ПРОГРАММИРОВАНИЕ, 2019, № 5, с. 43–55

ТЕОРИЯ ПРОГРАММИРОВАНИЯ: ФОРМАЛЬНЫЕ МОДЕЛИ И СЕМАНТИКА

УДК 004.415.2

ГРАФООРИЕНТИРОВАННЫЙ ПРОГРАММНЫЙ КАРКАС ДЛЯ РЕАЛИЗАЦИИ СЛОЖНЫХ ВЫЧИСЛИТЕЛЬНЫХ МЕТОДОВ

© 2019 г. А. П. Соколов[a],*, А. Ю. Першин[a],**

[a] *Московский государственный технический университет имени Н.Э. Баумана*
105005 Москва, ул. 2-ая Бауманская, д. 5, стр. 1, Россия

**E-mail: alsokolo@bmstu.ru*

***E-mail: apershin@bmstu.ru*

Поступила в редакцию 09.10.2018 г.
После доработки 18.11.2018 г.
Принята к публикации 26.11.2018 г.

В работе представлен графоориентированный подход к реализации сложных вычислительных методов, который ставит своей целью обеспечение инженера-математика набором инструментов для организации своего кода, который позволяет уменьшить трудозатраты на его дальнейшие верификацию, валидацию и поддержку. Предлагаемый графоориентированный подход базируется на трех уровнях абстракции: категориальном (описание преобразований данных алгоритма в алгебраических терминах теории категорий), графовом (описание алгоритма в виде ориентированного графа) и интерфейсном (конкретная программная реализация инструментов для построения алгоритма на основе использования введенного в работе понятия графовой модели сложного вычислительного метода). Представлены отдельные особенности реализованного программного каркаса для создания и обхода графовых моделей. Представлен пример эффективного использования графоориентированного подхода при разработке программных реализаций вычислительных методов микромеханики композиционных материалов.

DOI: 10.1134/S0132347419050066

1. ВВЕДЕНИЕ

Вопросы проектирования архитектуры, разработки и гибкости дальнейшей поддержки программного обеспечения всегда широко обсуждались и были ключевыми при его создании. Многие подходы, такие как использование различных парадигм программирования, шаблонов проектирования, систем контроля версий, идеологии “гибкой” разработки программных продуктов, юнит-тестирование и проч., выработавшихся с момента появления первых программ, в настоящее время de facto стали стандартом в области индустриального программирования. Большинство из них появились как ответ на конкретные потребности производителей программного обеспечения. В то же время, в области научного программирования подобные стандарты могут вырабатываться с заметным опозданием или не вырабатываться вовсе, что зачастую приводит к отсутствию какого-либо системного подхода в написании научного кода [1]. Часто это объясняется сложностью и глубиной предметной области разрабатываемого кода, что отводит вопрос его организации и качества на второй план. Кроме того, научное программирование чаще всего не предполагает создание конечного продукта, как это происходит в индустриальном программировании [2]. Такая ситуация может в дальнейшем приводить к сложностям в верификации, валидации и поддержке кода, что в свою очередь приводит к необоснованному доверию научному программному обеспечению, используемому конечным пользователем [3].

При разработке программных реализаций численных методов предполагается, что исходный код должен быть одновременно (1) эффективным с точки зрения необходимого машинного времени и памяти, (2) адекватно отражающим моделируемый процесс (т.е. прошедшим верификацию и валидацию тем или иным способом) и (3) удобным для дальнейшей поддержки. Большинство работ по реализации численных методов в основном посвящены первому пункту [4]. Концепции верификации и валидации широко обсуждаются как на исследовательском [5, 6], так и на административном [7] уровнях, так как их игнорирование приводит к общему недоверию к результатам расчетов и, зачастую, невозможности предсказать то или иное явление в критически важный момент [8]. Однако, даже эффективный и корректный код может оказаться бесполезным, если его поддержка оказывается затруднительной, что мотивирует накопление и публикацию “лучших

практик", применяемых при программировании численных методов [9]. Во многом возможность будущей поддержки кода, реализующего численный метод, определяется также тем, насколько легко написанный код усваивается новым разработчиком. Следует также отметить, что игнорирование лучших практик организации кода ведет к невозможности использовать юнит-тестирование кода, которое позволяет упростить верификацию и валидацию программного обеспечения.

Трудоемкость, заложенная в изложенных выше требованиях к научному коду, в частности, определяется и системой организации кода. Многие существующие платформы для создания вычислительных моделей ориентированы на компонентную модель, позволяющую минимизировать написание кода за счет использования визуального программирования [10–12]. Одними из наиболее значимых разработок в этом сегменте являются LabView и Simulink.

LabView используется в системах сбора и обработки данных, а также для автоматизации управления техническими объектами и технологическими процессами (АСУ ТП) по аналогии с системами класса SCADA. Помимо решения задач в области АСУ ТП LabView в большей степени ориентирована на решение задач автоматизации научных исследований. Основное назначение LabView – построение моделей технических систем и устройств, для этой цели LabView включает в состав множество специализированных библиотек реализованных компонентов, на базе которых возможно построение новых систем из конкретных технических областей.

Еще одной известной системой визуального моделирования является пакет Simulink, входящий в поставку программной системы MatLab. Simulink – это графическая среда имитационного моделирования, позволяющая при помощи блок-диаграмм в виде ориентированных графов, строить модели динамических, дискретных, непрерывных и гибридных, нелинейных и разрывных систем. Simulink также ориентирован на решение задач моделирования сложных технических систем и представляет собой модельно-ориентированную систему проектирования. В первую очередь Simulink позволяет создавать графическую модель системы, собрав ее на основе стандартных базовых элементов (интегратор, звено задержки, сумматор, произведение, справочная таблица, мультиплексор, переключатель, селектор шины и пр.), которой будет соответствовать конкретная математическая модель системы на основе систем обыкновенных дифференциальных уравнений или уравнений в частных производных.

Существуют также специализированные вычислительные каркасы (фреймворки), позволяющие реализовать вычислительные модели, к примеру, на основе готовых решателей с использованием символьного языка описания уравнений в частных производных.

Примером является система визуального моделирования FEniCS [13, 14]. В первую очередь система FEniCS ориентирована на задачи, которые могут быть решены при помощи метода конечных элементов. FEniCS позволяет пользователям быстро преобразовывать математические модели процессов моделирования в удобный и просто сопровождаемый конечно-элементный код. FEniCS является кроссплатформенной системой с открытым кодом и предоставляет удобные интерфейсы для доступа к функциям с использованием языков программирования Python и C++. Ряд реализованных алгоритмов FEniCS могут задействовать высокопроизводительные многопроцессорные вычислительные ресурсы. Одна из важнейших особенностей FEniCS – реализация гибкого механизма ввода математических моделей буквально в исходной интегро-дифференциальной форме. В рамках FEniCS реализованы удобные механизмы работы с конечно-элементными расчетными сетками, функциями решения систем линейных алгебраических уравнений и прочее [13].

Также следует отметить систему TensorFlow. TensorFlow представляет собой библиотеку с открытым исходным кодом, для построения программных реализаций численных методов. В основе TensorFlow также лежат понятия теории графов, а также понятие граф потоков данных. Принципиальным отличием TensorFlow от представляемой в статье разработки являются различия в определениях узлов и ребер. Для TensorFlow узлы представляют математические операции, а ребра – многомерные массивы данных(тензоры), тогда как для представляемого способа узлы определяют фиксированные состояния общих данных (которые, в том числе, могут включать массивы данных), а ребра определяют функции преобразования данных.

В том числе известны и другие системы, созданные ранее, но которые не нашли столь широкого распространения: GRIDS, FUNSOFT. Следует отметить также и другие узкоспециализированные системы и языки программирования: PTOLEMY II, ProGraph, Apache Storm, IBM InfoSphere Streams, Erlang, PROGRES, Cantata, LUSTRE, KLIEG, DOME, GME, Atom3, Moses, DRACO, GenVoca, Datadvance (pSeven).

Более универсальным, но и низкоуровневым, является применение процедурного и/или объектно-ориентированного подходов при создании каркасов вычислительных библиотек. Многие прикладные разработки созданы именно с использованием этих подходов (BLAS (C, ASM), BOOST (C++), Netgen/NGSolve (C++), ROOT

(C++), OpenFoam (C++) [15] и многие другие). Особенностью доработки и/или расширения возможностей созданных прикладных инженерных разработок на этой основе является требование перегрузки операторов и/или виртуальных методов, предоставляемых фиксированными интерфейсами используемых базовых вычислительных библиотек.

Представленные подходы зачастую не предполагают разработку различных численных методов, используя унифицированные подходы, и почти всегда являются проблемно-ориентированными (например, концентрируются на моделировании уравнений в частных производных методом конечного элемента, к примеру NetGen и NGSolve [16]).

В данной работе мы представляем графоориентированный подход к реализации сложных вычислительных методов, который ставит своей целью обеспечение инженера-математика набором инструментов для организации своего кода, позволяющим уменьшить трудозатраты на его дальнейшие верификацию, валидацию и поддержку. Предлагаемый графоориентированный подход базируется на трех уровнях абстракции: категориальном (описание преобразований данных алгоритма в алгебраических терминах теории категорий), графовом (описание алгоритма в виде ориентированного графа) и интерфейсном (конкретная программная реализация инструментов для построения алгоритма на основе графовой модели). Все три уровня рассматриваются последовательно в дальнейших разделах, после чего приводятся примеры использования графоориентированного подхода для программной реализации сложных вычислительных методов.

2. КАТЕГОРИЯ СОСТОЯНИЙ СЛОЖНОГО ВЫЧИСЛИТЕЛЬНОГО МЕТОДА

Алгебраическим фундаментом графоориентированного подхода является категория состояний $\mathscr{C}$ сложного вычислительного метода (СВМ) с копроизведением. Теория категорий активно используется в прикладных направлениях науки для выявления обобщенных свойств графовых моделей обработки сигналов [17, 18], сетей [19] и анализа вычислимости [20]. Отметим, что в данной статье представлено только определение категории состояний, ее объектов, морфизмов и копроизведения. Дальнейшее развитие категории состояний и ее анализ являются частью будущих работ.

Для построения категории состояний $\mathscr{C}$ с копроизведением необходимо [21]: (1) определить объекты, морфизмы и копроизведение, (2) определить ассоциативную композицию морфизмов и (3) задать тождественный морфизм. Объектами категории $\mathscr{C}$ являются *состояния СВМ*, которые принадлежат соответствующему *пространству состояний СВМ* $\mathbb{S}$. Пространство состояний СВМ в свою очередь строится на основе *множества элементарных состояний СВМ*.

Определение 2.1. Множеством элементарных состояний СВМ будем называть множество $\mathbb{W}$ пар “имя параметра – множество допустимых значений параметра” таких, что $\mathbb{W} = Ind(String) \times \bigcup_{i=0}^{N_\mathrm{T}} \{Ind(T_i)\}$, где $Ind(X)$ – оператор индукции, определяющий множество всех возможных значений типа X, *string* – тип строк в заданном языке программирования и T_i – i-й произвольный тип заданного языка программирования (число типов N_T бесконечно в случае языков с пользовательскими типами).

Следующие пары могут служить примерами элементов множества элементарных состояний СВМ:

$$s_a = (\text{«}RealParam\text{»}, \mathbb{R}), \tag{1}$$

$$s_b = (\text{«}IntegerParam\text{»}, \mathbb{Z}), \tag{2}$$

$$s_c = (\text{«}StringParam\text{»}, Ind(\mathrm{string})), \tag{3}$$

где $s_a, s_b, s_c \in \mathbb{W}$. Здесь предполагается, что множества $\mathbb{R}$ и $\mathbb{Z}$ индуцированы гипотетическими типами неограниченных вещественных и целых чисел с бесконечной точностью.

Определение 2.2. Пространством состояний СВМ $\mathbb{S}$ будем называть множество таких подмножеств $S_i \subset \mathbb{W}$ множества элементарных состояний СВМ, что элементы каждого такого подмножества имеют уникальные имена в рамках этого подмножества, т.е. $\forall s_1,\ s_2 \in S_i : s_1 \neq s_2 \Rightarrow pr_1(s_1) \neq pr_1(s_2)$, где pr_1 – проекция декартова произведения на первую координату соответствующей пары.

Определение 2.3. Состоянием СВМ будем называть элемент $S \in \mathbb{S}$.

В контексте теории категорий состояния СВМ являются объектами, то есть $S \in \mathscr{C}$, а преобразования между ними описываются морфизмами специального вида. Состояние СВМ может иметь конкретную реализацию, называемую *данными СВМ*.

Определение 2.4. Пусть $S \in \mathscr{C}$ – состояние СВМ. Тогда множество D, такое, что $\forall s \in S, \exists d \in D$: $d = (n, v)$, $n = pr_1(s)$, $v \in pr_2(s)$, будем называть данными СВМ, находящимися в состоянии S.

Определение 2.5. Пусть $S_{in}, S_{out} \in C$. Тогда морфизмом f из S_{in} в S_{out} будем называть отображение следующего вида:

$$\begin{gathered} f\colon pr_2(s_1) \times pr_2(s_2) \times \ldots \times pr_2(s_n) \to \\ \to pr_2(\tilde{s}_1) \times pr_2(\tilde{s}_2) \times \ldots \times pr_2(\tilde{s}_m), \end{gathered} \tag{4}$$

где $s_1, s_2, \ldots, s_n \in S_{in}$, $\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_m \in S_{out}$, $n = |S_{in}|$ и $m = |S_{out}|$. Морфизм f будем также называть функ-

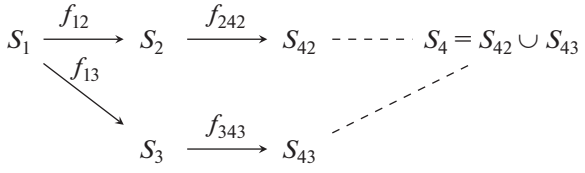

Рис. 1. Диаграмма преобразований состояния S_1, состоящая из двух параллельных участков (из S_1 в S_{42} и из S_1 в S_{43}), которые объединяются в состояние S_4 путем копроизведения конечных состояний на каждом из участков: $S_4 = S_{42} \cup S_{43}$.

цией-обработчиком. Объект S_{in} будем называть входным состоянием СВМ f, а объект S_{out}, в свою очередь, выходным состоянием f. Преобразование данных D_{in}, находящихся в состоянии СВМ S_{in}, в данные D_{out}, находящиеся в состоянии СВМ S_{out}, с помощью функции-обработчика f будем обозначать $D_{out} = f(D_{in})$.

Функции-обработчики можно разделить на два типа. Функции-обработчики первого типа отображают множество $S_{in} \in \mathscr{C}$ само в себя, т.е. $S_{out} = S_{in}$, и в таком случае являются эндоморфизмами. Функции-обработчики второго типа помимо отображения S_{in} дополняют это множество новыми элементами. Функции-обработчики второго типа отличаются внесением изменений в структуру множества S_{out} относительно множества S_{in}: они могут добавлять или удалять элементы. Заметим, что легко убедиться, что существует композиция морфизмов $f_n \circ f_{n-1} \circ \ldots \circ f_1$, где операция композиции удовлетворяет аксиоме ассоциативности. Наконец, для любого $S \in \mathscr{C}$ можно построить тождественный морфизм id_S: $S \to S$.

Копроизведение в категории состояний интерпретируется как объединение состояний и необходимо для объединения конечных состояний нескольких параллельных композиций. Следует отметить, что копроизведение определено только для тех состояний, имена параметров которых не пересекаются.

Определение 2.6. Копроизведением состояний $S_1, S_2 \in \mathscr{C}$, где $\{pr_1(s) | s \in S_1\} \cap \{pr_1(s) | s \in S_2\} = \varnothing$, будем называть состояние $S = S_1 \cup S_2$.

Также необходимо ввести вспомогательное понятие *разбиение состояния СВМ*, которое потребуется при определении понятия графовая модель СВМ.

Определение 2.7. Разбиением состояния СВМ $S \in \mathscr{C}$ будем называть такое семейство объектов $\{S_i\}_{i=1}^n$, что $S_i = \{(pr_1(s), D) | s \in S_i, D \subset pr_2(s)\}$, при этом $\forall s \in S, \exists s_1 \in S_1, \ldots, s_n \in S_n$: $pr_1(s_1) = \ldots = pr_1(s_n) \Rightarrow \bigcup_{i=1}^n pr_2(s_i) = pr_2(s)$, $pr_2(s_i) \cap pr_2(s_j) = \varnothing$, где $i \neq j$.

Таким образом, построенная категория состояний с копроизведением позволяет описать произвольный алгоритм, состоящий из последовательных участков преобразования (переходов из начальных состояний в конечные через композицию морфизмов) и параллельных участков преобразования, конечные состояния которых образуются через копроизведение конечных состояний соответствующих последовательных участков преобразования, что проиллюстрировано на рис. 1.

3. ГРАФОВАЯ МОДЕЛЬ СЛОЖНОГО ВЫЧИСЛИТЕЛЬНОГО МЕТОДА

Графовая модель ставит своей целью графическое описание программной реализации алгоритма СВМ, используя ранее введенные понятия категории состояний $\mathscr{C}$. Построенная категория не учитывает, что в реальной программной реализации алгоритма СВМ могут формироваться некорректные данные (как из-за неидеальности программных реализаций морфизмов, так и из-за изначально некорректных входных данных), что мотивирует введение механизма обработки исключительных ситуаций. Действительно, несмотря на то, что функции-обработчики могут отображать целые множества, зачастую желаемый результат будет формироваться только для некоторых подмножеств. Таким образом представляется разумным реализация *функции-предиката* для каждой функции-обработчика, которая будет задавать разбиение $\{S_{true}, S_{false}\}$ для входного состояния $S_{in} \in \mathscr{C}$ функции-обработчика и определять для каких данных f могла бы быть использована.

Определение 3.1. Пусть $S_{in} \in \mathscr{C}$ – входное состояние СВМ функции-обработчика f, $\mathbb{D} = \{D_i\}_{i=1}^n$ – семейство всех возможных данных СВМ, находящихся в состоянии S_{in}, и $\{S_{true}, S_{false}\}$ – разбиение состояния S_{in}. Тогда функцией-предикатом будем называть такое отображение $p : \mathbb{D} \to \{0,1\}$, что элементы прообраза $\mathbb{D}_{true} = p^{-1}(\{1\})$ находятся в состоянии S_{true} и элементы прообраза $\mathbb{D}_{false} = p^{-1}(\{0\})$ находятся в состоянии S_{false}, где $\mathbb{D}_{true} \cup \mathbb{D}_{false} = \mathbb{D}$. При этом состояние S_{true} будем называть валидным, а состояние S_{false} соответственно невалидным.

Таким образом предполагается, что функция-обработчик f может быть использована только в том случае, когда $p(D_{in}) = 1$, где D_{in} – некоторые входные данные, находящиеся во входном состоянии функции-обработчика f. Тогда случай $p(D_{in}) = 0$ понимается как исключительная ситуация. Объединенную функцию будем называть *функцией пе-*

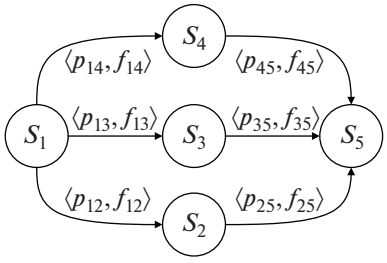

Рис. 2. Графовая модель с параллельными ветвями выполнения.

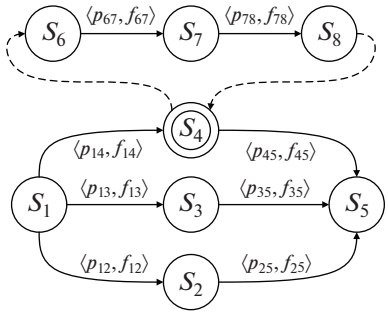

Рис. 3. Графовая модель с иерархической вложенной графовой моделью. В данном случае состояние S4 заменяется на графовую модель, состоящую из трех состояний S6, S7 и S8 и соответсвующих функций перехода.

рехода и обозначать $\langle p, f\rangle$. Ее исполнение осуществляется в соответствии с тернарным оператором языка C/C++: $\langle p, f\rangle(D_{in}) = p(D_{in})?f(D_{in}) : D_{in}$. С точки зрения категории $\mathscr{C}$ функция перехода $\langle p, f\rangle$ несет ту же семантику морфизма состояний СВМ, что и функция-обработчик f, в то время как с алгоритмической точки зрения функция перехода в том числе организует и обработку исключительных ситуаций.

Алгоритмическую структуру переходов между состояниями СВМ удобно представлять в виде графа, в частности с помощью графа потока управления [22]. В данной работе понятие графа потока управления расширяется и вводится понятие *графовой модели СВМ*.

Определение 3.2. *Графовой моделью СВМ* будем называть тройку объектов (G, Σ, Φ), где $G = (Nodes, Edges)$ – ориентированный граф с узлами *Nodes* и ребрами *Edges*, Σ – множество состояний СВМ и Φ – множество функций перехода. Каждому узлу ставится в соответствие состояние $S \in \Sigma$, в то время как каждому ребру ставится в соответствие функция-перехода $\phi \in \Phi$.

Если из состояния $S \in \mathscr{C}$ выходят несколько ребер, то возникает несколько одновременных переходов из состояния S в другие состояния, что понимается как распараллеливание потока выполнения. Более того, если соответствующие функции-предикаты $p_1, p_2, \ldots, p_n$ заданы таким образом, что их валидные состояния $S^{(1)}_{true}, S^{(2)}_{true}, \ldots, S^{(n)}_{true}$ не пересекаются, то имеет место ветвление потока выполнения, эквивалентное оператору выбора switch языка C/C++. Такие функции-предикаты называются *взаимоисключающими*. Случай, когда несколько ребер входят в одно состояние, интерпретируется как слияние параллельных потоков выполнения. Полученное состояние является результатом объединения выходных состояний ребер и описывается копроизведением, заданным в категории $\mathscr{C}$ (определение 2.6). Таким образом представленная графовая модель способна описывать алгоритмы с параллельными ветвями и произвольным условным ветвлением унифицированным образом.

Пример графовой модели, которая в зависимости от свойств предикатов может представлять последовательный или параллельный алгоритм, изображен на рис. 2. К примеру, пусть $S^{(12)}_{true}$, $S^{(13)}_{true}$ и $S^{(14)}_{true}$ – валидные состояния, задаваемые функциями-предикатами p_{12}, p_{13}, p_{14} и определяемые в соответствии с определением 3.1. Тогда в случае $S^{(12)}_{true} \cap S^{(13)}_{true} = \varnothing$, $S^{(12)}_{true} \cap S^{(14)}_{true} = \varnothing$, $S^{(13)}_{true} \cap S^{(14)}_{true} = \varnothing$ графовая модель соответствует последовательному потоку выполнения с условным ветвлением, так как функции-предикаты p_{12}, p_{13}, p_{14} являются взаимоисключающими. Если же $S^{(12)}_{true} = S^{(13)}_{true} = S^{(14)}_{true}$, графовая модель соответствует трем параллельным потокам выполнения.

По аналогии с диаграммами потоков данных (DFD) узлы графовой модели могут представлять собой вложенные графовые модели, что позволяет создать иерархическую структуру неограниченной сложности и при необходимости абстрагироваться от деталей реализации. Рисунок 3 демонстрирует пример простой графовой модели с подобной вложенностью.

Ключевым отличием предложенной графовой модели от других моделей описания потоков данных (например, data flow [23] и flow-based programming [24]) является определение узлов графа как состояний данных, а не функций, обрабатывающих данные. Указанная особенность делает графовую модель близкой по сути к конечному автомату. Состояния данных имеют конкретную семантику и, в общем случае, произвольные названия, что делает графовую модель самодокументируемой: она описывает не только последовательность преобразований, но и последовательность состояний, в которых оказываются данные после этих преобразований.

4. ПРОГРАММНЫЙ КАРКАС ДЛЯ СОЗДАНИЯ И ОБХОДА ГРАФОВОЙ МОДЕЛИ

Для создания графовых моделей был разработан ряд программных инструментов, объединенных в программных каркас *comsdk*, написанный на языке C++ и являющийся частью “Распределенной вычислительной системы GCD” [25]. Программный каркас *comsdk* включает в себя API для создания состояний СВМ (узлов графовой модели) и их объединения с помощью функций-предикатов и функций-обработчиков. Функции-предикаты и функции-обработчики являются функциями или методами классов C++ с заранее заданной и фиксированной сигнатурой, использование которых возможно на базе статической или динамической компоновки. Программный каркас также содержит в себе функции для запуска обхода графа с учетом условных ветвлений и параллельных веток.

Данные D должны быть представлены в форме словаря со строковым ключом и произвольным типом хранимых данных. Многие интерпретируемые языки программирования имеют базовую поддержку словарей со множественными типами значений (например, dict в Python). В более общем случае для реализации подобного словаря достаточно, чтобы используемый язык программирования поддерживал динамическую идентификацию типа данных (RTTI), которая свойственна многим языкам с объектно-ориентированной парадигмой программирования. В данной реализации программного каркаса *comsdk* использовался словарь, написанный на языке C++ с помощью boost::any. Формирование исходных данных и заполнение словаря для подачи на вход графовой модели реализуется с помощью файлов aINI-формата [26].

Вне зависимости от используемого языка программирования данные D должны передаваться в функции-предикаты и функции-обработчики по ссылке без использования операции копирования. Копирования не происходит и при параллелизации потока выполнения – предполагается, что функции-обработчики в различных потоках преобразуют различные элементы данных D для избежания неопределенности параллелизма (race condition).

Для реализации различных методов выполнения параллельных веток, было введено понятие *стратегия распараллеливания*. При создании состояний СВМ с помощью программного каркаса, разработчик графовой модели указывает, какая стратегия распараллеливания будет использована при распараллеливании потока выполнения в случае двух и более выходных ребер состояния. Кроме встроенных в программный каркас стратегий распараллеливания (псевдопараллелизм, многопоточность и многопроцессность) разработчик графовой модели может добавить и использовать свои собственные реализации стратегий. Стратегией распараллеливания по умолчанию является псевдопараллелизм.

Кроме стратегии распараллеливания каждое состояние, создаваемое с помощью программного каркаса, предполагает задание *стратегии выбора*. Стратегия выбора определяет, каким образом при наличии двух и более выходных ребер следует интерпретировать значения, возвращаемые функциями-предикатами. Например, в случае распараллеливания потока выполнения предполагается, что функции-предикаты всех ребер должны вернуть 1 для продолжения корректного исполнения, а возврат хотя бы одного 0 интерпретируется как ошибка. В программном каркасе подобная стратегия выбора называется “all or nothing” и является стратегией выбора по умолчанию. В случае же условного ветвления потока выполнения, функции-предикаты являются взаимоисключающими и для корректного исполнения только одна из них должна возвращать 1, в то время как все остальные должны возвращать 0. Тогда возврат всех 0 или двух и более 1 интерпретируется как ошибка. Подобная стратегия выбора именуется “only one” в программном каркасе. Как и в случае со стратегией распараллеливания, разработчик графовой модели всегда может добавить собственные стратегии выбора.

Так как API программного каркаса является низкоуровневым средством создания графовых моделей, был разработан формат описания графовых моделей *adot*, расширяющий формат описания графов *dot* [27] с помощью дополнительных атрибутов и определений, описывающих функции-предикаты, функции-обработчики и функции перехода в целом. Пример файла с описанием графовой модели, изображенной на рис. 4, приведен в листинге 1. Детальное описание формата *adot* остается за рамками данной статьи. Для автоматизации процесса создания графовых моделей из *adot*-файлов API программного каркаса содержит в себе метод для преобразования *adot*-файла в объект графовой модели. Подобный способ описания графовых моделей позволяет абстрагироваться от конкретного языка програмирования, на котором реализован API программного каркаса, и описывать графовую модель в ее собственных понятиях (состояния, функции-предикаты и функции-обработчики). Это делает возможным реализацию альтернативных API, в том числе, и на других языках программирования без необходимости переделывать уже построенные графовые модели.

Рассмотрим алгоритм обхода графа, изображенного на рис. 4 и описанного в листинге 1. Пусть D – входные данные графа. Начальное состо-

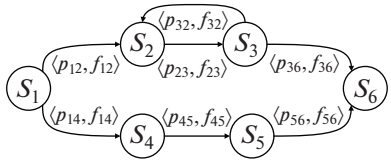

Рис. 4. Графовая модель с параллельными ветвями выполнения и циклом. Следует отметить, что для корректной работы цикла $S_2 \leftrightarrow S_3$ функции-предикаты p_{32} и p_{36} должны быть взаимоисключающими.

яние S_1 имеет два выходных ребра, и, как видно из листинга 1, многопоточную стратегию распараллеливания. Это означает, что при $p_{12}(D) = p_{14}(D) = 1$ будет создан один дополнительный поток, в котором будет выполняться ребро, ведущее к состоянию S_4, в то время как в изначальном потоке будет выполняться ребро, ведущее к состоянию S_2 (здесь предполагается, что выбор ребра, который будет выполняться во вновь созданном потоке, случаен). Затем потоки будут выполняться параллельно вплоть до состояния S_6, где происходит объединение двух потоков. Особое внимание стоит обратить на состояние S_3, которое имеет два выходных ребра, но при этом не образует распараллеливания, что представлено в листинге 1 для этого состояния определенной стратегией выбора “only one”. Это состояние, таким образом, задает условное ветвление, образующее цикл $S_2 \leftrightarrow S_3$ при условии того, что функции-предикаты p_{32} и p_{36} являются взаимоисключающими. Только после завершения работы этого цикла выполнится ветка, ведущая к узлу S_6.

Листинг 1: Пример описания графовой модели с помощью adot-формата

```
1 digraph EXAMPLE {
2     S1 [parallelization_policy=multithreding]
3     S3 [selection_policy=only_one]
4     p_12 [modulename=predicates, entry_func=predicate_12]
5     p_23 [modulename=predicates, entry_func=predicate_23]
6     p_32 [modulename=predicates, entry_func=predicate_32]
7     p_36 [modulename=predicates, entry_func=predicate_36]
8     p_14 [modulename=predicates, entry_func=predicate_14]
9     p_45 [modulename=predicates, entry_func=predicate_45]
10    p_56 [modulename=predicates, entry_func=predicate_56]
11    f_12 [modulename=funcs, entry_func=func_12]
12    f_23 [modulename=funcs, entry_func=func_23]
13    f_32 [modulename=funcs, entry_func=func_32]
14    f_36 [modulename=funcs, entry_func=func_36]
15    f_14 [modulename=funcs, entry_func=func_14]
16    f_45 [modulename=funcs, entry_func=func_45]
17    f_56 [modulename=funcs, entry_func=func_56]
18    __BEGIN__ -> S1
19    S1 -> S2 [edge=(p_12, f_12)]
20    S2 -> S3 [edge=(p_23, f_23)]
21    S3 -> S2 [edge=(p_32, f_32)]
22    S3 -> S6 [edge=(p_36, f_36)]
23    S6 -> __END__
24    S1 -> S4 [edge=(p_14, f_14)]
25    S4 -> S5 [edge=(p_45, f_45)]
26    S5 -> S6 [edge=(p_56, f_56)]
27 }
```

5. ПРИМЕНЕНИЕ ГРАФООРИЕНТИРОВАННОГО КАРКАСА ПРИ СОЗДАНИИ ВЫЧИСЛИТЕЛЬНЫХ МЕТОДОВ МИКРОМЕХАНИКИ КОМПОЗИЦИОННЫХ МАТЕРИАЛОВ

С развитием промышленных технологий и методов проектирования при изготовлении конструкций все активнее применяются гетерогенные материалы, в частности, композиционные (КМ), пористые, градиентные материалы и про-

чие. В современных условиях использование особых свойств гетерогенных материалов на практике фактически невозможно без применения автоматизированных систем при их проектировании. В процессе проектирования предполагается, что новый материал будет обладать заранее заданными свойствами, что сопряжено с проведением большого числа испытаний (физических и/или вычислительных). Количество параметров, оказывающих сильное влияние на свойства КМ, существенно, что связано с: выраженной анизотропией характеристик многих КМ, сильной зависимостью свойств КМ от типов наполнителей и их концентрации, применяемой технологией изготовления и прочими факторами. Физический эксперимент сопряжен с существенными материальными затратами, таким образом применяются методы математического моделирования для автоматизированного прогнозирования характеристик КМ (например, [28] и многие др.), обеспечивающие численное решение задач микромеханики КМ.

К задачам микромеханики КМ относятся: а) задача прямой гомогенизации (оценка эффективных свойств КМ на основе свойств отдельных компонент) [29]; б) задача обратной гомогенизации (оценка характеристик отдельных компонент по известным эффективным свойствам КМ) [30]; в) задача проектирования КМ [31, 32] с заранее задаваемыми свойствами.

Известным и достаточно универсальным численным методом решения задачи прямой гомогенизации является метод асимптотического осреднения (МАО), разработанный Бахваловым Н.С. [33] на основе теории возмущений [34] и примененный Победрей Б.Е. при исследовании упругих, вязкоупругих и упругопластических характеристик композиционных материалов [35]. В случае трехмерной постановки задачи применение МАО предполагает многократный запуск процедур метода конечных элементов (МКЭ) [36].

Представленный в работе графоориентированный каркас эффективно может быть применен для построения программных реализаций указанных методов, как зависящих друг от друга: метода конечных элементов (МКЭ); метода асимптотического осреднения (МАО). На рис. 5 представлена графовая модель программной реализации МКЭ. Модель включает в свой состав три вложенных графовых модели: $[S_1]$, $[S_2]$, $[S_3]$. Модель $[S_1]$ определяет процедуры препроцессинга, $[S_2]$ – обработки, а $[S_3]$ – постпроцессинга. Функции-предикаты обозначены p_α, а функции-обработчики f_α, описания представлены в табл. 1. Содержательный смысл многих функций-обработчиков и функций-предикатов зависит от решаемой задачи: положим для задачи теплопроводности функция f_{1112_2} будет функцией загрузки коэффициентов теплопроводности, тогда как для задачи о напряженно-деформированном состоянии функция f_{1112_2} должна загружать упругие свойства материала. Таким образом в табл. 1 представлены общие описания функций, а в табл. 2 – общие описания состояний данных.

Представленная модель может быть использована при реализации МКЭ для решения задач: а) о напряженно-деформированном состоянии в рамках линейной теории упругости; б) теплопроводности; в) магнито- и электростатики и ряда других при условии фиксированных значений характеристик отдельных элементов конструкции.

Представленный подход активно применялся при решении прикладных задач [37–40], где для хранения разрешающих систем линейных алгебраических уравнений (СЛАУ) использовались различные форматы CSR (для несимметричных матриц) и CSlR (для симметричных матриц) [41, 42], что также было реализовано соответствующей заменой функций перехода.

На рис. 6 представлена графовая модель программной реализации МАО. Описания состояний данных модели представлены в табл. 3.

Решение задачи обратной гомогенизации базируется на применении того или иного метода оптимизации и предположении уже реализованного алгоритма конкретного метода решения задачи прямой гомогенизации (например, МАО).

Наконец, численное решение задачи проектирования КМ с заранее задаваемыми свойствами предполагает использование реализованных алгоритмов решения как прямых так и обратных задач гомогенизации.

В заключение следует отметить, что применение разработанного программного каркаса позволяет обеспечивать гибкую доработку созданных методов с целью возможностей исследования различных физико-механических характеристик КМ, в том числе: тепловых, прочностных, электрических и других.

6. ЗАКЛЮЧЕНИЕ

В данной работе был представлен графоориентированный подход к построению крупномасштабных архитектур программных реализаций СВМ, базирующийся на понятиях теории категорий и введенного понятия графовой модели алгоритма СВМ. С точки зрения алгоритма функционирования графовая модель является расширением понятия конечного автомата, в то время как функции перехода, будучи зависимыми только от данных D, трактуются как чистые функции [43], то есть функции без побочных эффектов, что позволяет наследовать их полезные свойства, обнаруживаемые в функциональной парадигме программирования.

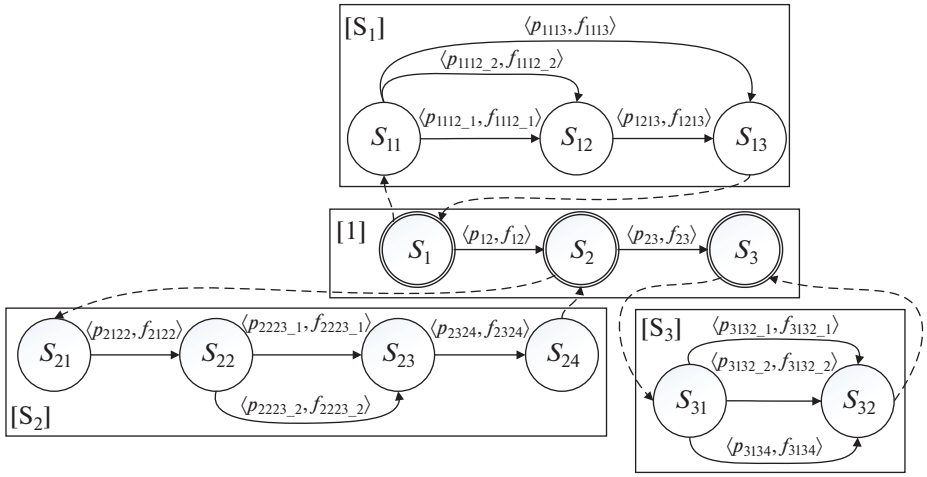

Рис. 5. Графовая модель метода конечных элементов, представленная в виде одной базовой модели и трех вложенных моделей, определяющих процедуры: [S_1] препроцессинга, [S_2] вычисления, [S_3] постпроцессинга.

Для эффективного использования описанной выше графовой модели необходимо придерживаться определенной идеологии при разработке функций перехода. В первую очередь, следует отметить, что функции-обработчики выполняют преобразование входных данных, т.е. выполняют основные вычислительные операции, в то время как функции-предикаты определяют, действительно ли входные данные находятся в нужном состоянии для очередной функции-обработчика, и, таким образом, гарантируют, что на вход функции-обработчика будут поданы корректные данные. Подобная семантика функций-предикатов предполагает, что функции-обработчики и функции-предикаты должны разрабатываться одновременно в рамках одной функции перехода, тогда как разные функции перехода могут создаваться параллельно несколькими независимыми разработчиками, что создает потенциал для параллелизации процесса разработки программной реализации СВМ. Автономность процесса разработки также автоматически дает возможность организо-

Таблица 1. Описания функций-предикатов и функций-обработчиков графовой модели программной реализации МКЭ

Модель	Функция	Описание функций предикатов и обработчиков графовых моделей
[S_1]	f_{1112_1}	Загрузка конечно-элементной аппроксимации расчетной области.
	f_{1112_2}	Загрузка характеристик элементов расчетной области. Типы свойств определяются задачей. Загрузка должна осуществляться после возможной загрузки геометрии расчетной области.
	f_{1113}	Загрузка краевых условий. Типы условий определяются задачей.
	f_{1213}	Формирование объекта конечно-элементная задача, на основе загруженных данных и сохранение объекта.
[S_2]	f_{2122}	Сборка разрешающей СЛАУ.
	f_{2223_1}	Решения СЛАУ.
	f_{2223_2}	Возможная выгрузка сопутствующей информации о процессе решения СЛАУ.
	f_{2324}	Вычисление производных полученного решения СЛАУ.
[S_3]	f_{3132_1}	Сохранение скалярных результатов расчета.
	f_{3132_2}	Сохранение отдельных результатов расчета в базе данных.
	f_{3134}	Сохранение векторных результатов расчетов в графических форматах.

Таблица 2. Описания состояний данных графовых моделей программной реализации МКЭ

Модель	Состояние данных	Описание состояния данных
$[S_1]$	S_{11}	Файл постановки задачи (формат на основе aINI[?]) загружен в объект общих данных.
	S_{12}	Загружена конечно-элементная аппроксимация расчетной области и свойства материалов каждого элемента конструкции.
	S_{13}	Выбран и сформирован объект – тип используемого конечного элемента. Загружены краевые условия, определенные для поверхностей расчетной области. Процедура препроцессинга МКЭ завершена.
$[S_2]$	S_{21}	Начало вычислительной процедуры МКЭ.
	S_{22}	Загружен решатель СЛАУ. Процедура сборки разреженной разрешающей СЛАУ завершена.
	S_{23}	Разрешающая СЛАУ решена. Дополнительно могли быть проведены процедуры выгрузки сформированной СЛАУ.
	S_{24}	Завершены процедуры вычисления производных полученного результата решения разрешающей СЛАУ (например, вычисление компонент тензора деформации, напряжений, тепловых потоков, градиента температуры и т.д.). Конкретный тип вычисляемых производных параметров зависит от типа решаемой задачи и модифицируется применяемыми функциями перехода.
$[S_1]$	S_{31}	Начало процедуры постпроцессинга МКЭ.
	S_{32}	Завершена процедура постпроцессинга МКЭ: сформированы файлы скалярных и векторных результатов. Возможно проведены: выгрузка статистических результатов в базу данных, сохранение трехмерной модели расчетной области и сопутствующих ей полей полученных отдельных компонент векторных результатов.

вать модульное тестирование и документирование разрабатываемого кода, так как функции-обработчики естественным образом формируют модули.

Важной проблемой при поддержке программных реализаций СВМ является описание и объяснение структуры программного продукта для нового разработчика или любого другого заинтересованного лица. Одним из достоинств графового представления процесса вычислений является наглядная визуализация процесса обработки данных. Более того, используя предложенный подход, появляется возможность хранения служебной информации в данных СВМ (метаданных),

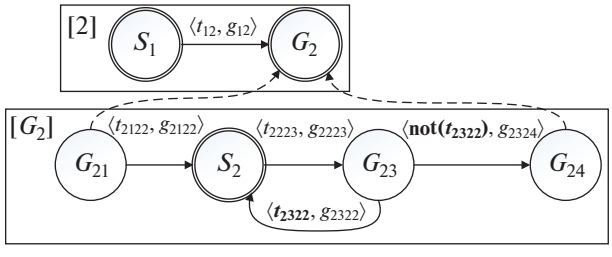

Рис. 6. Графовая модель метода асимптотического осреднения, основанная на графовых моделях МКЭ.

что теоретически позволяет собирать статистическую информацию о процессе обработки данных, которая может дать разработчикам программной реализации СВМ обратную связь о работе функций-предикатов и функций-обработчиков. Кроме того, используя метаданные, графовая модель может естественным образом визуализировать процесс ее выполнения в режиме реального времени. Работа над реализацией подобной автоматической визуализации ведется в настоящий момент с применением веб-технологий в рамках подпроекта *comwpc* “Распределенной вычислительной системы GCD”.

Следует отметить, что графовая модель задает лишь высокоуровневую архитектуру программной реализации СВМ и не претендует на описание простых алгоритмов, таких как, например, обход элементов вектора в цикле. Аналогично, графовая модель не является конкурентом классическим инструментам распараллеливания кода в научных приложениях, таким как OpenMP, MPI, CUDA и прочим, так как параллелизация, реализуемая в графовой модели, является крупнозернистой [44], то есть параллелизацией на уровне задач. Наконец, графовая модель не накладывает ограничений на парадигму програм-

Таблица 3. Описания состояний данных графовой модели программной реализации МАО

Модель	Состояние данных	Описание состояния данных
$[G_2]$	G_{21}	Начало вычислительной процедуры МАО. Предполагается, что процедура препроцессинга МАО завершилась успешно.
	S_2	Инициализированы краевые условия конкретной локальной задачи МАО и поданы на вход вложенной графовой модели вычислительной процедуры МКЭ.
	G_{23}	Сохранены результаты решения локальной задачи МАО в объекте общих данных.
	G_{24}	Проведен постпроцессинг процедуры МАО, включая расчет эффективных характеристик исследуемого КМ.
[2]	S_1	Препроцессинг продедуры МКЭ с точностью до замены отдельных функций перехода, в частности: загружается массив стандартных краевых условий локальных задач МАО.
	G_2	Проведение вычислительной процедуры МАО с включенными процедурами постпроцессинга МАО.

мирования, используемую при написании функций-предикатов и функций-обработчиков.

Будучи обобщенным описанием процесса вычислений, графовая модель может использоваться для описания вычислительных кампаний, то есть комплексных вычислений, состоящих из решения серии сложных вычислительных задач, зависящих друг от друга по тому или иному логическому правилу. Подобная функциональность становится особенно востребованной, когда препроцессинг и постпроцессинг проводятся локально, в то время как ресурсоемкие вычисления выполняются на удаленном суперкомпьютере, использующим планировщик задач. Одним из перспективных направлений развития графоориентированного подхода и программного каркаса *comsdk* является поддержка создания соответствующих графовых моделей и работы с Oracle Grid Engine.

Наконец, очевидно, что при разработке графовых моделей некоторые функции-обработчики, функции-предикаты и целые подграфы будут повторяться вновь и вновь, что и было продемонстрировано представленным примером. Для избежания их повторной разработки и тестирования планируется создание репозитория, где будут храниться наиболее хорошо документированные и протестированные функции перехода и графовые модели, доступ к которым будет организован приложением, схожим по принципу функционирования с пакетными менеджерами. Подобная разработка также предполагает реализацию инструментов встроенного самодокументирования функций-предикатов и функций-обработчиков, основанных на динамическом анализе их входных и выходных параметров.

В результате проведенных разработок подана заявка на получение патента на изобретение № 2017122058 от 22.06.2017. Разработка графоориентированного подхода и программного каркаса *comsdk* выполнена авторами на инициативной основе при разработке ядра “Распределенной вычислительной системы GCD”.

СПИСОК ЛИТЕРАТУРЫ

1. How do scientists develop and use scientific software? / J.E. Hannay, C. MacLeod, J. Singer et al. // Proceedings of the 2009 ICSE workshop on Software Engineering for Computational Science and Engineering / IEEE Computer Society. 2009. P. 1–8.
2. *Killcoyne S., Boyle J.* Managing Chaos: Bridging the cultural divide between engineers and scientists working within the life sciences // Computing in Science Engineering. 2018.
3. Troubling trends in scientific software use / L.N. Joppa, G. McInerny, R. Harper et al. // Science. 2013. V. 340. № 6134. P. 814–815.
4. Numerical recipes 3rd edition: The art of scientific computing / W.H. Press, S.A. Teukolsky, W.T. Vetterling et al. Cambridge university press, 2007.
5. *Roy C.J., Oberkampf W.L.* A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing // Computer methods in applied mechanics and engineering. 2011. V. 200. № 25–28. P. 2131–2144.
6. *Алексеев А.К., Бондарев А.Е.* Применение сопряженных уравнений в задачах верификации и валидации расчетов // Новые информационные технологии в автоматизированных системах. 2012. № 15. С. 104–112.
7. ASC predictive science academic alliance program verification and validation whitepaper: Tech. Rep.: / *R.*

Klein, S. Doebling, F. Graziani et al.: Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2006.

8. *Resio D.T., Westerink J.J.* Modeling the physics of storm surges // Physics Today. 2008. № 9. P. 33–38.
9. Best practices for scientific computing / G.Wilson, D.A. Aruliah, C.T. Brown et al. // PLoS biology. 2014. V. 12. № 1. P. e1001745.
10. *Parker S.G., Johnson C.R.* SCIRun: a scientific programming environment for computational steering // Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference / IEEE. 1995.
11. Toward a common component architecture for high-performance scientific computing / R. Armstrong, D. Gannon, A. Geist et al. // High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on / IEEE. 1999. P. 115–124.
12. SCIRun2: A CCA framework for high performance computing / K. Zhang, K. Damevski, V. Venkatachalapathy et al. // High-Level Parallel Programming Models and Supportive Environments, 2004. Proceedings. Ninth International Workshop / IEEE. 2004. P. 72–79.
13. *Mortensen Mikael, Langtangen Hans Petter, Wells Garth N.* A FEniCS-based programming framework for modeling turbulent flow by the Reynolds-averaged Navier-Stokes equations // Advances inWater Resources. 2011. V. 34. № 9. P. 1082–1101. FEniCS (software).
14. *Logg A., Mardal K.-A., Wells G.* Automated solution of differential equations by the finite element method: The FEniCS book. Springer Science & Business Media, 2012. V. 84.
15. OpenFOAM: A C++ library for complex physics simulations / H. Jasak, A. Jemcov, Z. Tukovic et al. // International workshop on coupled methods in numerical dynamics / IUC Dubrovnik, Croatia. V. 1000. 2007. P. 1–20.
16. *Schoberl J.* An advancing front 2D/3D-mesh generator based on abstract rules // Computing and Visualization in Science. 1997. V. 1. № 1. P. 41–52.
17. *Bonchi F., Sobocinski P., Zanasi F.* Full abstraction for signal flow graphs // ACM SIGPLAN Notices. 2015. V. 50. № 1. P. 515–526.
18. *Baez J.C., Erbele J.* Categories in control // Theory and Applications of Categories. 2015. V. 30. № 24. P. 836–881.
19. Baez J.C., Coya B., Rebro F. Props in network theory // arXiv preprint arXiv:1707.08321. 2017.
20. *Pavlovic D.* Monoidal computer I: Basic computability by string diagrams // Information and computation. 2013. V. 226. P. 94–116.
21. *Awodey S.* Category theory. Oxford University Press, 2010.
22. *Allen F.E.* Control flow analysis // ACM Sigplan Notices. 1970. V. 5. № 7. P. 1–19.
23. *Johnston W.M., Hanna J., Millar R.J.* Advances in dataflow programming languages // ACM computing surveys (CSUR). 2004. V. 36. № 1. P. 1–34.
24. *Morrison J.P.* Flow-Based Programming: A new approach to application development. CreateSpace, 2010.
25. *Соколов А.П., Шпакова Ю.В., Першин А.Ю.* Проектирование распределенной программной системы GCD численного моделирования композитов // Сборник трудов XXV Международной научной конференции “Математические методы в технике и технологиях” (ММТТ–25) в 10 т. Т. 5. Волгоград: 2012. С. 79–80.
26. *Соколов А.П., Першин А.Ю.* Программный инструментарий для для создания подсистем ввода данных при разработке систем инженерного анализа // Программная инженерия. 2017. Т. 8. № 12. С. 543–555.
27. The DOT Language – Graphviz. https://www.graphviz.org/doc/info/lang.html.
28. *Wang Q., Zhang R., Wang J. et al.* An efficient method for geometric modeling of 3D braided composites // Journal of Engineered Fibers and Fabrics. 2016. V. 11. № 14. P. 76–87.
29. *Buryachenko V.A.* Micromechanics of Heterogeneous Materials. New York: Springer, 2007. 687 p.
30. *Sigmund Ole*. Materials with prescribed constitutive parameters: An inverse homogenization problem // International Journal of Solids and Structures. 1994. V. 31. № 17. P. 2313–2329.
31. *Бобрышев А.Н., Жарин Д.Е., Шафигуллин Л.Н., Гумеров М.И.* Система автоматизированного проектирования полимерных наполненных композиционных материалов специального назначения // Кузнечно-штамповочное производство. Обработка материалов давлением. 2009. № 8. С. 9–16.
32. *Киселев А.М.* Определение перспективных направлений в построении автоматизированных систем проектирования 3D-преформ и прогнозирования заданных свойств композиционных материалов на их основе // Известия высших учебных заведений. Технология текстильной промышленности. 2009. Т. 356. № 2. С. 141–145.
33. *Бахвалов Н.С., Панасенко Г.П.* Осреднение процессов в периодических средах. Москва: Наука, 1984. 352 с.
34. *Коул Дж.* Методы возмущений в прикладной математике. Москва: Издательство “Мир”, 1972. С. 276.
35. *Победря Б.Е.* Механика композиционных материалов. М.: МГУ, 1984. 336 с.
36. Hutton David V. Fundamentals of Finite Element Analysis. Pullman (WA, USA): McGraw-Hill Science/Engineering/Math, 2004. C. 494.
37. *Dimitrienko Yu.I., Sborshchikov S.V., Sokolov A.P.* Numerical simulation of microdestruction and strength characteristics of spatially reinforced composites // Composites: Mechanics, Computations, Applications. 2013. V. 4. № 4. P. 345–364.
38. *Димитриенко Ю.И., Сборщиков С.В., Соколов А.П., Гафаров Б.Р., Садовничий Д.Н.* Численное и экспериментальное моделирование прочностных характеристик сферопластиков // Композиты и наноконструкторы. 2013. Т. 19. № 3. С. 35–51.
39. *Соколов А.П., Щетинин В.Н., Сапелкин А.С.* Параллельный алгоритм построения поверхности прочности КМ для архитектуры Intel Many Integrated Core Architecture // Программные системы: теория и приложения. 2016. Т. 7. № 2. С. 3–25.

40. *Соколов А.П., Першин А.Ю., Щетинин В.Н., Сапелкин А.С.* Реверсивная многомасштабная гомогенизация физико-механических характеристик гетерогенных периодических сред с использованием графоориентированного программного подхода // Композиты и наноконструкторы. 2017. Т. 9. № 3–4. С. 25–38.

41. *Баландин М.Ю., Шурина Э.П.* Методы решения СЛАУ большой размерности. Новосибирск: Изд-во НГТУ, 2000. С. 70.

42. *Saad Y.* Iterative methods for sparse linear systems. New York (USA): Yousef Saad, 2000. 3. 460.

43. *Peyton Jones and Simon L. Haskell* 98 Language and Libraries: The Revised Report (PDF). Cambridge, United Kingdom: Cambridge University Press, 2003. 95 p.

44. *Баденко В.Л.* Высокопроизводительные вычисления. Учебное пособие. СПб.: Изд-во Политехн. ун-та, 2010. 180 с.