля программирования [14] обеспечивает достаточно равномерное распределение работы по системе на начальном этапе вычислений и при необходимости динамическое его перераспределение на последующих этапах. Требуется только создать достаточное количество ПМП.

Однако в отличие от стандартной схемы рекурсивного деления работы здесь присутствует вычисление $surr\_m$ – окружения вершины m, объекта, содержащего довольно большой объем информации, для сбора которой требуется порядка $O(N)$ операций. Собранная однажды, эта информация требуется при рассмотрении всех претендентов на роль n. Здесь приходится выбирать из двух вариантов: либо заниматься повторными вычислениями в каждой новой активации подзадачи, которая может выполняться на каком угодно ПМ, либо сохранять ее в блоке параметров и передавать как часть ПМП. После проведенных экспериментов предпочтение было отдано второму варианту.

Естественно, возникает вопрос о том, когда следует прекратить деление задачи и продолжить вычисления на данной ветви последовательно. Если этого не сделать, затраты на порождение параллельных подзадач и их оформление в виде ПМП сведут на нет (если не хуже) весь выигрыш от распараллеливания. Автором было принято самое естественное решение: ограничить глубину рекурсии, то есть по достижении некого значения $maxDepth$ (фактически это размер построенной части подстановки $Core$) мы переключаемся в последовательный режим.

Необходимость вычисления окружения вершины m и включения его в блок параметров, а также восстановления состояния активации на обратном ходе рекурсии потребовали внесения изменений в метод $Match(s)$. В нем были выделены три основных этапа вычислений: нахождение окружения вершины m до цикла по n, собственно цикл и восстановление состояния на обратном ходе. В параллельной версии метод $Match(actState, mode)$ имеет поэтому два параметра: объект, хранящий текущее состояние активации, и режим, определяющий, какие этапы следует выполнить. Предусмотрено три значения этого режима: выполняется только начальный этап, все этапы (последовательное выполнение) или только восстановление состояния после возврата из рекурсивного вызова метода $Match(s)$. Соответствующая блок-схема представлена на Рис. 2.

За порождение параллельных подзадач, их оформление в виде активаций, вызов (на данном вычислительном модуле или как ПМП), а также синхронизацию на обратном пути рекурсии отвечает метод $Parallel\_VF2(Bp)$. Его блок-схема представлена на Рис. 3.

Описание эксперимента и его результаты

Целью эксперимента была оценка эффективности работы предложенного параллельного алгоритма на сгенерированных случайным образом парах графов с различными характеристиками. В [18] предложена обширная база данных с разбитым по разным категориям набором графов именно для решения задач о граф-граф или граф-подграф изоморфизме. В ней представлены случайные графы различной плотности, регулярные и нерегулярные сети размерности 2D, 3D и 4D, а также регулярные и нерегулярные графы с ограниченной степенью вершины.

Мы исследовали работу нашего алгоритма на нескольких сотнях пар графов из этой базы. На исходных данных из некоторых групп даже последовательный алгоритм обеспечивал слишком быстрое получение результата, поэтому на них исследовать работу параллельного алгоритма

представлялось неинтересным. В результате мы отобрали те примеры исходных данных, время решения задачи на которых составляло для имевшихся в нашем распоряжении компьютерах несколько минут или несколько десятков минут. Далее будут представлены результаты решения задачи для двух категорий данных из этой базы:

- поиск всех изоморфизмов для графов на 1000 вершин с плотностью заполнения 0.1 (группа iso_r01_m1000);
- поиск всех граф-подграф изоморфизмов для пары графов размера 200 и 40 соответственно с плотностью заполнения 0.1 (группа $si2\_r01\_m200$).

Также автором была написана программа генерации пар случайных графов для данной задачи, которые удовлетворяли бы некоторым дополнительным условиям. На этих исходных данных также производилось исследование эффективности параллельного алгоритма. Ниже представлены результаты решения задачи для двух групп исходных данных:

- поиск всех граф-подграф изоморфизмов для пары регулярных графов размера 400 и 80 соответственно с плотностью заполнения 0.5 (группа $Gpi\_Regular\_400x50x80\_\_1$);
- поиск всех граф-подграф изоморфизмов для пары случайных графов размера 300 и 50 соответственно с плотностью заполнения 0.4 с гарантированным наличием не менее двух изоморфизмов (группа $Gpi\_Random\_300x0.4\_\_50x0.4\_\_2$).

В процессе тестирования использовались компьютеры на базе двухъядерного процессора Intel Core i3-7100 (максимальное количество потоков 4) с тактовой частотой 3.90 GHz и 8 GB оперативной памяти, работающие под управлением 64-разрядной ОС Windows 10. Пропускная способность сети равнялась 100 Mb/s. Как было отмечено выше одним из важных параметров, управляющих ходом вычислений, в нашем алгоритме является значение $maxDepth$ ограничивающее глубину рекурсии. В ходе эксперимента мы задали ему значение 2, что позволило создать достаточное для балансировки нагрузки количество подзадач и в то же время избежать неоправданных накладных расходов на их оформление в виде ПМП.

В таблицах приведены результаты вычислений для некоторых пар графов из упомянутых четырех групп. В них указано количество ветвей для порождаемого дерева поиска ($Branches$), время решения задачи последовательным алгоритмом и время решения параллельным алгоритмом при использовании нескольких компьютеров (РМ). Тот факт, что параллельный алгоритм даже на одном компьютере работал примерно вдвое быстрее последовательного, объясняется наличием двух ядер и многопоточной организацией вычислений. Можно также отметить заметную тенденцию к росту ускорения с увеличением объема вычислений, что позволяет рассчитывать на хорошую масштабируемость предложенного алгоритма.

Table 1. The Acceleration of the RP-algorithm on examples iso_r01_m1000

Таблица 1. Ускорение РП-алгоритма на примерах iso_r01_m1000

Source Data		**Sequently**	**1 PM**	**2 PM**	**4 PM**	**6 PM**	**8 PM**	**12 PM**	**16 PM**
1 (A77)	Exec. Time (s)	556.33	302.03	176.12	107.95	87.78	77.44	61.96	54.52
Branches (mln):	Ratio to Seq		***1.84***	***3.16***	***5.15***	***6.34***	***7.18***	***8.98***	***10.20***
119.25	Ratio to 1 PM		**1.00**	**1.71**	**2.80**	**3.44**	**3.90**	**4.87**	**5.54**
2 (A97)	Exec. Time (s)	642.94	337.16	279.02	170.72	154.45	122.18	102.47	89.20
Branches (mln):	Ratio to Seq		***1.91***	***2.30***	***3.77***	***4.16***	***5.26***	***6.27***	***7.21***
146.87	Ratio to 1 PM		**1.00**	**1.21**	**1.97**	**2.18**	**2.76**	**3.29**	**3.78**
3 (A72)	Exec. Time (s)	660.73	344.99	203.05	130.40	101.50	89.16	69.07	63.84
Branches (mln):	Ratio to Seq		***1.92***	***3.25***	***5.07***	***6.51***	***7.41***	***9.57***	***10.35***
136.06	Ratio to 1 PM		**1.00**	**1.70**	**2.65**	**3.40**	**3.87**	**4.99**	**5.40**
4 (A03)	Exec. Time (s)	779.31	404.12	241.05	150.40	116.62	94.34	75.58	65.37
Branches (mln):	Ratio to Seq		***1.93***	***3.23***	***5.18***	***6.68***	***8.26***	***10.31***	***11.92***
176.68	Ratio to 1 PM		**1.00**	**1.68**	**2.69**	**3.47**	**4.28**	**5.35**	**6.18**
5 (A92)	Exec. Time (s)	955.58	474.45	286.76	175.38	126.64	141.27	79.83	70.36
Branches (mln):	Ratio to Seq		***2.01***	***3.33***	***5.45***	***7.55***	***6.76***	***11.97***	***13.58***
169.38	Ratio to 1 PM		**1.00**	**1.65**	**2.71**	**3.75**	**3.36**	**5.94**	**6.74**
6 (A57)	Exec. Time (s)	1161.75	577.64	365.49	212.59	174.97	129.84	110.98	84.29
Branches (mln):	Ratio to Seq		***2.01***	***3.18***	***5.46***	***6.64***	***8.95***	***10.47***	***13.78***
271.67	Ratio to 1 PM		**1.00**	**1.58**	**2.72**	**3.30**	**4.45**	**5.20**	**6.85**
7 (A64)	Exec. Time (s)	1566.34	771.90	515.62	296.38	223.50	178.24	152.70	147.13
Branches (mln):	Ratio to Seq		***2.03***	***3.04***	***5.28***	***7.01***	***8.79***	***10.26***	***10.65***
356.51	Ratio to 1 PM		**1.00**	**1.50**	**2.60**	**3.45**	**4.33**	**5.06**	**5.25**
8 (A16)	Exec. Time (s)	1735.61	850.66	481.77	285.07	213.69	180.66	140.23	116.73
Branches(mln):	Ratio to Seq		***2.04***	***3.60***	***6.09***	***8.12***	***9.61***	***12.38***	***14.87***
367.71	Ratio to 1 PM		**1.00**	**1.77**	**2.98**	**3.98**	**4.71**	**6.07**	**7.29**
9 (A74)	Exec. Time (s)	1715.23	871.33	492.44	278.35	195.31	158.87	118.27	95.88
Branches (mln):	Ratio to Seq		***1.97***	***3.48***	***6.16***	***8.78***	***10.80***	***14.50***	***17.89***
371.62	Ratio to 1 PM		**1.00**	**1.77**	**3.13**	**4.46**	**5.48**	**7.37**	**9.09**
10 (A45)	Exec. Time (s)	1918.89	907.84	603.72	368.71	269.00	207.04	177.11	135.57
Branches (mln):	Ratio to Seq		***2.11***	***3.18***	***5.20***	***7.13***	***9.27***	***10.83***	***14.15***
397.53	Ratio to 1 PM		**1.00**	**1.50**	**2.46**	**3.37**	**4.38**	**5.13**	**6.70**

Table 2. The Acceleration of the RP-algorithm on examples si2_r01_m200

Таблица 2. Ускорение РП-алгоритма на примерах si2_r01_m200

Source Data		Sequently	1 PM	2 PM	4 PM	6 PM	8 PM	12 PM	16 PM
1 (A40)	Exec. Time (s)	711.20	385.67	204.40	108.65	81.57	63.72	52.62	47.49
Branches (mln):	Ratio to Seq		***1.84***	***3.48***	***6.55***	***8.72***	***11.16***	***13.51***	***14.98***
771.14	Ratio to 1 PM		**1.00**	**1.89**	**3.55**	**4.73**	**6.05**	**7.33**	**8.12**
2 (A68)	Exec. Time (s)	728.63	372.65	201.00	112.33	83.52	71.21	51.63	46.19
Branches (mln):	Ratio to Seq		***1.96***	***3.63***	***6.49***	***8.72***	***10.23***	***14.11***	***15.78***
802.31	Ratio to 1 PM		**1.00**	**1.85**	**3.32**	**4.46**	**5.23**	**7.22**	**8.07**
3 (A13)	Exec. Time (s)	782.61	410.75	211.30	118.89	85.21	79.62	58.66	47.23
Branches (mln):	Ratio to Seq		***1.91***	***3.70***	***6.58***	***9.18***	***9.83***	***13.34***	***16.57***
856.15	Ratio to 1 PM		**1.00**	**1.94**	**3.45**	**4.82**	**5.16**	**7.00**	**8.70**
4 (A88)	Exec. Time (s)	812.73	457.84	217.13	121.96	91.51	84.19	55.10	48.92
Branches (mln):	Ratio to Seq		***1.78***	***3.74***	***6.66***	***8.88***	***9.65***	***14.75***	16.61
898.74	Ratio to 1 PM		**1.00**	**2.11**	**3.75**	**5.00**	**5.44**	**8.31**	**9.36**
5 (A62)	Exec. Time (s)	833.84	423.90	220.64	120.87	95.26	83.36	57.88	51.68
Branches (mln):	Ratio to Seq		***1.97***	***3.78***	***6.90***	***8.75***	***10.00***	***14.41***	***16.13***
904.91	Ratio to 1 PM		**1.00**	**1.92**	**3.51**	**4.45**	**5.09**	**7.32**	**8.20**
6 (A73)	Exec. Time (s)	1099.85	578.71	297.23	155.17	111.92	97.58	68.71	59.10
Branches (mln):	Ratio to Seq		***1.90***	***3.70***	***7.09***	***9.83***	***11.27***	***16.01***	***18.61***
1 199.167	Ratio to 1 PM		**1.00**	**1.95**	**3.73**	**5.17**	**5.93**	**8.42**	**9.79**
7 (A52)	Exec. Time (s)	1160.73	557.99	293.32	156.87	120.87	92.00	67.15	62.34
Branches (mln):	Ratio to Seq		***2.08***	***3.96***	***7.40***	***9.60***	***12.62***	***17.28***	***18.62***
1 247.93	Ratio to 1 PM		**1.00**	**1.90**	**3.56**	**4.62**	**6.07**	**8.31**	**8.95**
8 (A32)	Exec. Time (s)	1218.52	616.86	308.85	170.32	121.14	103.42	69.23	60.69
Branches(mln):	Ratio to Seq		***1.98***	***3.95***	***7.15***	***10.06***	***11.78***	***17.60***	***20.08***
1 308.56	Ratio to 1 PM		**1.00**	**2.00**	**3.62**	**5.09**	**5.96**	**8.91**	**10.16**
9 (A90)	Exec. Time (s)	1361.54	691.69	351.73	184.57	135.65	122.36	77.83	64.71
Branches (mln):	Ratio to Seq		***1.97***	***3.87***	***7.38***	***10.04***	***11.13***	***17.49***	***21.04***
1 475.29	Ratio to 1 PM		**1.00**	**1.97**	**3.75**	**5.10**	**5.65**	**8.89**	**10.69**
10 (A35)	Exec. Time (s)	1531.67	749.91	388.59	206.69	146.86	113.03	86.31	78.30
Branches (mln):	Ratio to Seq		***2.04***	***3.94***	***7.41***	***10.43***	***13.55***	***17.75***	***19.56***
1 668.56	Ratio to 1 PM		**1.00**	**1.93**	**3.63**	**5.11**	**6.63**	**8.69**	**9.58**

Table 3. The Acceleration of the RP-algorithm on examples Gpi_Regular_400x50x80__1

Таблица 3. Ускорение РП-алгоритма на примерах Gpi_Regular_400x50x80__1

Source Data		Sequently	1 PM	2 PM	4 PM	6 PM	8 PM	12 PM	16 PM
1 (V05)	Exec. Time (s)	1333.11	727.86	372.66	188.56	139.69	110.83	78.00	66.59
Branches (mln):	Ratio to Seq		***1.83***	***3.58***	***7.07***	***9.54***	***12.03***	***17.09***	***20.02***
796.41	Ratio to 1 PM		**1.00**	**1.95**	**3.86**	**5.21**	**6.57**	**9.33**	**10.93**
2 (V03)	Exec. Time (s)	1344.30	740.92	375.28	193.61	137.74	116.74	82.62	66.09
Branches (mln):	Ratio to Seq		***1.81***	***3.58***	***6.94***	***9.76***	***11.52***	***16.27***	***20.34***
801.77	Ratio to 1 PM		**1.00**	**1.97**	**3.83**	**5.38**	**6.35**	**8.97**	**11.21**
3 (V08)	Exec. Time (s)	1354.75	750.12	379.48	203.57	164.28	121.22	82.53	66.22
Branches (mln):	Ratio to Seq		***1.81***	***3.57***	***6.65***	***8.25***	***11.18***	***16.42***	***20.46***
807.60	Ratio to 1 PM		**1.00**	**1.98**	**3.68**	**4.57**	**6.19**	**9.09**	**11.33**
4 (V01)	Exec. Time (s)	1319.81	696.24	357.22	191.56	137.92	124.07	81.89	66.55
Branches (mln):	Ratio to Seq		***1.90***	***3.69***	***6.89***	***9.57***	***10.64***	***16.12***	19.83
792.02	Ratio to 1 PM		**1.00**	**1.95**	**3.63**	**5.05**	**5.61**	**8.50**	**10.46**
5 (V07)	Exec. Time (s)	1386.27	738.58	367.57	193.48	138.88	109.58	81.70	64.77
Branches (mln):	Ratio to Seq		***1.88***	***3.77***	***7.17***	***9.98***	***12.65***	***16.97***	***21.40***
803.88	Ratio to 1 PM		**1.00**	**2.01**	**3.82**	**5.32**	**6.74**	**9.04**	**11.40**
6 (V00)	Exec. Time (s)	1343.85	732.08	366.71	200.75	140.48	123.63	81.51	65.26
Branches (mln):	Ratio to Seq		***1.84***	***3.66***	***6.69***	***9.57***	***10.87***	***16.49***	***20.59***
801.47	Ratio to 1 PM		**1.00**	**2.00**	**3.65**	**5.21**	**5.92**	**8.98**	**11.22**
7 (V04)	Exec. Time (s)	1346.08	731.88	368.80	192.22	136.93	121.54	80.59	67.68
Branches (mln):	Ratio to Seq		***1.84***	***3.65***	***7.00***	***9.83***	***11.08***	***16.70***	***19.89***
803.17	Ratio to 1 PM		**1.00**	**1.98**	**3.81**	**5.34**	**6.02**	**9.08**	**10.81**
8 (V06)	Exec. Time (s)	1352.78	728.87	368.90	190.83	135.72	123.45	82.09	69.15
Branches(mln):	Ratio to Seq		***1.86***	***3.67***	***7.09***	***9.97***	***10.96***	***16.48***	***19.56***
806.28	Ratio to 1 PM		**1.00**	**1.98**	**3.82**	**5.37**	**5.90**	**8.88**	**10.54**
9 (V02)	Exec. Time (s)	1486.93	766.66	406.07	220.99	152.59	127.02	84.95	70.14
Branches (mln):	Ratio to Seq		***1.94***	***3.66***	***6.73***	***9.74***	***11.71***	***17.50***	***21.20***
881.90	Ratio to 1 PM		**1.00**	**1.89**	**3.47**	**5.02**	**6.04**	**9.02**	**10.93**
10 (V09)	Exec. Time (s)	2757.84	1416.25	722.60	375.27	252.99	216.88	140.86	108.71
Branches (mln):	Ratio to Seq		***1.95***	***3.82***	***7.35***	***10.90***	***12.72***	***19.58***	***25.37***
1 631.54	Ratio to 1 PM		**1.00**	**1.96**	**3.77**	**5.60**	**6.53**	**10.05**	**13.03**

Table 4. The Acceleration of the RP-algorithm on examples Gpi_Random_300x0.4__50x0.4__2

Таблица 4. Ускорение РП-алгоритма на примерах Gpi_Random_300x0.4__50x0.4__2

Source Data		**Sequently**	**1 PM**	**2 PM**	**4 PM**	**6 PM**	**8 PM**	**12 PM**	**16 PM**
1 (V07)	Exec. Time (s)	597.71	326.64	176.33	99.33	75.61	62.60	49.74	43.26
Branches (mln):	Ratio to Seq		***1.83***	***3.39***	***6.02***	***7.91***	***9.55***	***12.02***	***13.82***
465.25	Ratio to 1 PM		**1.00**	**1.85**	**3.29**	**4.32**	**5.22**	**6.57**	**7.55**
2 (V01)	Exec. Time (s)	857.16	475.14	242.08	127.16	98.30	81.80	56.37	49.49
Branches (mln):	Ratio to Seq		***1.80***	***3.54***	***6.74***	***8.72***	***10.48***	***15.20***	***17.32***
668.94	Ratio to 1 PM		**1.00**	**1.96**	**3.74**	**4.83**	**5.81**	**8.43**	**9.60**
3 (V00)	Exec. Time (s)	891.87	472.17	252.58	133.73	97.87	84.28	58.61	55.11
Branches (mln):	Ratio to Seq		***1.89***	***3.53***	***6.67***	***9.11***	***10.58***	***15.22***	***16.18***
693.16	Ratio to 1 PM		**1.00**	**1.87**	**3.53**	**4.82**	**5.60**	**8.06**	**8.57**
4 (V02)	Exec. Time (s)	1283.61	672.86	354.47	179.13	129.66	112.08	77.30	60.59
Branches (mln):	Ratio to Seq		***1.91***	***3.62***	***7.17***	***9.90***	***11.45***	***16.61***	***21.19***
1 001.86	Ratio to 1 PM		**1.00**	**1.90**	**3.76**	**5.19**	**6.00**	**8.70**	**11.11**
5 (V03)	Exec. Time (s)	1393.92	715.05	381.60	197.44	139.02	139.02	81.60	64.96
Branches (mln):	Ratio to Seq		***1.95***	***3.65***	***7.06***	***10.03***	***10.03***	***17.08***	***21.46***
1 071.65	Ratio to 1 PM		**1.00**	**1.87**	**3.62**	**5.14**	**5.14**	**8.76**	**11.01**
6 (V04)	Exec. Time (s)	1615.98	854.22	440.69	227.83	161.15	137.82	91.46	71.15
Branches (mln):	Ratio to Seq		***1.89***	***3.67***	***7.09***	***10.03***	***11.73***	***17.67***	***22.71***
1 263.30	Ratio to 1 PM		**1.00**	**1.94**	**3.75**	**5.30**	**6.20**	**9.34**	**12.01**
7 (V08)	Exec. Time (s)	2507.05	1181.36	603.40	312.92	217.82	185.49	119.96	93.84
Branches (mln):	Ratio to Seq		***2.12***	***4.15***	***8.01***	***11.51***	***13.52***	***20.90***	***26.72***
1 800.63	Ratio to 1 PM		**1.00**	**1.96**	**3.78**	**5.42**	**6.37**	**9.85**	**12.59**
8 (V09)	Exec. Time (s)	2630.02	1349.70	683.68	344.68	237.55	227.27	130.56	106.76
Branches(mln):	Ratio to Seq		***1.95***	***3.85***	***7.63***	***11.07***	***11.57***	***20.14***	***24.63***
2 051.08	Ratio to 1 PM		**1.00**	**1.97**	**3.92**	**5.68**	**5.94**	**10.34**	**12.64**

References

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co, San Francisco, 1979.

[2] J. R. Ullmann, "An Algorithm for Subgraph Isomorphism", *Journal of the Association for Computing Machinery*, vol. 23, no. 1, pp. 31–42, 1976.

[3] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "Performance evaluation of the VF Graph Matching Algorithm", in *Proc. of the 10th ICIAP. IEEE Computer SocietyPress*, 1999, pp. 1172–1177.

[4] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "An Improved Algorithm for Matching Large Graphs", in *Proc. of the 3rd IAPR-TC-15 International Workshop on Graph-based Representations – Italy*, 2001, pp. 149–159.

[5] V. Carletti, P. Foggia, A. Saggese, and M. Vento, "Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 804–818, 2018.

[6] M. B. Il'yashenko, "Unificirovannyj podhod k resheniyu zadach morfizma na grafah", *Elektronnoe modelirovanie*, vol. 30, no. 1, pp. 19–41, 2008.

[7] M. B. Il'yashenko, "Reshenie zadachi poiska graf-podgraf izomorfizma dlya semanticheskogo analiza specializirovannyh cifrovyh sistem", *Nauchnye trudy DonTU. Seriya "Informatika, kibernetika i vychislitel'naya tekhnika"*, vol. 16, pp. 46–57, 2012.

[8] J. R. Ullmann, "Bit-Vector Algorithms for Binary Constraint Satisfaction and Subgraph Isomorphism", *Journal of Experimental Algorithmics*, vol. 15, no. 1, pp. 1–64, 2011.

[9] M. B. Il'yashenko, "Razrabotka i issledovanie parallel'nogo algoritma proverki graf-podgraf izomorfizma", *Radioelektronika. Informatika. Upravlenie*, vol. 1, pp. 63–69, 2006.

[10] V. Carletti, P. Foggia, P. Ritrovato, M. Vento, and V. Vigilante, "A parallel Algorithm for Subgraph Isomorphism", in *International Workshop on Graph-Based Representations in Pattern Recognition*, ser. LNCS, vol. 11510, Springer, Cham, 2019, pp. 141–151.

[11] V. V. Vasilchikov, *Sredstva parallelnogo programmirovaniya dlya vychislitelnykh sistem s dinamicheskoy balansirovkoy zagruzki*. YarGU, Yaroslavl, 2001.

[12] V. V. Vasilchikov, *Kommunikatsionnyy modul dlya organizatsii polnosvyaznogo soedineniya kompyuterov v lokalnoy seti s ispolzovaniem .NET Framework*, 2013.

[13] V. V. Vasilchikov, *Biblioteka podderzhki rekursivno-parallelnogo programmirovaniya dlya .NET Framework*, 2013.

[14] V. V. Vasilchikov, "On the recursive-parallel programming for the. NET framework", *Automatic Control and Computer Sciences*, vol. 48, no. 7, pp. 575–580, 2014.

[15] V. V. Vasilchikov, "On optimization and parallelization of the Little algorithm for solving the travelling salesman problem", *Automatic Control and Computer Sciences*, vol. 51, no. 7, pp. 551–557, 2017.

[16] V. V. Vasilchikov, "On a recursive-parallel algorithm for solving the knapsack problem", *Automatic Control and Computer Sciences*, vol. 52, no. 7, pp. 810–816, 2018.

[17] V. V. Vasilchikov, "Parallel algorithm for solving the graph isomorphism problem", *Modeling and analysis of information systems*, vol. 27, no. 1, pp. 86–94, 2020.

[18] P. Foggia, C. Sansone, and M. Vento, *A Database of Graphs for Isomorphism and Sub-Graph Isomorphism Benchmarkin*, 2001. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.6803&rep=rep1&type=pdf.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 29, NO. 1, 2022
journal homepage: www.mais-journal.ru

THEORY OF COMPUTING

LTL-Specification of Bounded Counter Machines

E. V. Kuzmin[1]

DOI: 10.18255/1818-1015-2022-1-44-59

[1]P. G. Demidov Yaroslavl State University, 14 Sovetskaya str., Yaroslavl 150003, Russia.

MSC2020: 68Q60, 68N30, 68Q04, 03B44, 03B70, 03D10
Research article
Full text in Russian

Received January 11, 2022
After revision February 21, 2022
Accepted March 9, 2022

The article revises the results of the work devoted to the problem of representing the behaviour of a program system as a set of formulas of the linear temporal logic LTL, followed by the use of this representation to verify the satisfiability of the program system properties through the procedure of proving the validity of logical inferences, expressed in terms of the LTL logic. The LTL logic is applied to bounded Minsky counter machines that are considered as program systems of which we need to get the specification of its behaviour. Earlier, when working with the temporal logic LTL as a program logic, a special pseudo-operator was actually introduced to refer to the previous values of variables involved in elementary propositions. Despite the fact that this pseudo-operator is easily implemented in the Cadence SMV verifier when proving the validity of logical LTL-inferences, the classical definition of the LTL logic does not imply its presence. In this article, only binary variables will be used to build an LTL-specification for the behaviour of a bounded counter machine, and tracking of previous values of these variables will be carried out exclusively within the LTL logic itself through the appropriate formulas.

Keywords: non-classical logic; linear temporal logic; counter machines; LTL-specification

INFORMATION ABOUT THE AUTHORS

Egor V. Kuzmin (correspondence author) | orcid.org/0000-0003-0500-306X. E-mail: kuzmin@uniyar.ac.ru
Professor, Doctor of Science.

Funding: This work was supported by P. G. Demidov Yaroslavl State University Project № VIP-016.

For citation: E. V. Kuzmin, “LTL-Specification of Bounded Counter Machines”, *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 44-59, 2022.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 29, № 1, 2022
сайт журнала: www.mais-journal.ru

THEORY OF COMPUTING

LTL-спецификация ограниченных счётчиковых машин

Е. В. Кузьмин[1]

DOI: 10.18255/1818-1015-2022-1-44-59

[1]Ярославский государственный университет им. П. Г. Демидова, ул. Советская, д. 14, г. Ярославль, 150003 Россия.

УДК 519.7
Научная статья
Полный текст на русском языке

Получена 11 января 2022 г.
После доработки 21 февраля 2022 г.
Принята к публикации 9 марта 2022 г.

В статье пересматриваются результаты работы, посвящённой задаче представления поведения программной системы в виде набора формул линейной темпоральной логики LTL с последующим использованием этого представления для проверки выполнимости программных свойств системы через процедуру доказательства справедливости логических выводов, выраженных в терминах логики LTL. В качестве программных систем, для спецификации поведения которых применяется логика LTL, рассматриваются счётчиковые машины Минского с ограничениями. Ранее при работе с темпоральной логикой LTL как с программной логикой фактически был введён специальный псевдооператор обращения к предыдущим значениям переменных, задействованных в элементарных высказываниях. Несмотря на то что этот псевдооператор легко реализуется в верификаторе Cadence SMV при доказательстве справедливости логических LTL-выводов, классическое определение логики LTL не предполагает его наличия. В данной статье для построения LTL-спецификации поведения ограниченной счётчиковой машины будут использоваться только бинарные переменные, а отслеживание их предыдущих значений будет осуществляться исключительно в рамках самой логики LTL посредством соответствующих формул.

Ключевые слова: неклассическая логика; линейная темпоральная логика; счётчиковые машины; LTL-спецификация

ИНФОРМАЦИЯ ОБ АВТОРАХ

Егор Владимирович Кузьмин
автор для корреспонденции

orcid.org/0000-0003-0500-306X. E-mail: kuzmin@uniyar.ac.ru
профессор, доктор физ.-мат. наук.

Финансирование: Работа выполнена в рамках инициативной НИР ЯрГУ им. П. Г. Демидова № VIP-016.

Для цитирования: E. V. Kuzmin, “LTL-Specification of Bounded Counter Machines”, *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 44-59, 2022.

Введение

В статье пересматриваются результаты работы [1], посвящённой задаче представления поведения программной системы в виде набора формул линейной темпоральной логики LTL с последующим использованием этого представления для проверки выполнимости программных свойств системы через процедуру доказательства справедливости логических выводов, выраженных в терминах логики LTL. В качестве программных систем, для спецификации поведения которых применяется логика LTL, рассматриваются счётчиковые машины Минского [2—4] с ограничениями.

Ранее в [1] при работе с темпоральной логикой LTL как с программной логикой фактически был введён специальный псевдооператор обращения к предыдущим значениям переменных, задействованных в элементарных высказываниях. Несмотря на то что этот псевдооператор легко реализуется в верификаторе Cadence SMV [5] при доказательстве справедливости логических LTL-выводов, классическое определение логики LTL не предполагает его наличия.

В данной статье для построения LTL-спецификации поведения ограниченной счётчиковой машины будут использоваться только бинарные переменные, а отслеживание их предыдущих значений будет осуществляться исключительно в рамках самой логики LTL посредством соответствующих формул.

1. Ограниченные счётчиковые машины

Ограниченная счётчиковая машина представляет собой счётчиковую машину Минского [2—4], дополненную ограничениями на ёмкости счётчиков. Более формально, *ограниченная счётчиковая машина* — это набор из шести элементов $(q_0, q_n, Q, V, \Delta, \mathrm{bnd})$, где $Q = \{q_0, \dots, q_n\}$ — конечное непустое множество состояний машины; $q_0 \in Q$ — начальное состояние; $q_n \in Q$ — финальное состояние; $X = \{x_1, \dots, x_m\}$ — конечное непустое множество счётчиков, которые могут принимать значения из $\mathbb{N} \cup \{0\}$; $\Delta = \{\delta_0, \dots, \delta_{n-1}\}$ — набор правил переходов по состояниям машины; δ_i — правило переходов для состояния q_i; $\mathrm{bnd}\colon X \longrightarrow \mathbb{N}$ — отображение, устанавливающее предельное значение $\mathrm{bnd}x \in \mathbb{N}$ для каждого счётчика $x \in X$. Состояния q_i, $0 \leqslant i \leqslant n-1$, подразделяются на два типа. Состояния первого типа имеют правила переходов вида:

$$(\delta_i)\ \ q_i:\ \ \text{if } x_j < \mathrm{bnd}x_j \text{ then } x_j := x_j + 1;\ \text{goto } q_k,$$

где $1 \leqslant j \leqslant m, 0 \leqslant k \leqslant n$. Для состояний второго типа имеем, $1 \leqslant j \leqslant m, 0 \leqslant k, l \leqslant n$:

$$(\delta_i)\ \ q_i:\ \ \text{if } x_j > 0 \text{ then } (x_j := x_j - 1;\ \text{goto } q_k) \text{ else goto } q_l.$$

Для финального состояния q_n правил переходов не предусмотрено. Это означает, что при попадании в состояние q_n ограниченная счётчиковая машина завершает свою работу. Если при срабатывании правила переходов первого типа оказывается, что значение соответствующего счётчика достигло предельного значения, работа машины также останавливается.

Конфигурация ограниченной счётчиковой машины — это набор $(q_i, c_1, \dots, c_m)$, где q_i — состояние машины, $c_1, \dots, c_m$ — натуральные числа (включая ноль), являющиеся значениями соответствующих счётчиков.

Рассмотрим множество состояний $Q = \{q_0, \dots, q_n\}$ как набор булевых переменных. Будем считать, что некоторая переменная $q_i \in Q$ принимает значение 1, если счётчиковая машина находится в состоянии q_i $(0 \leqslant i \leqslant n)$. В других случаях q_i будет иметь значение 0. Тогда конфигурацию сётчиковой машины можно представить в виде вектора значений соответствующего набора переменных

$$(q_0, \dots, q_n, x_1, \dots, x_m).$$

Исполнением счётчиковой машины называется последовательность конфигураций $s_0\ s_1\ s_2\ s_3\ s_4\ \dots$, индуктивно определяемая в соответствии с правилами переходов. Счётчиковая машина имеет одно

исполнение из начальной конфигурации s_0, так как для каждого состояния предусмотрено не более одного правила переходов. Машина, получив на вход некоторый набор значений счётчиков, стартует из состояния q_0 и либо останавливается в состоянии q_n с выходным набором значений счётчиков, либо прекращает работу в одном из нефинальных состояний из-за достижения счётчиком своего предельного значения, либо зацикливается, реализуя тем самым частичную числовую функцию.

Далее в статье под счётчиковой машиной будет пониматься именно ограниченная счётчиковая машина. При небольшом количестве счётчиков будем обозначать их буквами a, b, c и d.

2. Примеры ограниченных счётчиковых машин

Рассмотрим в качестве примеров счётчиковую машину умножения двух чисел и счётчиковую машину возведения числа в квадрат [3].

Ограниченная счётчиковая машина $4cM$ умножения двух натуральных чисел имеет семь состояний $q_0, q_1, \ldots, q_6$, где q_0 является начальным состоянием, q_6 — финальное состояние. Множество счётчиков $X = \{a, b, c, d\}$. В начальной конфигурации счётчики a и b получают значения n и m соответственно, а начальные значения остальных счётчиков c и d равны нулю. В финальной конфигурации результат вычисления будет содержаться в счётчике c при нулевых значениях счётчиков a и d. Счётчик b будет содержать исходное число m. На значения счётчиков накладываются ограничения bnd$a$, bnd$b$, bnd$c$ и bnd$d$. Правила переходов по состояниям машины $4cM$ представлены ниже:

(δ_0) q_0 : if $a > 0$ then ($a := a - 1$; goto q_1) else goto q_6;
(δ_1) q_1 : if $b > 0$ then ($b := b - 1$; goto q_2) else goto q_4;
(δ_2) q_2 : if $c <$ bnd$c$ then $c := c + 1$; goto q_3;
(δ_3) q_3 : if $d <$ bnd$d$ then $d := d + 1$; goto q_1;
(δ_4) q_4 : if $d > 0$ then ($d := d - 1$; goto q_5) else goto q_0;
(δ_5) q_5 : if $b <$ bnd$b$ then $b := b + 1$; goto q_4.

Четырёхсчётчиковая машина $4cM$ в графическом виде показана на рис. 1, где для переменной $x \in X$ обозначение «$x+$» соответствует увеличению счётчика на единицу с последующим переходом к новому состоянию машины вниз или по стрелке, а «$x-$» используется для обозначения условного вычитания единицы с переходом вниз при $x > 0$ или по правосторонней стрелке в случае нулевого значения счётчика x.

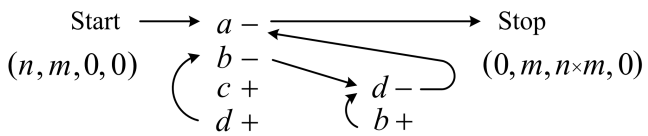

Fig. 1. Graphical representation of the counter machine $4cM$ of multiplying two numbers

Рис. 1. Графическое представление счётчиковой машины $4cM$ умножения двух чисел

Ограниченная счётчиковая машина $3cM$ возведения числа n в квадрат имеет восемь состояний $q_0, q_1, \ldots, q_7$, где q_0 — начальное состояние, q_7 — финальное состояние, и три счётчика a, b и c. В начальной конфигурации счётчик a получает значение n, а начальные значения двух других счётчиков b и c равны нулю. В финальной конфигурации результат вычисления будет содержаться в счётчике c при нулевых значениях остальных счётчиков a и b. На значения счётчиков накладываются ограничения bnd$a$, bnd$b$ и bnd$c$. Машина $3cM$ имеет следующие правила переходов:

(δ_0) q_0 : if $a > 0$ then $(a := a - 1;$ goto $q_1)$ else goto q_7;
(δ_1) q_1 : if $c <$ bnd$c$ then $c := c + 1;$ goto q_2;
(δ_2) q_2 : if $a > 0$ then $(a := a - 1;$ goto $q_3)$ else goto q_5;
(δ_3) q_3 : if $b <$ bnd$b$ then $b := b + 1;$ goto q_4;
(δ_4) q_4 : if $c <$ bnd$c$ then $c := c + 1;$ goto q_1;
(δ_5) q_5 : if $b > 0$ then $(b := b - 1;$ goto $q_6)$ else goto q_0;
(δ_6) q_6 : if $a <$ bnd$a$ then $a := a + 1;$ goto q_5.

Правила переходов счётчиковой машины $3cM$ в графическом виде представлены на рис. 2.

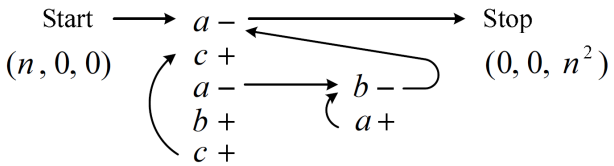

Fig. 2. Graphical representation of the counter machine $3cM$ of squaring a number

Рис. 2. Графическое представление счётчиковой машины $3cM$ возведения числа в квадрат

3. Линейная темпоральная логика

Линейная темпоральная логика LTL (linear temporal logic), или темпоральная логика линейного времени (linear-time temporal logic), представляет собой расширение классической логики высказываний с помощью модальных операторов, позволяющих учитывать временной аспект в последовательностях событий и объектов. Темпоральная логика LTL находит важное применение в области формальной верификации, где она используется для описания требований к аппаратным и программным системам [6]. В данной статье формализм логики LTL будет задействован для описания поведения счётчиковых машин, а также спецификации их свойств, подлежащих дальнейшему анализу на выполнимость методом проверки модели (model checking) [7, 8].

Дадим определение логики LTL, как некоторой абстрактной модальной темпоральной логики, являющейся одним из представителей внушительного ряда неклассических логик [9, 10].

Символами языка логики LTL являются пропозициональные переменные p, p_0, p_1, p_2, ... (элементарные высказывания), константы true и false, логические связки $\neg$ (отрицание), $\wedge$ (конъюнкция), $\vee$ (дизъюнкция), $\Longrightarrow$ (импликация), $\Longleftrightarrow$ (эквивалентность), темпоральные модальные операторы **X** (neXt, непосредственное следование), **U** (Until, условное ожидание), **F** (Future, неизбежность), **G** (Globally, инвариантность) и знаки пунктуации «(» и «)».

Формулами языка логики LTL являются те и только те строки символов, которые могут быть рекурсивно построены из пропозициональных переменных и констант по следующему правилу:

если φ и ψ — формулы, то выражения $\neg\varphi$, $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, $(\varphi \Longrightarrow \psi)$, $(\varphi \Longleftrightarrow \psi)$, $\mathbf{X}\,\varphi$, $(\varphi\,\mathbf{U}\,\psi)$, $\mathbf{F}\,\varphi$ и $\mathbf{G}\,\varphi$ также являются формулами.

Значение (истинность или ложность) формул темпоральной логики LTL определяется через понятие интерпретации.

Интерпретация формул логики LTL представляет собой набор (w_0, W, T, v), где W обозначает некоторое непустое множество объектов, интуитивно понимаемое как множество *возможных миров*, $w_0 \in W$ — это начальный мир, а $T : W \longrightarrow W$ — функция переходов между мирами.

Если возможные миры w и w' принадлежат W и связаны функцией переходов $T(w) = w'$, то для простоты будем писать $w \longrightarrow w'$. Запись $w \longrightarrow w'$ читается как «мир w' непосредственно достижим из мира w» и означает, что существует прямой переход из мира w в мир w'.

Обозначим $w \xrightarrow{*} w'$ транзитивную достижимость мира w' из мира w за ноль или более шагов (применений функции T), т. е. $w \xrightarrow{*} w'$ означает, что $\exists\, i \in \mathbb{N} \cup \{0\}$: $T^i(w) = w'$, где $T^i(w) = T(T^{i-1}(w))$ и $T^0(w) = w$. Запись $w \xrightarrow{i} w'$ будет обозначать достижимость w' из w ровно за i шагов. В случае $i = 0$ считается, что мир w транзитивно достигает сам себя за ноль шагов, т. е. $w' = w$.

Функция ν сопоставляет истинное (возвращается результат 1) или ложное (результат 0) значение каждой паре вида (p, w), где p — пропозициональная переменная, а $w \in W$. Запишем это как $\nu_w(p) = 1$ и $\nu_w(p) = 0$ соответственно.

Интуитивно, $\nu_w(p) = 1$ означает, что в мире w высказывание p истинно, а $\nu_w(p) = 0$ — в мире w высказывание p ложно.

Функция ν_w для каждого мира $w \in W$ расширяется на все множество формул по рекурсивным правилам. Рекурсивные правила расширения функции ν_w для логических связок $\neg$, $\wedge$, $\vee$, $\Longrightarrow$ и $\Longleftrightarrow$ следующие. Для каждого мира $w \in W$ имеем:

$\nu_w(\neg\varphi) = 1$, если $\nu_w(\varphi) = 0$, иначе $\nu_w(\neg\varphi) = 0$;

$\nu_w(\varphi \wedge \psi) = 1$, если $\nu_w(\varphi) = \nu_w(\psi) = 1$, иначе $\nu_w(\varphi \wedge \psi) = 0$;

$\nu_w(\varphi \vee \psi) = 1$, если $\nu_w(\varphi) = 1$ или $\nu_w(\psi) = 1$, иначе $\nu_w(\varphi \vee \psi) = 0$;

$\nu_w(\varphi \Longrightarrow \psi) = 1$, если $\nu_w(\varphi) = 0$ или $\nu_w(\psi) = 1$, иначе $\nu_w(\varphi \Longrightarrow \psi) = 0$;

$\nu_w(\varphi \Longleftrightarrow \psi) = 1$, если $\nu_w(\varphi) = \nu_w(\psi)$, иначе $\nu_w(\varphi \Longleftrightarrow \psi) = 0$.

Из этих правил видно, что классические логические связки действуют в рамках одного текущего мира w.

Для темпоральных модальных операторов логики LTL рекурсивные правила расширения функции ν_w выглядят следующим образом. Для каждого мира $w \in W$ имеем:

$\nu_w(\mathbf{X}\,\varphi) = 1$, если $\exists\, w' \in W$ такой, что $w \longrightarrow w'$ и $\nu_{w'}(\varphi) = 1$, иначе $\nu_w(\mathbf{X}\,\varphi) = 0$,

т. е. формула φ должна выполняться в следующем достижимом мире;

$\nu_w(\mathbf{F}\,\varphi) = 1$, если $\exists\, w' \in W$ такой, что $w \xrightarrow{*} w'$ и $\nu_{w'}(\varphi) = 1$, иначе $\nu_w(\mathbf{F}\,\varphi) = 0$,

т. е. формула φ должна выполняться в некотором мире w', который транзитивно достижим из текущего мира w;

$\nu_w(\mathbf{G}\,\varphi) = 1$, если $\forall\, w' \in W$ такого, что $w \xrightarrow{*} w'$, имеем $\nu_{w'}(\varphi) = 1$, иначе $\nu_w(\mathbf{G}\,\varphi) = 0$,

т. е. формула φ должна выполняться в текущем мире w и во всех мирах w', которые транзитивно достижимы из мира w;

$\nu_w(\varphi\,\mathbf{U}\,\psi) = 1$, если $\exists\, w' \in W$ такой, что для некоторого $i \in \mathbb{N} \cup \{0\}$ имеем $w \xrightarrow{i} w'$ и $\nu_{w'}(\psi) = 1$, а $\forall\, w'' \in W$ такого, что $\exists\, j, j < i$: $w \xrightarrow{j} w''$, выполняется $\nu_{w''}(\varphi) = 1$, иначе $\nu_w(\varphi\,\mathbf{U}\,\psi) = 0$,

т. е. формула ψ должна выполняться в некотором мире w', который транзитивно достижим из текущего мира w, но при этом во всех мирах, которые предшествуют миру w' на пути из w, должна выполняться формула φ.

В логике LTL операторы **F** и **G** являются двойственными: $\mathbf{G}\,\varphi = \neg\mathbf{F}\neg\varphi$. При этом сам оператор **F** представляет собой специальный случай применения оператора **U**, а именно $\mathbf{F}\varphi = \text{true}\,\mathbf{U}\,\varphi$.

Отметим, что каждую интерпретацию в логике LTL можно представить в виде бесконечной последовательности миров (с начальным миром w_0), связанных между собой функцией переходов, со своей оценочной функцией ν.

Для обозначения того, что из набора LTL-формул φ_1, φ_2, …, φ_m (предпосылок), где $m \in \mathbb{N}$, логически выводится формула ψ (заключение), используется металингвистический символ «$\models$». Вывод в логике LTL является справедливым, если он сохраняет свою истинность в начальных мирах всех интерпретаций. Пусть Σ — произвольное множество формул (набор предпосылок). Тогда заключение ψ будет *логически следовать* из набора формул Σ, т. е. $\Sigma \models \psi$, тогда и только тогда, когда не существует такой интерпретации, при которой все формулы из Σ в начальном мире являются истинными, а формула ψ в нём ложна. Другими словами, каждая интерпретация, обращающая в начальном мире все формулы из Σ в истину, в этом же мире обращает в истину и формулу ψ.

Более формально, логический вывод $\Sigma \models \psi$ справедлив тогда и только тогда, когда при всех возможных интерпретациях (w_0, W, T, v) для мира $w_0 \in W$ выполняется условие, что если $v_{w_0}(\varphi) = 1$ для всех предпосылок $\varphi \in \Sigma$, то $v_{w_0}(\psi) = 1$.

Запись $\Sigma \not\models \psi$ означает, что условие $\Sigma \models \psi$ не выполняется. В этом случае будет существовать хотя бы одна интерпретация (w_0, W, T, v), в рамках которой в начальном мире w_0 все формулы из Σ выполняются, а формула ψ в мире w_0 является ложной. Такая интерпретация называется контрпримером, или контр-моделью.

4. LTL как программная логика

Пусть $\mathrm{Cl}(mcM)$ — это класс ограниченных m-счётчиковых машин с $n+1$ булевыми переменными-состояниями $q_0, q_1, q_2, \ldots, q_n$ и соответствующими ограничениями $\mathrm{bnd}x_1, \mathrm{bnd}x_2, \ldots, \mathrm{bnd}x_m$ для счётчиков $x_1, x_2, \ldots, x_m$. Тогда применительно к этому классу m-счётчиковых машин $\mathrm{Cl}(mcM)$ линейная темпоральная логика LTL может быть рассмотрена следующим образом.

Сопоставим возможному миру $w \in W$ логики LTL конфигурацию некоторой счётчиковой машины из класса $\mathrm{Cl}(mcM)$ на некотором шаге её *исполнения*. Тогда каждый возможный мир будет взаимно однозначно соответствовать вектору значений переменных $(q_0, q_1, q_2, \ldots, q_n, x_1, x_2, \ldots, x_m)$, т. е. миры с одинаковым набором значений указанных переменных считаются одним миром.

Таким образом, для класса $\mathrm{Cl}(mcM)$ ограниченных m-счётчиковых машин множество различных миров W в каждой интерпретации (w_0, W, T, v) логики LTL будет конечным. Этот факт обеспечивает возможность проверки справедливости всех логических выводов в рамках логики LTL, рассмотренной применительно к классу счётчиковых машин $\mathrm{Cl}(mcM)$.

Непосредственную достижимость $w \longrightarrow w'$ возможного мира w' из мира w будем понимать как осуществление перехода из одной конфигурации счётчиковой машины в другую после срабатывания одного из правил переходов. Если для текущей конфигурации ни одно из правил переходов не выполняется, то предполагается, что происходит «пустой» переход в мир w', который будет соответствовать прежней конфигурации счётчиковой машины, т. е. $w' = w$.

В качестве элементарного высказывания логики LTL будем рассматривать выражение над переменными, построенное с использованием операторов сравнения, арифметических операторов и констант (целых неотрицательных чисел). Элементарное высказывание p формулируется таким образом, чтобы в результате оно могло принимать только два логических значения 1 (истина) или 0 (ложь). Результат оценочной функции $v_w(p)$ для элементарного высказывания p и мира w будет зависеть от значений задействованных в p переменных, которые они имеют в мире w.

Примером простого элементарного высказывания является выражение в виде одной булевой переменной-состояния.

Теперь для любой счётчиковой машины M из класса $\mathrm{Cl}(mcM)$ может быть построен такой набор LTL-формул, который будет описывать в точности все возможные исполнения счётчиковой машины M и только их. Всякий раз когда все LTL-формулы из набора будут одновременно обращаться в 1 (иметь значение «истина») при некоторой интерпретации (w_0, W, T, v), это будет означать, что интерпретация (w_0, W, T, v) соответствует одному из возможных исполнений машины M,

т. е. повторяет последовательность конфигураций в исполнении. И наоборот, для любого исполнения счётчиковой машины M будет существовать повторяющая его интерпретация, которая будет приводить к одновременному выполнению всех LTL-формул из рассматриваемого набора.

Пусть набор LTL-формул $\Sigma = \{\varphi_1, \dots, \varphi_k\}$, где $k \in \mathbb{N}$, описывает все возможные исполнения некоторой счётчиковой машины M из класса $\mathrm{Cl}(mcM)$, у которой q_0 — это начальное состояние, а состояние q_n является финальным. Рассмотрим LTL-формулу $\psi = «q_0 \Longrightarrow \mathbf{F}\,\mathbf{G}\,q_n»$. Эта формула является истинной только для тех интерпретаций, которые в начальном мире имеют $q_0 = 0$ или же если в начальном мире выполняется $q_0 = 1$, то в последовательности миров этих интерпретаций рано или поздно появится мир, начиная с которого для всех следующих миров будет выполняться $q_n = 1$. Тогда справедливость логического вывода $\varphi_1, \dots, \varphi_k \vDash \psi$ будет означать, что счётчиковая машина M, стартуя из состояния q_0 при любых начальных значениях счётчиков, обязательно завершит свою работу в финальном состоянии q_n, т. е. в результате конечной последовательности сработавших правил переходов машина окажется в финальном состоянии.

Если же имеет место $\varphi_1, \dots, \varphi_k \nvDash \psi$, то существует интерпретация (контрпример), соответствующая такому исполнению счётчиковой машины, которое для некоторых входных данных не приводит к требуемому результату вычислений, поскольку не попадает в финальное состояние. При этом набор формул $\varphi_1, \dots, \varphi_k, \neg\psi$ будет описывать все интерпретации, являющиеся контрпримерами, т. е. те интерпретации, для которых все формулы из этого набора одновременно являются истинными.

Получили, что задача проверки выполнимости для счётчиковой машины некоторого свойства, заданного LTL-формулой, может быть сведена к задаче доказательства справедливости логического вывода в рамках линейной темпоральной логики LTL.

В следующих разделах будет показано, каким образом поведение счётчиковых машин может быть представлено в виде набора LTL-формулы, а также будут рассмотрены примеры LTL-свойств счётчиковых машин.

5. LTL-спецификация ограниченной счётчиковой машины

Основная идея представления поведения ограниченной счётчиковой машины в виде набора LTL-формул состоит в описании (на языке логики LTL) того, каким образом происходит изменение значения каждой переменной счётчиковой машины на каждом шаге её исполнения.

При этом каждый счётчик $x_j \in X$, $1 \leqslant j \leqslant m$, задаётся своим набором из k бинарных переменных $x_j^{k-1}, x_j^{k-2}, \dots, x_j^1, x_j^0$ следующим образом:

$$x_j = \sum_{i=0}^{k-1} x_j^i 2^i = (x_j^{k-1} x_j^{k-2} \dots x_j^1 x_j^0)_2, \quad \text{где} \quad k = \lceil \log_2(\mathrm{bnd} x_j + 1) \rceil,$$

$\lceil\ \rceil$ — функция округления до большего целого числа.

Изменение значения бинарной переменной, соответствующей состоянию машины или одной из переменных счётчика, задаётся с помощью трёх LTL-формул. Первая LTL-формула описывает ситуации, при которых происходит возрастание значения соответствующей переменной, вторая LTL-формула определяет условия, приводящие к уменьшению значения переменной. Третья LTL-формула имеет жёсткую конструкцию, составленную из элементов первых двух формул. Она описывает условия, при которых рассматриваемая переменная не меняет своего значения во время работы счётчиковой машины.

Для спецификации поведения состояния q используются LTL-формулы вида:

$\mathbf{G}\,(\,\neg q \wedge\ \mathbf{X}(q)\ \Longrightarrow\ \mathit{FiringCondA}\,)$;

$\mathbf{G}\,(\ \ q \wedge \neg\mathbf{X}(q)\ \Longrightarrow\ \mathit{FiringCondD}\,)$.

Первая формула означает, что всякий раз, когда состояние q счётчиковой машины активируется, из этого следует, что было выполнено условие $FiringCondA$ перехода в состояние q, т. е. сработало некоторое правило переходов. Выражение $FiringCondD$ во второй формуле описывает условия выхода из состояния q.

Третья LTL-формула, описывающая условия сохранения прежнего состояния q, имеет вид

$$\mathbf{G}\,(\,(q \Longleftrightarrow \mathbf{X}(q)) \implies \neg(\neg q \wedge FiringCondA) \wedge \neg(q \wedge FiringCondD)\,).$$

Для спецификации поведения переменной x_j^i представления счётчика x_j используются LTL-формулы следующего вида:

$$\begin{aligned}
&\mathbf{G}\,(\,\neg x_j^i \wedge \ \mathbf{X}(x_j^i) \implies FiringCondInc \wedge x_j < \mathrm{bnd}x_j \wedge \ x_j^{i-1} \wedge \ x_j^{i-2} \wedge \cdots \wedge \ x_j^1 \wedge \ x_j^0 \ \vee \\
&\qquad\qquad FiringCondDec \wedge x_j > 0 \qquad \wedge \neg x_j^{i-1} \wedge \neg x_j^{i-2} \wedge \cdots \wedge \neg x_j^1 \wedge \neg x_j^0\ );\\
&\mathbf{G}\,(\ \ x_j^i \wedge \neg\mathbf{X}(x_j^i) \implies FiringCondInc \wedge x_j < \mathrm{bnd}x_j \wedge \ x_j^{i-1} \wedge \ x_j^{i-2} \wedge \cdots \wedge \ x_j^1 \wedge \ x_j^0 \ \vee \\
&\qquad\qquad FiringCondDec \wedge x_j > 0 \qquad \wedge \neg x_j^{i-1} \wedge \neg x_j^{i-2} \wedge \cdots \wedge \neg x_j^1 \wedge \neg x_j^0\ );\\
&\mathbf{G}\,(\,(x_j^i \Longleftrightarrow \mathbf{X}(x_j^i)) \implies \neg(FiringCondInc \wedge x_j < \mathrm{bnd}x_j \wedge \ x_j^{i-1} \wedge \ x_j^{i-2} \wedge \cdots \wedge \ x_j^1 \wedge \ x_j^0 \ \vee \\
&\qquad\qquad FiringCondDec \wedge x_j > 0 \qquad \wedge \neg x_j^{i-1} \wedge \neg x_j^{i-2} \wedge \cdots \wedge \neg x_j^1 \wedge \neg x_j^0)\ ).
\end{aligned}$$

Здесь выражения $FiringCondInc$ и $FiringCondDec$ представляют собой дизъюнкции состояний счётчиковой машины, при которых возникает необходимость изменения значения переменной x_j, если, конечно, это допускается соответствующими условиями $x_j < \mathrm{bnd}x_j$ и $x_j > 0$, где $x_j = \sum_{i=0}^{k-1} x_j^i 2^i$ и $k = \lceil \log_2(\mathrm{bnd}x_j + 1) \rceil$. А именно, $FiringCondInc = (q_p \vee \cdots \vee q_r)$, где $q_p, \ldots, q_r$ — состояния первого типа, в правилах переходов которых участвует счётчик x_j, т. е. происходит увеличение счётчика x_j на единицу при условии, что $x_j < \mathrm{bnd}x_j$. И $FiringCondDec = (q_s \vee \cdots \vee q_t)$, где $q_s, \ldots, q_t$ — состояния второго типа с правилами переходов, в которых значение счётчика x_j уменьшается на единицу при условии $x_j > 0$.

Если в описании машины для счётчика x_j правил переходов первого типа нет, то для переменной x_j^i имеем $FiringCondInc = \mathrm{false}$. Аналогичным образом, если для счётчика x_j нет правил переходов второго типа, то $FiringCondDec = \mathrm{false}$.

Заметим, что структура LTL-формул спецификации поведения переменной x_j^i позволяет заменить их одной LTL-формулой вида

$$\begin{aligned}
&\mathbf{G}\,(\ \mathbf{X}(x_j^i) \ \Longleftrightarrow \ \neg(\,x_j^i \Longleftrightarrow (FiringCondInc \wedge x_j < \mathrm{bnd}x_j \wedge \ x_j^{i-1} \wedge \ x_j^{i-2} \wedge \cdots \wedge \ x_j^1 \wedge \ x_j^0 \ \vee \\
&\qquad\qquad FiringCondDec \wedge x_j > 0 \qquad \wedge \neg x_j^{i-1} \wedge \neg x_j^{i-2} \wedge \cdots \wedge \neg x_j^1 \wedge \neg x_j^0))\ ).
\end{aligned}$$

Опираясь на формальное определение ограниченной счётчиковой машины, опишем более подробно схему построения LTL-спецификации изменения значения переменной-состояния.

LTL-формула, учитывающая условия, при которых счётчиковая машина переходит из некоторого текущего состояния в новое состояние q_k, т. е. логическая переменная q_k получает значение 1, имеет следующий вид:

$$\mathbf{G}\,(\ \neg q_k \wedge \mathbf{X}(q_k) \implies q_i \wedge x_j > 0 \vee \cdots \vee q_r \wedge \neg(x_t > 0) \vee \cdots \vee q_s \wedge x_l < \mathrm{bnd}x_l\ ),$$

где условия вида $q_i \wedge x_j > 0$ соответствуют правилам переходов второго типа

$$(\delta_i)\ \ q_i : \ \text{if } x_j > 0 \text{ then } (x_j := x_j - 1;\ \text{goto } \mathbf{q_k}) \text{ else goto } q_h,$$

а условия вида $q_r \wedge \neg(x_t > 0)$ — правилам

$$(\delta_r)\ \ q_r : \ \text{if } x_t > 0 \text{ then } (x_t := x_t - 1;\ \text{goto } q_p) \text{ else goto } \mathbf{q_k}.$$

Условия вида $q_s \wedge x_l < \mathrm{bnd} x_l$ соответствуют правилам переходов первого типа

$(\delta_s)\ \ q_s :\ \ \mathsf{if}\ x_l < \mathrm{bnd} x_l\ \mathsf{then}\ x_l := x_l + 1;\ \mathsf{goto}\ \mathbf{q_k}.$

Важно отметить, что все переменные приведённой LTL-формулы, стоящие в условиях после оператора импликации, должны быть отличными от переменной-состояния q_k, т. е. переход в то же самое состояние здесь не специфицируется.

Если счётчиковая машина не имеет правил переходов, приводящих в состояние q_k, то для этого состояния LTL-формула «активации» строится как

$$\mathbf{G}(\ \neg q_k \wedge \mathbf{X}(q_k) \implies \mathrm{false}\ ).$$

LTL-формула «деактивации» (выхода из) состояния q_k, которому соответствует правило переходов первого типа со счётчиком x_j, имеет вид:

$$\mathbf{G}(\ q_k \wedge \neg\mathbf{X}(q_k) \implies x_j < \mathrm{bnd} x_j\ ).$$

Для правила переходов второго типа без петель получаем LTL-формулу

$$\mathbf{G}(\ q_k \wedge \neg\mathbf{X}(q_k) \implies \mathrm{true}\ ),$$

где логическая константа true означает, что переменная q_k должна быть обязательно сброшена в ноль уже на следующем шаге после её активации, т. е. после присваивания значения 1.

Несмотря на то что импликация внутри последней LTL-формулы является тавтологией, построение этой формулы имеет смысл, поскольку условие срабатывания в виде константы true участвует в LTL-формуле, описывающей ситуации, при которых переменная q_k сохраняет своё значение. По сути здесь налагается запрет иметь переменной q_k значение 1 дольше одного шага исполнения счётчиковой машины.

Если правило переходов для q_k имеет вид петли $(\delta_k)\ \ \mathbf{q_k} :\ \mathsf{if}\ x_j < \mathrm{bnd} x_j\ \mathsf{then}\ x_j := x_j + 1;\ \mathsf{goto}\ \mathbf{q_k}$, состояние q_k соответствует двум петлям $(\delta_k)\ \ \mathbf{q_k} :\ \mathsf{if}\ x_l > 0\ \mathsf{then}\ (x_j := x_j - 1;\ \mathsf{goto}\ \mathbf{q_k})\ \mathsf{else}\ \mathsf{goto}\ \mathbf{q_k}$ или же q_k является финальным состоянием, то переменная q_k будет иметь следующую LTL-формулу:

$$\mathbf{G}(\ q_k \wedge \neg\mathbf{X}(q_k) \implies \mathrm{false}\ ).$$

Если состояние второго типа q_k имеет в правиле переходов со счётчиком x_j только одну петлю, то для переменной q_k выбирается соответствующий вариант LTL-формулы

$$\mathbf{G}(\ q_k \wedge \neg\mathbf{X}(q_k) \implies x_j > 0\ )\quad \text{или}\quad \mathbf{G}(\ q_k \wedge \neg\mathbf{X}(q_k) \implies \neg(x_j > 0)\ ).$$

LTL-формула, описывающая ситуации, при которых переменная-состояние q_k сохраняет своё значение после применения некоторого правила переходов, строится автоматически по уже рассмотренным формулам «активации» и «деактивации»:

$$\mathbf{G}(\ (q_k \iff \mathbf{X}(q_k)) \implies \neg(\neg q_k \wedge \mathit{FiringCondA}) \wedge \neg(q_k \wedge \mathit{FiringCondD})\ ),$$

где $\mathit{FiringCondA}$ — условие, которое стоит после оператора импликации в LTL-формуле активации состояния q_k, а $\mathit{FiringCondD}$ — соответствующее условие из формулы деактивации состояния q_k.

И наконец, во всех рассмотренных LTL-формулах заменим условные выражения $x_j < \mathrm{bnd} x_j$ и $x_j > 0$ на соответствующие булевы функции $\mathrm{lt}(x_j^{k-1}, \ldots, x_j^0)$ и $\mathrm{gt}(x_j^{k-1}, \ldots, x_j^0)$, реализующие эти выражения:

$$x_j < \mathrm{bnd} x_j \ \equiv\ \mathrm{lt}(x_j^{k-1}, \ldots, x_j^0) \stackrel{\mathrm{def}}{=} \neg x_j^{k-1}\ op_{k-1}\ (\neg x_j^{k-2}\ op_{k-2}\ (\ldots(\neg x_j^1\ op_1\ (\neg x_j^0\ op_0\ 0))\ldots)),$$

$$x_j > 0 \ \equiv\ \mathrm{gt}(x_j^{k-1}, \ldots, x_j^0) \stackrel{\mathrm{def}}{=} x_j^{k-1} \vee x_j^{k-2} \vee \cdots \vee x_j^1 \vee x_j^0,$$

где $x_j = \sum_{i=0}^{k-1} x_j^i\, 2^i$, $\mathrm{bnd}x_j = \sum_{i=0}^{k-1} b_j^i\, 2^i$, $b_j^i \in \{0,1\}$, $k = \lceil \log_2(\mathrm{bnd}x_j + 1) \rceil$, $op_i \in \{\vee, \wedge\}$, $0 \leqslant i \leqslant k-1$. При этом $op_i = \vee$, если $b_j^i = 1$, и $op_i = \wedge$, если $b_j^i = 0$.

Отметим, что эти замены условных выражений могут быть выполнены через бинарные переменные $\mathrm{lt}x_j$ и $\mathrm{gt}x_j$ и LTL-формулы

$$\mathbf{G}\,(\ \mathrm{lt}x_j \iff \neg x_j^{k-1}\ op_{k-1}\ (\neg x_j^{k-2}\ op_{k-2}\ (\ldots(\neg x_j^1\ op_1\ (\neg x_j^0\ op_0\ 0))\ldots))\ ),$$

$$\mathbf{G}\,(\ \mathrm{gt}x_j \iff x_j^{k-1} \vee x_j^{k-2} \vee \cdots \vee x_j^1 \vee x_j^0\ ).$$

Таким образом, во всех формулах LTL-спецификации ограниченной счётчиковой машины будут использоваться только логические операторы и двоичные переменные.

6. LTL-спецификация счётчиковой машины возведения числа в квадрат

Построим LTL-спецификацию поведения ограниченной трехсчётчиковой машины $3cM$ возведения числа n в квадрат для всех значений n из диапазона 0 до 15 включительно. В этой LTL-спецификации переменная-счётчик a при инициализации получает значение n, $0 \leqslant n \leqslant 15$. Переменная-состояние q_0 инициализируется единицей, т. е. изначально имеем $q_0 = 1$. Остальные переменные при инициализации выставляются в ноль. В LTL-формулах в качестве предельных значений переменных-счётчиков будем использовать конкретные числа: $\mathrm{bnd}a = 15$, $\mathrm{bnd}b = 15$ и $\mathrm{bnd}c = 225$.

Целочисленные переменные-счётчики a, b и c, а также число n, которое подаётся на вход счётчиковой машины, будем представлять с помощью соответствующих наборов бинарных переменных. При этом формула Sn используется для фиксации значения n на протяжении одного исполнения счётчиковой машины $3cM$, стартующей из начального состояния q_0 при $a = n$, $b = 0$ и $c = 0$.

Sn : $\mathbf{G}\,(\ (\mathbf{X}(n_3) \iff n_3) \wedge (\mathbf{X}(n_2) \iff n_2) \wedge (\mathbf{X}(n_1) \iff n_1) \wedge (\mathbf{X}(n_0) \iff n_0)\ )$;

Sinit : $q_0 \wedge \neg q_1 \wedge \neg q_2 \wedge \neg q_3 \wedge \neg q_4 \wedge \neg q_5 \wedge \neg q_6 \wedge \neg q_7 \wedge (a_3 \iff n_3) \wedge (a_2 \iff n_2) \wedge (a_1 \iff n_1) \wedge (a_0 \iff n_0) \wedge \neg b_3 \wedge \neg b_2 \wedge \neg b_1 \wedge \neg b_0 \wedge \neg c_7 \wedge \neg c_6 \wedge \neg c_5 \wedge \neg c_4 \wedge \neg c_3 \wedge \neg c_2 \wedge \neg c_1 \wedge \neg c_0$;

Sgl : $\mathbf{G}\,(\ \mathrm{gt}a \iff a_3 \vee a_2 \vee a_1 \vee a_0\ ) \wedge$
$\mathbf{G}\,(\ \mathrm{gt}b \iff b_3 \vee b_2 \vee b_1 \vee b_0\ ) \wedge$
$\mathbf{G}\,(\ \mathrm{lt}a \iff \neg a_3 \vee \neg a_2 \vee \neg a_1 \vee \neg a_0\ ) \wedge$
$\mathbf{G}\,(\ \mathrm{lt}b \iff \neg b_3 \vee \neg b_2 \vee \neg b_1 \vee \neg b_0\ ) \wedge$
$\mathbf{G}\,(\ \mathrm{lt}c \iff \neg c_7 \vee \neg c_6 \vee \neg c_5 \vee \neg c_4 \wedge \neg c_3 \wedge \neg c_2 \wedge \neg c_1 \wedge \neg c_0\ )$;

Sa : $\mathbf{G}\,(\ \mathbf{X}(a_3) \iff \neg(a_3 \iff (q_6 \wedge \mathrm{lt}a \wedge a_2 \wedge a_1 \wedge a_0 \vee (q_0 \vee q_2) \wedge \mathrm{gt}a \wedge \neg a_2 \wedge \neg a_1 \wedge \neg a_0))\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(a_2) \iff \neg(a_2 \iff (q_6 \wedge \mathrm{lt}a \wedge a_1 \wedge a_0 \quad \vee (q_0 \vee q_2) \wedge \mathrm{gt}a \wedge \neg a_1 \wedge \neg a_0))\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(a_1) \iff \neg(a_1 \iff (q_6 \wedge \mathrm{lt}a \wedge a_0 \quad \vee (q_0 \vee q_2) \wedge \mathrm{gt}a \wedge \neg a_0))\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(a_0) \iff \neg(a_0 \iff (q_6 \wedge \mathrm{lt}a \quad \vee (q_0 \vee q_2) \wedge \mathrm{gt}a))\ )$;

Sb : $\mathbf{G}\,(\ \mathbf{X}(b_3) \iff \neg(b_3 \iff (q_3 \wedge \mathrm{lt}b \wedge b_2 \wedge b_1 \wedge b_0 \vee q_5 \wedge \mathrm{gt}b \wedge \neg b_2 \wedge \neg b_1 \wedge \neg b_0))\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(b_2) \iff \neg(b_2 \iff (q_3 \wedge \mathrm{lt}b \wedge b_1 \wedge b_0 \quad \vee q_5 \wedge \mathrm{gt}b \wedge \neg b_1 \wedge \neg b_0))\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(b_1) \iff \neg(b_1 \iff (q_3 \wedge \mathrm{lt}b \wedge b_0 \quad \vee q_5 \wedge \mathrm{gt}b \wedge \neg b_0))\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(b_0) \iff \neg(b_0 \iff (q_3 \wedge \mathrm{lt}b \quad \vee q_5 \wedge \mathrm{gt}b))\ )$;

Sc : $\mathbf{G}\,(\ \mathbf{X}(c_7) \iff \neg(c_7 \iff (q_1 \vee q_4) \wedge \mathrm{lt}c \wedge c_6 \wedge c_5 \wedge c_4 \wedge c_3 \wedge c_2 \wedge c_1 \wedge c_0)\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(c_6) \iff \neg(c_6 \iff (q_1 \vee q_4) \wedge \mathrm{lt}c \wedge c_5 \wedge c_4 \wedge c_3 \wedge c_2 \wedge c_1 \wedge c_0)\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(c_5) \iff \neg(c_5 \iff (q_1 \vee q_4) \wedge \mathrm{lt}c \wedge c_4 \wedge c_3 \wedge c_2 \wedge c_1 \wedge c_0)\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(c_4) \iff \neg(c_4 \iff (q_1 \vee q_4) \wedge \mathrm{lt}c \wedge c_3 \wedge c_2 \wedge c_1 \wedge c_0)\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(c_3) \iff \neg(c_3 \iff (q_1 \vee q_4) \wedge \mathrm{lt}c \wedge c_2 \wedge c_1 \wedge c_0)\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(c_2) \iff \neg(c_2 \iff (q_1 \vee q_4) \wedge \mathrm{lt}c \wedge c_1 \wedge c_0)\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(c_1) \iff \neg(c_1 \iff (q_1 \vee q_4) \wedge \mathrm{lt}c \wedge c_0)\ ) \wedge$
$\mathbf{G}\,(\ \mathbf{X}(c_0) \iff \neg(c_0 \iff (q_1 \vee q_4) \wedge \mathrm{lt}c)\ )$;

$$
\begin{aligned}
\text{Sq0}: \quad & \mathbf{G}(\ \neg q_0 \wedge \ \mathbf{X}(q_0) \implies q_5 \wedge \neg \mathrm{gt}b\ ) \wedge \\
& \mathbf{G}(\ \ q_0 \wedge \neg\mathbf{X}(q_0) \implies \mathrm{true}\ ) \wedge \\
& \mathbf{G}(\ (q_0 \iff \mathbf{X}(q_0)) \implies \neg(\neg q_0 \wedge q_5 \wedge \neg \mathrm{gt}b) \wedge \neg q_0\ ); \\
\text{Sq1}: \quad & \mathbf{G}(\ \neg q_1 \wedge \ \mathbf{X}(q_1) \implies q_0 \wedge \mathrm{gt}a \vee q_4 \wedge \mathrm{lt}c\ ) \wedge \\
& \mathbf{G}(\ \ q_1 \wedge \neg\mathbf{X}(q_1) \implies \mathrm{lt}c\ ) \wedge \\
& \mathbf{G}(\ (q_1 \iff \mathbf{X}(q_1)) \implies \neg(\neg q_1 \wedge (q_0 \wedge \mathrm{gt}a \vee q_4 \wedge \mathrm{lt}c)) \wedge \neg(q_1 \wedge \mathrm{lt}c)\ ); \\
\text{Sq2}: \quad & \mathbf{G}(\ \neg q_2 \wedge \ \mathbf{X}(q_2) \implies q_1 \wedge \mathrm{lt}c\ ) \wedge \\
& \mathbf{G}(\ \ q_2 \wedge \neg\mathbf{X}(q_2) \implies \mathrm{true}\ ) \wedge \\
& \mathbf{G}(\ (q_2 \iff \mathbf{X}(q_2)) \implies \neg(\neg q_2 \wedge q_1 \wedge \mathrm{lt}c) \wedge \neg q_2\ ); \\
\text{Sq3}: \quad & \mathbf{G}(\ \neg q_3 \wedge \ \mathbf{X}(q_3) \implies q_2 \wedge \mathrm{gt}a\ ) \wedge \\
& \mathbf{G}(\ \ q_3 \wedge \neg\mathbf{X}(q_3) \implies \mathrm{lt}b\ ) \wedge \\
& \mathbf{G}(\ (q_3 \iff \mathbf{X}(q_3)) \implies \neg(\neg q_3 \wedge q_2 \wedge \mathrm{gt}a) \wedge \neg(q_3 \wedge \mathrm{lt}b)\ ); \\
\text{Sq4}: \quad & \mathbf{G}(\ \neg q_4 \wedge \ \mathbf{X}(q_4) \implies q_3 \wedge \mathrm{lt}b\ ) \wedge \\
& \mathbf{G}(\ \ q_4 \wedge \neg\mathbf{X}(q_4) \implies \mathrm{lt}c\ ) \wedge \\
& \mathbf{G}(\ (q_4 \iff \mathbf{X}(q_4)) \implies \neg(\neg q_4 \wedge q_3 \wedge \mathrm{lt}b) \wedge \neg(q_4 \wedge \mathrm{lt}c)\ ); \\
\text{Sq5}: \quad & \mathbf{G}(\ \neg q_5 \wedge \ \mathbf{X}(q_5) \implies q_2 \wedge \neg\mathrm{gt}a \vee q_6 \wedge \mathrm{lt}a\ ) \wedge \\
& \mathbf{G}(\ \ q_5 \wedge \neg\mathbf{X}(q_5) \implies \mathrm{true}\ ) \wedge \\
& \mathbf{G}(\ (q_5 \iff \mathbf{X}(q_5)) \implies \neg(\neg q_5 \wedge (q_2 \wedge \neg\mathrm{gt}a \vee q_6 \wedge \mathrm{lt}a)) \wedge \neg q_5\ ); \\
\text{Sq6}: \quad & \mathbf{G}(\ \neg q_6 \wedge \ \mathbf{X}(q_6) \implies q_5 \wedge \mathrm{gt}b\ ) \wedge \\
& \mathbf{G}(\ \ q_6 \wedge \neg\mathbf{X}(q_6) \implies \mathrm{lt}a\ ) \wedge \\
& \mathbf{G}(\ (q_6 \iff \mathbf{X}(q_6)) \implies \neg(\neg q_6 \wedge q_5 \wedge \mathrm{gt}b) \wedge \neg(q_6 \wedge \mathrm{lt}a)\ ); \\
\text{Sq7}: \quad & \mathbf{G}(\ \neg q_7 \wedge \ \mathbf{X}(q_7) \implies q_0 \wedge \neg\mathrm{gt}a\ ) \wedge \\
& \mathbf{G}(\ \ q_7 \wedge \neg\mathbf{X}(q_7) \implies \mathrm{false}\ ) \wedge \\
& \mathbf{G}(\ (q_7 \iff \mathbf{X}(q_7)) \implies \neg(\neg q_7 \wedge q_0 \wedge \neg\mathrm{gt}a)\ ).
\end{aligned}
$$

Рассмотрим LTL-свойства счётчиковой машины $3cM$, которые обязательно должны выполняться для любого её исполнения. При записи формул будем использовать исходные названия счётчиков a, b и c, а также вспомогательную переменную n, предполагая, что $a = \sum_{i=0}^{3} a_i 2^i$, $b = \sum_{i=0}^{3} b_i 2^i$, $c = \sum_{i=0}^{7} c_i 2^i$ и $n = \sum_{i=0}^{3} n_i 2^i$.

P1 : $\mathbf{G}(\ q_7 \implies c = n^2 \wedge a = 0 \wedge b = 0\ )$.
Это свойство означает, что если счётчиковая машина $3cM$ оказывается в финальном состоянии q_7, то значения счётчиков a и b будут равны 0, а счётчик c будет содержать искомый результат n^2.

P2 : $\mathbf{G}(\ a + b \leqslant n\ )$.
Это свойство требует, чтобы суммарное значение счётчиков a и b было ограничено константой n на протяжении всего исполнения счётчиковой машины $3cM$.

P3 : $\mathbf{G}(\ c \leqslant n^2\ )$.
Требуется, чтобы значение счётчика c на протяжении всего исполнения машины $3cM$ не выходило за пределы n^2.

P4 : $\mathbf{G}\mathbf{F}(q_7)$.
Счётчиковая машина $3cM$ из любого состояния (в том числе и из начального q_0) всегда рано или поздно перейдёт в финальное состояние q_7.

P5 : $\mathbf{G}(\ q_7 \implies \mathbf{X}(q_7)\ )$.
Если счётчиковая машина $3cM$ попадает в финальное состояние q_7, то она навсегда остаётся в этом состоянии.

P6 : $\mathbf{G}(\ q_0 + q_1 + q_2 + q_3 + q_4 + q_5 + q_6 + q_7 = 1\ )$.
Всегда активно только одно состояние счётчиковой машины $3cM$.

Тогда справедливость логического вывода (в рамках логики LTL)

$$\mathrm{Sn}, \mathrm{Sinit}, \mathrm{Sa}, \mathrm{Sb}, \mathrm{Sc}, \mathrm{Sq0}, \mathrm{Sq1}, \mathrm{Sq2}, \mathrm{Sq3}, \mathrm{Sq4}, \mathrm{Sq5}, \mathrm{Sq6}, \mathrm{Sq7} \models \mathrm{P1}, \mathrm{P2}, \mathrm{P3}, \mathrm{P4}, \mathrm{P5}, \mathrm{P6}$$

будет означать, что счётчиковая машина $3cM$ для любого n, $0 \leqslant n \leqslant 15$, стартуя из начального состояния q_0 при $a = n$, $b = 0$ и $c = 0$, обязательно завершит работу в финальном состоянии q_7 с результатом $a = 0$, $b = 0$ и $c = n^2$. При этом на протяжении всего исполнения будет выполняться $a + b \leqslant n$ и $c \leqslant n^2$. Действительно, левая часть логического вывода описывает те и только те интерпретации, которые соответствуют всем возможным исполнениям счётчиковой машины $3cM$ при условии $0 \leqslant n \leqslant 15$, а правая часть вывода обязывает, чтобы эти интерпретации/исполнения удовлетворяли требуемым свойствам машины $3cM$.

Рассмотрим пример логического вывода, который не будет являться справедливым в логике LTL применительно к счётчиковой машине $3cM$. Потребуем, чтобы в каждом возможном исполнении машины $3cM$ итоговый результат её работы в состоянии q_7 был отличным от $c = 2n$ при $n > 0$:

$$\mathrm{P7}: \quad \mathbf{G}\,(\, q_7 \wedge n > 0 \implies \neg(c = 2n)\,).$$

В этом случае получим, что формула P7 не будет выводиться из приведённых выше предпосылок

$$\mathrm{Sn}, \mathrm{Sinit}, \mathrm{Sa}, \mathrm{Sb}, \mathrm{Sc}, \mathrm{Sq0}, \mathrm{Sq1}, \mathrm{Sq2}, \mathrm{Sq3}, \mathrm{Sq4}, \mathrm{Sq5}, \mathrm{Sq6}, \mathrm{Sq7} \not\models \mathrm{P7},$$

так как существует контрпример — интерпретация, при которой все формулы левой части логического вывода являются истинными, а правая часть в виде формулы-заключения P7 становится ложной. Эта интерпретация представляет собой цепочку из 16 различных миров, последний из которых имеет петлю, т. е. достижим сам из себя. Контрпример соответствует исполнению счётчиковой машины $3cM$ при $n = 2$.

Проверку справедливости рассмотренных логических выводов можно выполнить, например, с помощью программного средства верификации Cadence SMV [5, 7, 11].

7. Проверка справедливости логических выводов

Ниже приводится код на языке программного средства верификации Cadence SMV [5], позволяющий проверить выполнимость свойств счётчиковой машины $3cM$ с использованием линейной темпоральной логики LTL.

На вход верификатора Cadence SMV подаются описания переменных-состояний, переменных-счётчиков и «константы» n, а также описания вспомогательных переменных, применяющихся для реализации условных функций. Указываются области допустимых значений этих переменных. Запись формул логики LTL происходит через ключевое слово `assert`.

Проверка выполнимости каждого LTL-свойства `P1`, `P2`, `P3`, `P4`, `P5`, `P6` и `P7` проводится по отдельности посредством вызова соответствующей команды главного меню программы Cadence SMV.

Поскольку имена свойств задействованы в конструкции `using ... prove`, а точнее, указаны в разделе `prove`, то проверка выполнимости выбранного свойства будет проводиться только для тех интерпретаций логики LTL, которые одновременно удовлетворяют всем LTL-формулам, перечисленным в разделе `using`.

Контрпример для свойства `P7` выводится в виде таблицы, каждый столбец которой содержит вектор значений всех переменных в соответствующем мире найденной интерпретации.

На языке программного средства Cadence SMV символы «&», «|», «~», «->» и «<->» означают логические операторы «$\wedge$», «$\vee$», «$\neg$», «$\Longrightarrow$» и «$\Longleftrightarrow$» соответственно. А константы «true» и «false» имеют вид «1» и «0».

```
module main()
{ /* ------- описание переменных ------- */
  q0, q1, q2, q3, q4, q5, q6, q7: 0..1;
  a, b, n: 0..15;
  c: 0..255;
  c0, c1, c2, c3, c4, c5, c6, c7: 0..1;
  a0, a1, a2, a3: 0..1;
  b0, b1, b2, b3: 0..1;
  n0, n1, n2, n3: 0..1;
  gta, lta, gtb, ltb, ltc: 0..1;
  /* ------- представления счётчиков a, b и c ------- */
  a :=  a0 + a1*2 + a2*4 + a3*8;
  b :=  b0 + b1*2 + b2*4 + b3*8;
  n :=  n0 + n1*2 + n2*4 + n3*8;
  c :=  c0 + c1*2 + c2*4 + c3*8 + c4*16 + c5*32 + c6*64 + c7*128;
  /* ------- LTL-спецификация ------- */
  Sn: assert /* формулы для константы n */
      G( (X(n3) <-> n3) & (X(n2) <-> n2) & (X(n1) <-> n1) & (X(n0) <-> n0) );
  Sinit: assert /* формула инициализации */
          q0 & ~q1 & ~q2 & ~q3 & ~q4 & ~q5 & ~q6 & ~q7 &
         (a3 <-> n3) & (a2 <-> n2) & (a1 <-> n1) & (a0 <-> n0) &
         ~b3 & ~b2 & ~b1 & ~b0 & ~c7 & ~c6 & ~c5 & ~c4 & ~c3 & ~c2 & ~c1 & ~c0;
  Sgl: assert /* формулы для функций gt и lt */
       G( gta <->  a3 |  a2 |  a1 |  a0 ) &
       G( gtb <->  b3 |  b2 |  b1 |  b0 ) &
       G( lta <-> ~a3 | ~a2 | ~a1 | ~a0 ) &
       G( ltb <-> ~b3 | ~b2 | ~b1 | ~b0 ) &
       G( ltc <-> ~c7 | ~c6 | ~c5 | ~c4 & ~c3 & ~c2 & ~c1 & ~c0 );
  Sa: assert /* формулы для переменных a3, a2, a1, a0 представления счётчика a */
      G( X(a3) <->  ~(a3 <-> (q6 & lta & a2 & a1 & a0 | (q0 | q2) & gta & ~a2 & ~a1 & ~a0)) ) &
      G( X(a2) <->  ~(a2 <-> (q6 & lta & a1 & a0      | (q0 | q2) & gta & ~a1 & ~a0      )) ) &
      G( X(a1) <->  ~(a1 <-> (q6 & lta & a0           | (q0 | q2) & gta & ~a0           )) ) &
      G( X(a0) <->  ~(a0 <-> (q6 & lta                | (q0 | q2) & gta                )) );
  Sb: assert /* формулы для переменных b3, b2, b1, b0 представления счётчика b */
      G( X(b3) <->  ~(b3 <-> (q3 & ltb & b2 & b1 & b0 | q5 & gtb & ~b2 & ~b1 & ~b0)) ) &
      G( X(b2) <->  ~(b2 <-> (q3 & ltb & b1 & b0      | q5 & gtb & ~b1 & ~b0      )) ) &
      G( X(b1) <->  ~(b1 <-> (q3 & ltb & b0           | q5 & gtb & ~b0           )) ) &
      G( X(b0) <->  ~(b0 <-> (q3 & ltb                | q5 & gtb                )) );
  Sc: assert /* формулы для переменных c7, c6, c5, c4, c3, c2, c1, c0 представления счётчика c */
      G( X(c7) <->  ~(c7 <-> (q1 | q4) & ltc & c6 & c5 & c4 & c3 & c2 & c1 & c0) ) &
      G( X(c6) <->  ~(c6 <-> (q1 | q4) & ltc & c5 & c4 & c3 & c2 & c1 & c0     ) ) &
      G( X(c5) <->  ~(c5 <-> (q1 | q4) & ltc & c4 & c3 & c2 & c1 & c0          ) ) &
      G( X(c4) <->  ~(c4 <-> (q1 | q4) & ltc & c3 & c2 & c1 & c0               ) ) &
      G( X(c3) <->  ~(c3 <-> (q1 | q4) & ltc & c2 & c1 & c0                    ) ) &
      G( X(c2) <->  ~(c2 <-> (q1 | q4) & ltc & c1 & c0                         ) ) &
      G( X(c1) <->  ~(c1 <-> (q1 | q4) & ltc & c0                              ) ) &
      G( X(c0) <->  ~(c0 <-> (q1 | q4) & ltc                                   ) );
  Sq0: assert /* формулы для переменной q0 */
       G( ~q0  &  X(q0)  ->  q5 & ~gtb ) &
       G(  q0  & ~X(q0)  ->  1 ) &
       G( (q0 <-> X(q0)) ->  ~(~q0 & q5 & ~gtb) & ~q0 );
  Sq1: assert /* формулы для переменной q1 */
       G( ~q1  &  X(q1)  ->  q0 & gta | q4 & ltc ) &
       G(  q1  & ~X(q1)  ->  ltc ) &
       G( (q1 <-> X(q1)) ->  ~(~q1 & (q0 & gta | q4 & ltc)) & ~(q1 & ltc) );
```

```
  Sq2: assert /* формулы для переменной q2 */
       G( ~q2  &  X(q2)  ->  q1 & ltc ) &
       G(  q2  & ~X(q2)  ->  1 ) &
       G( (q2 <-> X(q2)) ->  ~(~q2 & q1 & ltc) & ~q2 );
  Sq3: assert /* формулы для переменной q3 */
       G( ~q3  &  X(q3)  ->  q2 & gta ) &
       G(  q3  & ~X(q3)  ->  ltb ) &
       G( (q3 <-> X(q3)) ->  ~(~q3 & q2 & gta) & ~(q3 & ltb) );
  Sq4: assert /* формулы для переменной q4 */
       G( ~q4  &  X(q4)  ->  q3 & ltb ) &
       G(  q4  & ~X(q4)  ->  ltc ) &
       G( (q4 <-> X(q4)) ->  ~(~q4 & q3 & ltb) & ~(q4 & ltc) );
  Sq5: assert /* формулы для переменной q5 */
       G( ~q5  &  X(q5)  ->  q2 & ~gta | q6 & lta ) &
       G(  q5  & ~X(q5)  ->  1 ) &
       G( (q5 <-> X(q5)) ->  ~(~q5 & (q2 & ~gta | q6 & lta)) & ~q5 );
  Sq6: assert /* формулы для переменной q6 */
       G( ~q6  &  X(q6)  ->  q5 & gtb ) &
       G(  q6  & ~X(q6)  ->  lta ) &
       G( (q6 <-> X(q6)) ->  ~(~q6 & q5 & gtb) & ~(q6 & lta) );
  Sq7: assert /* формулы для переменной q7 */
       G( ~q7  &  X(q7)  ->  q0 & ~gta ) &
       G(  q7  & ~X(q7)  ->  0 ) &
       G( (q7 <-> X(q7)) ->  ~(~q7 & q0 & ~gta) );
  /* ------- проверяемые свойства счётчиковой машины ------- */
  P1: assert G( q7 -> c=n*n & a=0 & b=0 );
  P2: assert G( a+b <= n );
  P3: assert G( c <= n*n );
  P4: assert G F(q7);
  P5: assert G( q7 -> X(q7) );
  P6: assert G( q0+q1+q2+q3+q4+q5+q6+q7 = 1 );
  P7: assert G( q7 & n>0 -> ~(c = 2*n) );
  /* ------- предпосылки и заключения в логических выводах ------- */
  using  /* используя предпосылки */
    Sn, Sinit, Sgl, Sa, Sb, Sc, Sq0, Sq1, Sq2, Sq3, Sq4, Sq5, Sq6, Sq7
  prove  /* доказать заключения */
    P1, P2, P3, P4, P5, P6, P7;
}
```

Заключение

В статье предложен способ описания поведения ограниченной счётчиковой машины с помощью набора формул линейной темпоральной логики LTL. В LTL-спецификации используются только двоичные переменные в виде элементарных высказываний. При этом обращение к предыдущим значениям переменных осуществляется исключительно в терминах самой логики LTL посредством соответствующих формул. Введённые ограничения на допустимые значения счётчиков играют роль ограничений на размер памяти программной системы и позволяют заранее определить количество двоичных переменных, необходимых для представления счётчиков машины Минского.

Ранее в работе [1] для хранения предыдущих значений счётчиков и состояний машины Минского отводились специальные дополнительные переменные. Однако при доказательстве справедливости логических LTL-выводов в программе Cadence SMV каждая переменная преобразуется в набор двоичных переменных. Таким образом, представление счётчиков и состояний машины сразу в виде двоичных переменных, не требующее выделения дублирующих переменных для хранения предыдущих значений, в два раза уменьшает общее число переменных, задействованных в описании поведения машины, что значительно сокращает время на проверку справедливости логических выводов.

References

[1] E. V. Kuzmin, "LTL-Specification of Counter Machines", in Russian, *Modeling and Analysis of Information Systems*, vol. 28, no. 1, pp. 104–119, 2021.

[2] M. Minsky, *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967.

[3] R. Schroeppel, "A Two Counter Machine Cannot Calculate 2^N", Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Artificial Intelligence Memo #257, 1972, p. 32.

[4] E. V. Kuzmin, *Counter Machines*, in Russian. Yaroslavl: Yaroslavl State University, 2010, p. 128.

[5] *Cadence SMV*. [Online]. Available: http://www.kenmcmil.com/smv.html.

[6] A. Pnueli, "The Temporal Logic of Programs", in *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, IEEE Computer Society Press, 1977, pp. 46–57.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. The MIT Press, 2001.

[8] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.

[9] G. Priest, *An Introduction to Non-Classical Logic. From if to is.* Cambridge University Press, 2008, p. 648.

[10] E. V. Kuzmin, *Non-Classical Propositional Logics*, in Russian. Yaroslavl: P.G. Demidov Yaroslavl State University, 2016, p. 160.

[11] E. Clarke, O. Grumberg, and K. Hamaguchi, "Another Look at LTL Model Checking", Carnegie Mellon University, Tech. Rep. CMU-CS-94-114, 1994.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 29, NO. 1, 2022
journal homepage: www.mais-journal.ru

THEORY OF COMPUTING

Methods for Change Parallelism in Process of High-level VLSI Synthesis

I. N. Ryzhenko[1], O. V. Nepomnyaschy[1], A. I. Legalov[2], V. V. Shaidurov[3]

DOI: 10.18255/1818-1015-2022-1-60-72

[1]Siberian Federal University, 79 Svobodny pr., Krasnoyarsk 660041, Russia.
[2]Higher School of Economics, 20 Myasnitskaya str., Moscow 101000, Russia.
[3]Krasnoyarsk Science Centre of the Siberian Branch of Russian Academy of Science, 50 Akademgorodok, Krasnoyarsk 660036, Russia.

MSC2020: 94-04
Research article
Full text in Russian

Received September 2, 2021
After revision February 23, 2022
Accepted March 9, 2022

In this paper methods for increasing the efficiency of VLSI development based on the method of architecture-independent design are proposed. The route of high-level VLSI synthesis is considered. The principle of constructing a VLSI hardware model based on the functional-flow programming paradigm is stated.
The results of the development of methods and algorithms for transformation functional-parallel programs into programs in HDL languages that support the design process of digital chips are presented. The principles of assessment are considered and the classes of resources required for the analysis of design solutions are identified. Reduction coefficients and methods of their calculation for each resource class have been introduced. An algorithm for calculating the reduction coefficients and estimating the required resources is proposed. An algorithm for converting parallelism is proposed, taking into account the specified constraints of the target platform. A mechanism for the exchange of metrics with an architecture-dependent level has been developed. Examples of parallelism reduction for the FPGA platform and practical implementation of FFT algorithms in the Virtex® UltraScale FPGA basis are given. The developed methods and algorithms make it possible to use the method of architecture-independent synthesis for transferring VLSI projects to various architectures by changing the parallelism of the circuit and equivalent transformations of parallel programs. The proposed approach provides many options for hardware solutions for implementation on various target platforms.

Keywords: parallel computing; dataflow; functional programming; high-level synthesis; VLSI

INFORMATION ABOUT THE AUTHORS

Igor Nikolaevich Ryzhenko correspondence author	orcid.org/0000-0002-3069-9102. E-mail: rodgi.krs@gmail.com Senior lecturer, PhD student.
Oleg Vladimirovich Nepomnyaschy	orcid.org/0000-0002-2459-6414. E-mail: 2955005@gmail.com Head of the Department of Computer Science, PhD, Professor.
Aleksandr Ivanovich Legalov	orcid.org/0000-0002-5487-0699. E-mail: alegalov@hse.ru Dr. (Doctor) of Technical Science, Professor.
Vladimir Viktorovich Shaidurov	orcid.org/0000-0002-7883-5804. E-mail: shaidurov04@mail.ru Dr. (Doctor) of Physics and Mathematics Sciences, Corresponded Member of RAS.

For citation: I. N. Ryzhenko, O. V. Nepomnyaschy, A. I. Legalov, and V. V. Shaidurov, “Methods for Change Parallelism in Process of High-level VLSI Synthesis”, *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 60-72, 2022.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 29, № 1, 2022
сайт журнала: www.mais-journal.ru

THEORY OF COMPUTING

Методы преобразования параллелизма в процессе высокоуровневого синтеза СБИС

И. Н. Рыженко[1], О. В. Непомнящий[1], А. И. Легалов[2], В. В. Шайдуров[3]

DOI: 10.18255/1818-1015-2022-1-60-72

[1]Сибирский Федеральный Университет, пр. Свободный, д. 79, г. Красноярск, 660041 Россия.
[2]Национальный исследовательский университет "Высшая школа экономики", ул. Мясницкая, д. 20, г. Москва, 101000 Россия.
[3]ФГБНУ Федеральный исследовательский центр "Красноярский научный центр Сибирского отделения Российской академии наук", Академгородок, д. 50, г. Красноярск, 660036 Россия.

УДК 004.4'416+004.432.42
Научная статья
Полный текст на русском языке

Получена 2 сентября 2021 г.
После доработки 23 февраля 2022 г.
Принята к публикации 9 марта 2022 г.

Предложены методы повышения эффективности разработки СБИС на основе метода архитектурно-независимого проектирования. Рассмотрен маршрут высокоуровневого синтеза СБИС. Изложен принцип построения аппаратной модели СБИС на основе функционально-потоковой парадигмы программирования.
Представлены результаты разработки методов и алгоритмов трансформации, функционально-потоковых параллельных программ в программы на языках описания аппаратуры, обеспечивающих поддержку процесса проектирования цифровых однокристальных систем. Рассмотрены принципы оценки и выделены классы ресурсов, требуемых для анализа проектных решений. Введены коэффициенты редукции и методики их расчета по каждому классу ресурсов. Предложен алгоритм расчета коэффициентов редукции и оценки требуемых ресурсов. Предложен алгоритм преобразования параллелизма с учетом заданных ограничений целевой платформы. Разработан механизм обмена метриками с архитектурно-зависимым уровнем. Приведены примеры редукции параллелизма для платформы ПЛИС и практическая реализация тестовых алгоритмов БПФ в базисе ПЛИС Virtex® UltraScale. Разработанные методы и алгоритмы позволяют использовать метод архитектурно-независимого синтеза для переноса проектов СБИС на различные архитектуры с помощью изменения параллелизма схемы и эквивалентных преобразований параллельных программ. Предложенный подход обеспечивает множество вариантов аппаратных решений для реализации на различных целевых платформах.

Ключевые слова: параллельные вычисления; потоки данных; функционально-потоковое параллельное программирование; цифровая интегральная схема; высокоуровневый синтез

ИНФОРМАЦИЯ ОБ АВТОРАХ

Игорь Николаевич Рыженко автор для корреспонденции	orcid.org/0000-0002-3069-9102. E-mail: rodgi.krs@gmail.com ст. преподаватель, аспирант.
Олег Владимирович Непомнящий	orcid.org/0000-0002-2459-6414. E-mail: 2955005@gmail.com зав. кафедрой вычислительной техники, к.т.н., профессор.
Александр Иванович Легалов	orcid.org/0000-0002-5487-0699. E-mail: alegalov@hse.ru доктор технических наук, профессор.
Владимир Викторович Шайдуров	orcid.org/0000-0002-7883-5804. E-mail: shaidurov04@mail.ru доктор физ.-мат. наук, член-корреспондент РАН.

Для цитирования: I. N. Ryzhenko, O. V. Nepomnyaschy, A. I. Legalov, and V. V. Shaidurov, "Methods for Change Parallelism in Process of High-level VLSI Synthesis", *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 60-72, 2022.

Введение

При проектировании сверхбольших интегральных схем (СБИС) используют два основных подхода: классический, подразумевающий низкоуровневое представление исходных алгоритмов на языках описания аппаратуры (HDL – hardware description language), и высокоуровневый, при котором проект изначально абстрагируется от конечной реализации и описывается на системном уровне (ESL – Entire System Level).

В настоящее время классический подход не позволяет обеспечить требуемые сроки реализации и технические характеристики для сложно-функциональных проектов, реализуемых в базисе СБИС. Поэтому эффективные решения находят при использовании высокоуровневых подходов, что подтверждается мировыми тенденциями перехода на более высокие уровни абстракции в процессе разработки [1—3].

Так же известны попытки использования традиционного маршрута проектирования совместно с высокоуровневыми программными средствами, которые ранее были ориентированы на решение совершенно других задач, например, MatLab или LabView [1]. Но и данном случае, при высокой сложности проекта, конечная реализация имеет необоснованно высокие ресурсные требования и зачастую не соответствует требуемым техническим характеристикам.

При этом известные высокоуровневые подходы используют модели вычислений, зачастую плохо подходящие для представления СБИС. Например, компания Xilinx предлагает технологию HLS-синтеза [4], где применяется описание исходного алгоритма на императивном Си-подобном языке, что не обеспечивает переносимость программ на платформы сторонних производителей. Кроме того, переход от исходного Си-описания к низкоуровневому для последующей реализации подразумевает ручное или полуавтоматическое выделение параллелизма, что требует значительных временных затрат и высокой квалификации разработчика.

Поскольку СБИС является системой параллельной обработки данных, архитектурная независимость может быть достигнута при использовании подходов, обеспечивающих переносимость параллельных программ, в том числе использование моделей вычислений непосредственно задающих программу в виде графа потока данных [5]. Такой подход рассматривается в ряде работ по программированию универсальных параллельных вычислительных систем [6, 7]. Известны работы по использованию концепции ресурсно-независимого параллелизма и для реконфигурируемых вычислительных систем [8], где предлагается использовать язык программирования высокого уровня COLAMO. Близкий подход реализован в языке Set@l, программа на котором представляет собой описание алгоритма в виде архитектурно-независимого информационного графа и набора архитектурно-зависимых аспектов, что позволяет переносить ее между различными параллельными архитектурами без изменения алгоритма [9].

Таким образом, для обеспечения архитектурной независимости, требований по ресурсным ограничениям и основным техническим характеристикам, а также для сокращения сроков проектирования следует использовать максимальное абстрагирование исходных алгоритмов от целевого кристалла и создать механизм перехода на нижний, RTL-уровень с поддержкой сквозной верификации и параллелизма на уровне операций. Авторами предложен метод представления алгоритмов на верхнем уровне иерархии системного проектирования, который при последующем нисходящем проектировании обеспечивает сохранение параллелизма [10]. При этом предложенный подход дополняет известные маршруты проектирования, обеспечивая переносимость и архитектурную независимость исходных алгоритмов.

1. Метод и маршрут высокоуровневого синтеза

Цифровые СБИС функционируют в синхронном потоковом режиме, а это означает, что в данном случае реализуется граф потока данных, где узлами являются составляющие СБИС функциональ-

ные блоки, а ребрами – сигналы данных и управления. При этом блоки функционируют параллельно, а зависимость между ними определяется только сигналами управления, которые определяют готовность данных на его входах. Поэтому основная идея предложенного метода архитектурно-независимого синтеза СБИС заключается в переходе от высокоуровневого представления исходных алгоритмов на языке функционально-потокового параллельного программирования к низкоуровневому RTL путем использования модели вычислений с неограниченными ресурсами [11]. Такой подход использует одинаковую модель вычислений на всех уровнях разработки, а язык и метод разработки верхнего уровня ориентирован на описание параллельных программ, управляемых по готовности данных.

В работе [11] описаны используемые функционально-потоковая модель параллельных вычислений и язык программирования «Пифагор». В рамках дальнейших исследований разработаны инструментальные средства, поддерживающие отладку, верификацию и выполнение функционально-потоковых параллельных (ФПП) программ [12]. Архитектурная независимость системы обеспечивается следующим образом: считается, что виртуальная машина, предназначенная для выполнения ФПП программ, имеет неограниченные вычислительные ресурсы. Это позволяет выделять для каждой операции новый вычислительный ресурс. Процесс перехода от архитектурно-независимого описания параллелизма к конкретной вычислительной системе, в данном случае к цифровой схеме на кристалле, представляет собой редукцию параллелизма проекта СБИС под имеющиеся вычислительные ресурсы.

Авторами предложен маршрут проектирования, согласно которому в ходе трансляции с ФП языка строятся следующие промежуточные представления и структуры данных: информационный граф (ИГ), описывающий функциональные преобразования данных; управляющий граф (УГ), определяющий передачу управления между выполняемыми функциями. При этом передачи управляющих сигналов могут происходить параллельно, что является особенностью разрабатываемой программы. Кроме этого вводится дополнительный слой, так называемый HDL-граф, который используется на этапе эквивалентных преобразований параллелизма и переходе к низкоуровневому описанию.

Ключевой особенностью предложенного метода синтеза являются механизмы сокращения (редукции) параллелизма задачи из исходного максимально-параллельного описания алгоритма под конкретные ресурсные ограничения платформы. В работах [6, 7, 11] показано, что такой подход обеспечивает переносимость параллельных алгоритмов на различные платформы, допуская также преобразование в императивные последовательные программы [13]. Для решения проблемы эффективного сокращения параллелизма необходимо выполнить оценку ресурсов и производительности получаемого архитектурного решения. Для этого требуется провести расчет коэффициента редукции по каждому классу ресурсов (R_i) и выполнить свертку параллелизма схемы для достижения требуемого коэффициента редукции R.

2. Оценка ресурсов

Для оценки ресурсов используется промежуточное представление – HDL-граф, в котором уже заданы архитектурно-зависимые данные. HDL-граф представляет собой ациклический граф в ярусно-параллельной форме, в каждом узле которого заданы типы и разрядности данных. При оценке ресурсов необходимо определить классы ресурсов H_i, по которым оценивается схема.

В качестве примера рассмотрим платформу ПЛИС, в которой выделим основные, характерные именно для ПЛИС, ресурсы:

- количество логических ячеек H_{lut};
- количество регистров H_{ff};
- количество блочной памяти H_{ram};
- количество арифметических и иных специализированных вычислительных блоков H_{dsp}.

Означенные ресурсы можно сгруппировать в два класса/подмножества: подмножество ресурсов памяти $H_{ram/ff} = \{H_{ff}, H_{ram}\}$ и подмножество вычислительных ресурсов $H_c = \{H_{lut}, H_{dsp}\}$:

$$H^{FF} = \sum H_i^{FF}. \tag{1}$$

Ресурсы внутри подмножества взаимозаменяемы с некоторыми ограничениями. Например, для ресурсов памяти любое хранение данных может быть реализовано как на блочной памяти, так и на триггерах, но ограниченное по объему. Для вычислительных ресурсов для любой арифметической/логической и т. д. операции можно реализовать схему на логических ячейках, в то время как тип и набор операций для специализированных блоков ограничен.

При оценке ресурсов памяти суммарный объем ресурсов, доступных на конкретной архитектурной платформе, может быть приведен к общему объему, измеренному в бит, Кбит и т. д.

Для оценки требуемого для конкретной реализации схемы ресурса рассмотрим HDL-граф схемы. Каждый i-тый слой ЯПФ формы состоит из набора информационных входов V_i и набора операций O_i. Каждый информационный вход вершин HDL-графа после типизации имеет разрядность W_i. Исходя из количества входов и разрядности каждого входа для каждого слоя графа, можно определить количество ресурсов памяти, требуемых для хранения результата на соответствующей стадии графа:

$$H_i^{FF} = \sum V_i * W_i. \tag{2}$$

Для расчета ресурса необходимо выполнить обход графа и суммирование разрядностей входов и выходов всех вершин:

$$H^{FF} = \sum H_i^{FF}. \tag{3}$$

После первоначальной оценки требуемого ресурса памяти для исходной максимально-параллельной реализации схемы возможны два варианта:

- требуемый ресурс меньше доступного $H^{FF} < H_{ram/ff}$;
- требуемый ресурс больше доступного $H^{FF} >= H_{ram/ff}$.

Первый вариант не требует расчета коэффициента редукции по ресурсам памяти $R_{ram/ff}$, но в случае изменения схемы при редукции по другим ресурсам оценку и проверку ресурса памяти необходимо повторить.

Во втором случае рассчитывается коэффициент редукции $R_{ram/ff}$:

$$R_{ram/ff} = H^{FF}/H_{ram/ff}. \tag{4}$$

Этот коэффициент далее используется в алгоритме редукции параллелизма схемы.

Помимо ограничения на объем памяти необходимо учитывать также ограничение на производительность памяти (ширина интерфейса данных). Для памяти, построенной на регистрах/триггерах, данного ограничения не существует, так как ширина шины данных равна объему данных. Для блочной памяти объем считываемых за такт данных меньше, чем объем хранимых данных. В случае, если ресурс H^{FF} превышает доступный объем регистров H_{ff}, необходимо рассчитывать суммарный интерфейс данных к памяти и коэффициент редукции по интерфейсу памяти $R_{ram/data}$.

Для реализации алгоритма расчета $R_{ram/data}$ необходимо определить минимально-необходимый коэффициент редукции по интерфейсу памяти. Для этого требуется выбрать из всех стадий конвейера набор стадий с максимальным суммарным, реализуемым на регистрах, интерфейсом. Поэтому из множества стадий конвейера выбирается подмножество стадий, такое что:

$$max \sum H_i^{FF} \&\& \sum H_i^{FF} < (H_{ram/ff} - H_f^{FF} f). \tag{5}$$

Тогда коэффициент $R_{ram/data}$ определяется как отношение суммарного интерфейса памяти оставшихся стадий к суммарному интерфейсу блочной памяти I_{ram}.

Для определения коэффициента редукции по интерфейсу памяти используется следующий алгоритм:

1. Выбрать подмножество стадий конвейера такое, что сумма ресурсов H_i^{FF} выбранных стадий будет меньше H_{ff}^{FF} и сумма выбранных значений H_i^{FF} будут максимальны.
2. Рассчитать ресурс памяти, реализуемый на блочной памяти/памяти с последовательным доступом:

$$H_{ram}^{FF} = H^{FF} - \sum H_i^{FF}, \tag{6}$$

   где i принадлежит подмножеству стадий конвейера, выбранных на шаге 1.
3. Рассчитать коэффициент $R_{ram/data}$:

$$R_{ram/data} = int(H_{ram}^{FF}/I_{ram}) + 1. \tag{7}$$

Рассмотрим в качестве примера вычисление 4-точечного быстрого преобразования Фурье (БПФ). В качестве входных типов данных зададим 16-ти битовый, знаковый.

Информационный граф после преобразования в HDL-граф и приведения в ярусно-параллельную форму показан на рисунке 1.

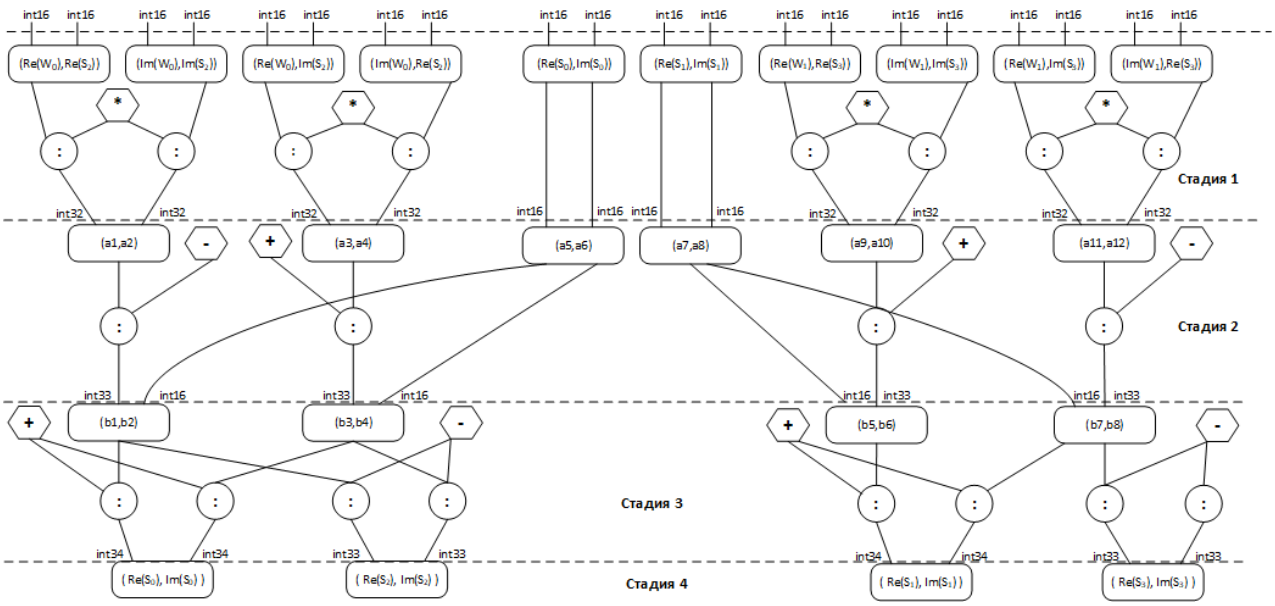

Fig. 1. HDL-graph max-parallel form of FFT size 4

Рис. 1. HDL-граф максимально-параллельной формы БПФ 4

Рассчитаем значение H_i^{FF} для каждой стадии конвейера:

H_1^{FF} = 6*2*16 = 192,

H_2^{FF} = 4*16 + 8*32 = 320,

H_3^{FF} = 4*16 + 4*33 = 196,

H_4^{FF} = 4*33 + 4*34 = 268.

Общее значение ресурса H^{FF} составит:

$$H^{FF} = \sum H_i^{FF} = 976 \text{ бит}. \tag{8}$$

Рассмотрим две архитектуры: A1 и A2. Значение $H_{ram/ff}$ для обеих архитектур равно 1536 бит. В архитектуре A1 весь ресурс памяти находится в регистрах, для такой архитектуры схема БПФ 4 в максимально-параллельной форме реализуется без изменений, как показано на рисунке 1.

В архитектуре A2 ресурс регистров составляет 512 бит и 1024 бит в виде блочной памяти с интерфейсом данных 36 бит. Значение суммарного интерфейса блочной памяти I_{ram} составит 36 бит. В соответствии с вышеприведенным алгоритмом выберем подмножество стадий, реализуемых на регистрах и имеющих максимальный интерфейс. В данном примере это может быть либо стадия 1, либо стадия 2.

Значение H_{ram}^{FF} в таком случае будет равно

$H_{ram}^{FF} = H^{FF} - H^{FF1} = 976 - 320 = 656$ бита.

Значение коэффициента редукции по интерфейсу памяти составит

$R_{ram/data} = H_{ram}^{FF}/I_{ram} = 656/36 = 19$.

В этом случае период подачи данных становится равным $R_{ram/data}$ и стадии конвейера 2,3,4 или 1,3,4 реализуются последовательно, так как результат записывается в один блок памяти.

HDL-граф после редукции на $R_{ram/data}$ представлен на рисунке 2. Запись значений a1-a12, вычисление значений b1-b8, S0-S3 производится в данном случае последовательно. Так как стадии 2,3 исходной схемы после редукции требуют 22 такта на выполнение, стадия 1 также может быть увеличена до 22 тактов без влияния на общую производительность системы, что приведет к пропорциональному уменьшению вычислительного ресурса стадии 1.

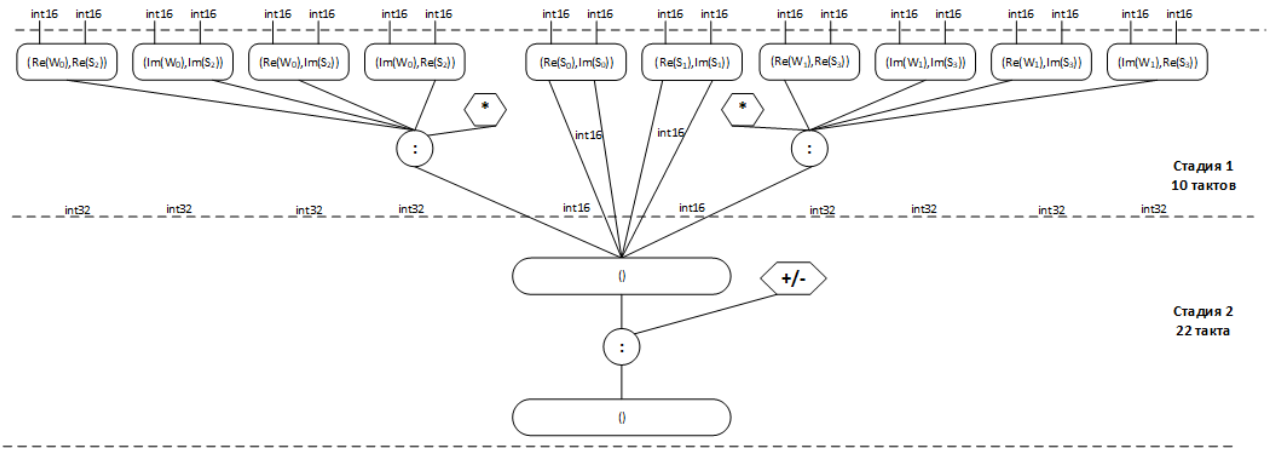

Fig. 2. HDL-graph after reduction

Рис. 2. HDL-граф после редукции

Для оценки вычислительных ресурсов рассмотрим слои (стадии) HDL-графа. Обозначим общее количество слоев (стадий конвейера) – M. На каждом j-том слое графа реализуется определенное множество операций $O^j = \{O_1^j, O_2^j \ldots O_i^j\}$. Для всего HDL-графа количество каждой операции типа i – N_i равно

$$N_i = \sum_{j=0}^{M} O_i^j. \quad (9)$$

Обозначим общее количество различных типов операций L, i=0. . . L. Под типом операции понимается тип арифметической/логической и т. д. операции вместе с указанием разрядностей данных, например, сложение 16 битных данных, сравнение 20 битных данных и т. д.

После расчета общего количества операций каждого типа необходимо исходя из доступного ресурса конкретной архитектуры оценить с какой степенью параллелизма возможно реализовать схему.

Методы и способы оценки ресурса для каждого конкретного типа операции рассмотрим далее, на данном этапе считаем, что количество ресурса для каждого конкретного типа операции известно.

Обозначим за $\dotplus$ – тип ресурса (логические ячейки, специализированные арифметические блоки DSP и т. д.), U_i^T – количество ресурса типа $\dotplus$, необходимое для реализации операции типа i. Тогда суммарный ресурс типа T для всех операций в HDL графе равен

$$U^T = \sum_{i=0}^{L} U_i^T * N_i. \tag{10}$$

По каждому классу вычислительных ресурсов можно рассчитать коэффициент редукции R_T как отношение суммарного требуемого ресурса к ресурсу целевой архитектуры, округленное до ближайшего большего целого:

$$R^T = int(U^T/H_T). \tag{11}$$

Итоговый коэффициент редукции по вычислительным ресурсам определяется как максимальный среди всех R_T:

$$R_{\text{выч}} = max\{R_T\}. \tag{12}$$

Рассмотрим данный подход на примере схемы графа изображенной на рисунке 1. Общее количество стадий конвейера данного графа M=4. Количество типов операций L составит 5: 16-битное умножение (i=0), сложение и вычитание 33 и 16 бит (i=1,2), сложение 34-битных данных и вычитание 33-битных данных(i=3,4). Количество операций по слоям графа составит:

$O^1 = 8, 0, 0, 0, 0,$

$O^2 = 0, 2, 2, 0, 0,$

$O^3 = 0, 0, 0, 4, 4,$

$O^4 = 0, 0, 0, 0, 0.$

Количество каждой операции i-го типа:

$N_0 = 8, N_1 = 2, N_2 = 2, N_3 = 4, N_4 = 4.$

В качестве примера возьмем архитектуру ПЛИС с двумя типами ресурсов для реализации вычислений: логические ячейки ($T = 0$) и DSP($T = 1$). Значения ресурса для каждого типа операции возьмем следующие:

$U_0^0 = 0, U_0^1 = 1$

$U_1^0 = 5, U_1^1 = 0,$

$U_2^0 = 5, U_2^1 = 0,$

$U_3^0 = 10, U_3^1 = 0,$

$U_4^0 = 10, U_4^1 = 0.$

Тогда, суммарный ресурс по каждому типу по всем операциям в графе составит:

$U^0 = \sum_{i=0}^{L} U_i^0 * N_i = 0 * 8 + 5 * 2 + 5 * 2 + 4 * 10 + 4 * 10 = 100,$

$U^1 = \sum_{i=0}^{L} U_i^1 * N_i = 1 * 8 + 0 * 2 + 0 * 2 + 0 * 10 + 0 * 10 = 8.$

Возьмем архитектуру, где количество DSP $H_1 = 2$, количество логических ячеек $H_0 = 400$. Тогда коэффициент редукции по каждому типу ресурсов составит:

$R_0 = int(U^0/H_0) = 100/400 = 0.25,$

$R_1 = int(U^1/H_1) = 8/2 = 4.$

Значение коэффициента редукции для вычислительных ресурсов $R_{\vee\cong\Xi}$ равно

$R_{\vee\cong\Xi} = max\{R_T\} = max\{0.25, 4\} = 4.$

После редукции каждой стадии с коэффициентом 4 задержка каждой стадии вырастет до 4 тактов, ресурс при этом снизится также в 4 раза. Результирующая схема приведена на рисунке 3. Необходимо также учитывать, что при изменении вычислительного ресурса может измениться и ресурс памяти.

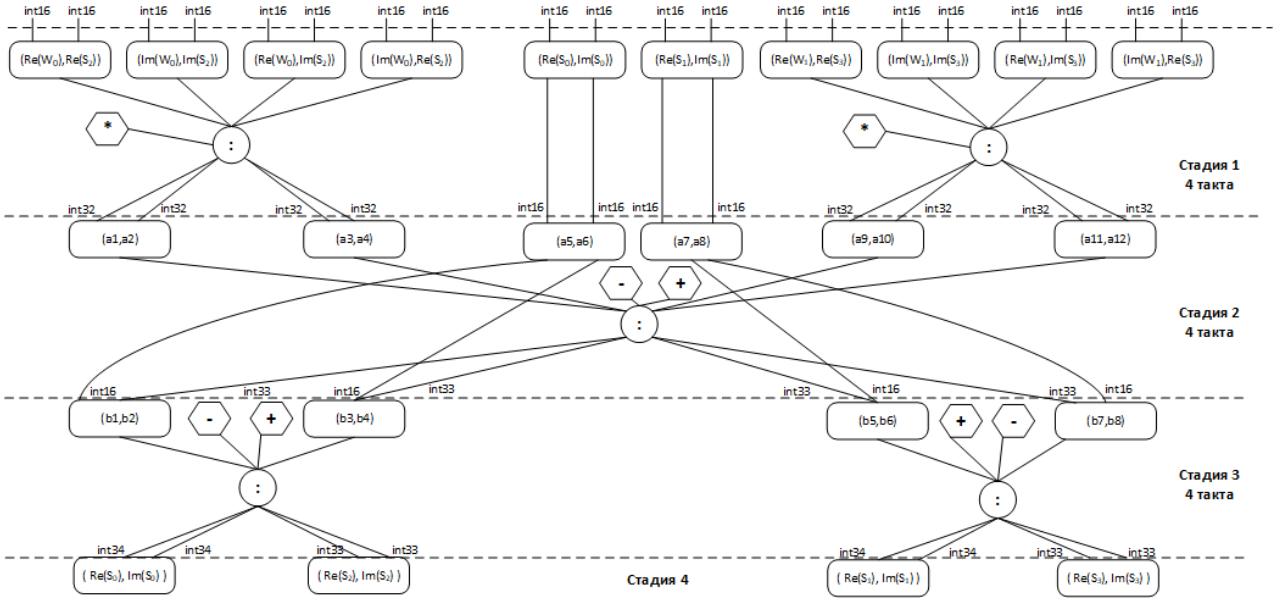

Fig. 3. HDL-graph after reduction with coefficient 4

Рис. 3. HDL-граф после редукции с коэффициентом редукции 4

3. Алгоритм преобразования параллелизма

Рассматривая задачу преобразования параллелизма в общем виде можно выделить 3 варианта преобразования исходной максимально-параллельной схемы в зависимости от доступного ресурса целевой платформы. Под доступным ресурсом целевой архитектуры при этом понимается наименьший из ресурсов – памяти или вычислительного ресурса.

Обозначим отношение доступного на платформе ресурса к ресурсу, требуемому для реализации в исходном максимально-параллельном виде, как P.

Возможные варианты отношения следующие:

- $P < 0.5$, возможна индукция (увеличение) количества схем;
- $0.5 < P < 1$, ресурса достаточно для размещения 1 максимально-параллельного варианта схемы;
- $P > 1$, необходима редукция параллелизма.

Первый вариант не требует преобразований максимально-параллельной схемы, при этом возможно увеличение производительности путем размещения нескольких схем параллельно:

$S = int(1/P)$.

Второй вариант схемы также не требует преобразований. Для третьего варианта схемы рассмотрим алгоритм с использованием коэффициентов редукции, описанных в выше.

Алгоритм редукции состоит из следующих шагов:

1. Рассчитать коэффициенты редукции R_{ram} и $R_{\text{выч}}$;
2. Выбрать максимальный коэффициент из R_{ram} и $R_{\text{выч}}$ - R_{max};
3. Редуцировать параллелизм схемы на R_{max};
4. Для измененной схемы пересчитать коэффициенты R_{ram} и $R_{\text{выч}}$, если они меньше 1, то следует завершение алгоритма;
5. Если какие-либо операции возможно реализовать с использованием другого типа ресурсов, то изменить данные операции на другой тип ресурсов без изменения коэффициента R и пересчитать коэффициенты R_{ram} и $R_{\text{выч}}$;
6. Если какой-либо из коэффициентов больше 1, то увеличить $R_{max} = R_{max} + 1$ и возврат на шаг 3.

На шаге 6 последовательное увеличение коэффициента редукции позволяет подобрать минимально-возможный коэффициент для удовлетворения требований по ресурсам и достижения при этом максимальной производительности (минимально-возможной редукции по производительности относительно исходного максимально-параллельного варианта).

4. Обмен метриками с архитектурно-зависимым уровнем

При оценке вычислительного ресурса необходимо иметь оценку ресурсоемкости операций, выделяемых в информационном HDL-графе. Данная оценка является архитектурно-зависимой, поскольку одна и та же операция на разных архитектурах занимает разный ресурс. Кроме этого набор базовых типов ресурсов, по которым выполняется оценка, так же может различаться на разных архитектурах.

Для описания архитектуры, требуемой при решении задачи оценки ресурсов, формируется модель, состоящая из 3 основных секций: секция описания типов ресурсов, секция описания общей емкости ресурсов платформы и секция описания ресурсоемкости базовых операций.

Для описания архитектуры, требуемой при решении задачи оценки ресурсов, формируется модель, состоящая из 3 основных секций: секция описания типов ресурсов, секция описания общей емкости ресурсов платформы и секция описания ресурсоемкости базовых операций. Секция описания содержит записи по количеству типов ресурсов. Каждая запись содержит общее количество данного типа ресурсов на конкретной целевой платформе. Описание ресурсоемкости операции содержит тип операции, количество операндов, разрядности операндов и количество по каждому типу ресурсов, необходимое для реализации данной операции с заданной разрядностью.

Рассмотрим в качестве примера описания ресурса платформу ПЛИС. Ресурсами памяти на платформе ПЛИС являются блоки памяти и регистры. Для вычислительных ресурсов это логические ячейки и арифметические блоки DSP. Пример описания типов ресурсов для платформы Xilinx Ultrascale приведен в таблице 1.

Table 1. Xilinx Ultrascale Platform Resource Types

Таблица 1. Типы ресурсов платформы Xilinx Ultrascale

Тип	Класс	Ед. измерения	Объем, бит	Размер типа данных, бит
LUT	Выч.	шт.	-	1
FF	Память	бит	1	1
BRAM	Память	бит	36864	36
DSP	Выч.	шт.	-	25

Описание общего количества ресурса для конкретной целевой платформы приведено в таблице 2.

Table 2. Resources for Virtex® UltraScale ™ FPGAs XCVU065

Таблица 2. Объем ресурсов для Virtex® UltraScale™ FPGAs XCVU065

Тип	Количество, шт
LUT	358080
FF	716160
BRAM	1260
DSP	600

Для операций, используемых в схеме на рисунке 1, секция описания ресурсоемкости приведена в таблице 3.

Как видно из таблицы 3, некоторые операции могут быть реализованы на разных типах вычислительных ресурсов, что необходимо учитывать в алгоритме редукции параллелизма, поскольку это расширяет количество вариантов реализации схемы без изменения коэффициента редукции R.

Table 3. Resource requirement for Virtex® UltraScale ™ XCVU065 FPGA

Таблица 3. Требуемый ресурс для ПЛИС Virtex® UltraScale™ XCVU065

Тип операции	Количество операндов	Разрядность операндов	LUT	FF	BRAM	DSP
+	2	33/33	33	0	0	0
-	2	33/33	33	0	0	0
+	2	34/34	34	0	0	0
-	2	34/34	34	0	0	0
+	2	32/16	32	0	0	0
-	2	32/16	32	0	0	0
*	2	16/16	0	0	0	1
+	2	33/33	0	0	0	1
-	2	33/33	0	0	0	1
+	2	34/34	0	0	0	1
-	2	34/34	0	0	0	1
+	2	32/16	0	0	0	1
-	2	32/16	0	0	0	1

5. Практические результаты

Для оценки полученных результатов, приведенные на рисунках 1–3 схемы были синтезированы с помощью входящего в разработанный пакет прикладных программ синтезатора HDL-описания с языка ФПП [14]. Для оценки ресурсов использована платформа Xilinx Ultrascale+. Синтез полученного HDL-описания производился с использованием САПР Vivado 2019.1. [15]. Оценка ресурсов производилась по 4 классам: логические ячейки (LUT), регистры (FF), память (BRAM) и арифметические блоки (DSP).

В качестве базы для сравнения взят максимально-параллельный вариант, приведенный на рисунке 1. Результаты синтеза и оценки ресурсов для базы приведены в таблице 4. Так как для синтеза арифметических операций на данной платформе существуют два варианта – использование LUT и использование DSP, в данной реализации использован сбалансированный вариант – синтез операций умножения с использованием DSP блоков и синтез операций сложения с использованием LUT.

Table 4. Estimating resources of case with maximum parallelism

Таблица 4. Оценка ресурсов базового варианта с максимальным параллелизмом

Схема	LUT	FF	BRAM	DSP
Максимально-параллельная схема	392	980	0	8
Редукция по памяти/регистрам	457	332	1	1
Редукция по вычислительным ресурсам	398	889	0	2

Результаты синтеза схемы, редуцированной по интерфейсу памяти/регистров, приведенной на рисунке 2, отражены в таблице 4. Полученный ресурс соответствует требованиям архитектуры A2.

Из таблицы 4 видно, что ресурс логических ячеек увеличился по сравнению с вариантом «до редукции». Это связано с накладными расходами, требуемыми для реализации схемы сериализации/мультиплексирования расчетов на меньшем количестве блоков. Для конечной реализации данный ресурс требует оценки на высокоуровневом этапе. Проблема разницы в оценке ресурсов на высокоуровневом и низкоуровневом этапе требует дальнейшего изучения.

Результаты синтеза схемы, редуцированной по вычислительным ресурсам (рисунок 3), приведены в таблице 4. Результат редукции по вычислительным ресурсам DSP ячеек соответствует требованиям, но при этом количество логических ячеек не изменилось при редукции. Этот эффект

также связан с накладными расходами при переходе к более последовательной схеме и требует учета при оценке ресурсов и дальнейшем выборе коэффициента редукции в п.4 вышеприведенного алгоритма.

Заключение

Представленные методы редукции позволяют реализовать изменение параллелизма исходного описания алгоритма и обеспечить реализацию механизма переноса на различные архитектуры с разными ресурсными ограничениями. В отличии от методов индукции параллелизма предложенный метод снижает сложность процесса переноса за счет уменьшения перебора количества вариантов реализации, получаемых в процессе синтеза под новые ресурсные ограничения.

Вместе с тем методы оценки ресурса на высокоуровневом этапе требуют учета накладных расходов при изменении параллелизма к более последовательным схемам. Текущий вариант оценки требует увеличения точности оценки. Для этого могут использоваться нейронные сети и методы машинного обучения, которые на основе параметров оценки схемы на высокоуровневом этапе могут с необходимой точностью предсказать значения параметров схемы после реализации на низком уровне.

Методы прямого подсчета ресурса результата реализации схемы осложнены ввиду множества преобразований, которые происходят при реализации схемы на этапах синтеза и имплементации на низкоуровневом этапе предлагаемого метода. Реализация точных методов оценки ресурса позволит дополнительно уменьшить количество вариантов схемы, рассматриваемых в процессе синтеза под ресурсные ограничения.

References

[1] T. M. Bhatt and D. McCain, “Matlab as a Development Environment for FPGA Design”, in *Design Automation Conference, Proceedings 42nd*, 2005, pp. 607–610.

[2] L. Lavagno, I. L. Markov, G. Martin, and L. K. Scheffer, *Electronic Design Automation for IC System Design, Verification, and Testing*. CRC Press, 2017.

[3] Z. Navabi, *System-Level Design and Modeling: ESL Using C/C++, SystemC and TLM-2.0*. Springer, 2015.

[4] *Vivado Design Suite User Guide. High-Level Synthesis. UG902.* [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_1/ug902-vivado-high-level-synthesis.pdf.

[5] Z. Yuan, Y. Ma, J. Bian, and K. Zhao, “Automatic enhanced CDFG generation based on runtime instrumentation”, in *IEEE 17th International Conference on*, 2013, pp. 92–97.

[6] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Hérault, and J. J. Dongarra, “PaRSEC: Exploiting Heterogeneity to Enhance Scalability”, *IEEE Computing in Science and Engineering*, vol. 15, no. 6, pp. 36–45, 2013.

[7] A. Danalis, G. Bosilca, A. Bouteiller, T. Herault, and J. Dongarra, “PTG: An Abstraction for Unhindered Parallelism”, in *Proceedings of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*, 2014, pp. 21–30.

[8] I. I. Levin and A. I. Dordopulo, “Resource-independent programming of hybrid reconfigurable computer systems”, in *Proceedings of Russian Supercomputing Days*, 2017, pp. 714–723.

[9] I. I. Levin, A. I. Dordopulo, I. V. Pisarenko, and A. K. Melnikov, “An approach to architecture-independent programming of computing systems based on the aspect-oriented Set@l language”, *Proceedings of the Southern Federal University. Technical sciences*, vol. 197, no. 3, pp. 46–57, 2018.

[10] O. V. Nepomnyaschy, I. N. Ryzhenko, and A. I. Legalov, "The method of architecturally independent high-level synthesis of VLSI", *Proceedings of the Southern Federal University. Technical sciences*, vol. 202, no. 8, pp. 38–47, 2018.

[11] A. I. Legalov, "Functional language for creation of architectural independent parallel programmes", *Computational Technologies*, vol. 10, no. 1, pp. 71–89, 2005.

[12] A. I. Legalov, V. S. Vasilyev, I. V. Matkovskii, and M. S. Ushakova, "A toolkit for the development of data-driven functional parallel programmes", in *International Conference on Parallel Computational Technologies, Springer*, 2018, pp. 16–30.

[13] V. Vasilev, A. I. Legalov, and S. V. Zykov, "The System for Transforming the Code of Dataflow Programs into Imperative", *Modeling and Analysis of Information Systems*, vol. 28, no. 2, pp. 198–214, 2021.

[14] O. V. Nepomnyaschy and I. N. Ryzhenko, "The method of high-level synthesis and software toolkit for description algorithm of VLSI", *Software Engineering*, vol. 11, no. 1, pp. 34–39, 2020.

[15] *Vivado Design Suite User Guide. Using the Vivado IDE UG893.* [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug893-vivado-ide.pdf.