S. S. S. Nasser, Yu. T. Lyachek, M. C. A. Muthanna
Saint Petersburg Electrotechnical University
M. M. A. Muthanna
Saint Petersburg University «SPBU»
A. Khakimov
Saint Petersburg State University of Telecommunications

ENERGY-AWARE ALGORITHM FOR LORA TECHNOLOGY: PROTOTYPE IMPLEMENTATION

The internet of Things (IoT) is an umbrella term describing large interconnected systems including sensor networks, typically exploiting wireless links and supporting fixed and mobile communications. LoRa (LongRange) is a popular Low Power Wide Area Network (LPWAN) standard designed for the IoT, because of its satisfactory performance and relatively low cost. The MAC layer protocol of LoRa is LoRaWAN (LongRange Wide Area Network) is one of them. LoRaWAN is a technical solution for physical and partially connected network layers. The article proposes an algorithm to ensure the quality of service traffic, latency and data loss, as well as to provide an efficient way to consume energy. We are implementing a prototype for edge computing based on LoRa. Numerical results show the proposed architecture will play a key role in building private LoRa networks.

LoRa, latency, energy, prototype, edge computing

УДК 004.942

А. П. Соколов, А. Ю. Першин
Московский государственный технический университет им. Н. Э. Баумана

Система автоматизированного проектирования композиционных материалов. Часть 2. Вычислительная подсистема, распределенные вычисления с применением графоориентированного подхода*

Описывается вычислительная подсистема, включая ее архитектуру, вошедшая в состав прототипа системы автоматизированного проектирования композиционных материалов (САПР КМ). Проведен обзор литературы и представлены известные программные подходы к созданию наукоемкого программного обеспечения, включая методики реализации вычислительных методов. Обоснована важность применения специальных подходов при разработке сложного наукоемкого программного обеспечения. Сделан акцент на методах разработки расширяемых и сопровождаемых вычислительных библиотек, реализующих сложные вычислительные методы (СВМ). Представлены особенности, отличающие разработку программной реализации СВМ от программной реализации вычислительного алгоритма решения конкретной узкоспециализированной задачи. Приведен перечень требований, предъявленных к вычислительной подсистеме САПР КМ при ее проектировании, а также принципы, которые легли в основу ее создания. Введено понятие «решатель» и представлены его взаимоотношения с понятиями «функциональный компонент», «вычислительная подсистема», «СВМ» и прочими понятиями системы, введенными в первой части статьи. Детально представлена процедура решения вычислительной задачи, в основе программной реализации которой лежит графовая модель соответствующего СВМ. Приведен состав разработанной вычислительной подсистемы. Представлены принципы разработки библиотек вычислительных функций с использованием графоориентированного подхода. Введено понятие «функция-селектор», а также представлено описание стратегий распараллеливания и ветвления, предполагающих использование таких функций. Представлен пример применения графоориентированного подхода при решении прикладной задачи анализа турбулентного течения вязкой несжимаемой жидкости.

Технологии разработки инженерного программного обеспечения, графоориентированный подход, разработка вычислительных библиотек, автоматизированное проектирование композиционных материалов, система инженерного анализа, разработка модулей расширения, технологии распределенных вычислений, потоко-ориентированные системы, e-science tools

Технологиям создания программного обеспечения (ПО) в целом посвящено огромное множество материалов, в том числе книги и научные публикации. Одновременно с этим оказывается довольно сложным найти материалы (согласно базам данных РИНЦ и Scopus) по технологиям разработки специ-

* Продолжение. Начало см. в Изв. СПбГЭТУ «ЛЭТИ». 2020. № 8–9.

ализированного наукоемкого ПО (в том числе коммерческого ПО), к которому можно отнести: различное инженерное ПО, CAD- и CAE-системы, а также ПО математического моделирования.

Разработка наукоемкого ПО отличается, помимо прочего, существенной сложностью реализуемых алгоритмов и необходимостью реализации сложных вычислительных методов. При создании такого ПО, в особенности коммерческого наукоемкого ПО, проблемы отладки встают все более остро: количество частей исходного кода программы, потенциально содержащих ошибки, значительно возрастает, что ведет к существенно бóльшим трудозатратам на отладку и сопровождение.

Привлечение бóльшего числа разработчиков не решает проблему, а еще больше усугубляет, так как приводит к проблеме согласованности внесения изменений в исходный код общей системы. Частичное решение этой проблемы предоставляют системы контроля версий (Subversion, GIT и др.), но и они не позволяют обеспечить логическую согласованность вносимых изменений. Каждый разработчик не может заранее знать всю архитектуру большой системы, но должен в ней вести работы, что небезопасно для целостности исходного кода – возможно нарушение принципов построения.

Для ПО рассматриваемого класса характерна увеличивающаяся ресурсоемкость, что требует наличия многопроцессорных высокопроизводительных компьютеров (*англ*. High-Performance Computers (HPC)). Такие требования могут вызвать необходимость переработки ранее написанного кода, включая разработку параллельных версий уже созданных последовательных программ.

Широкий спектр приложений, потенциальная ресурсоемкость актуальных прикладных вычислительных задач, серьезные требования к разработчикам определяют принципиальную сложность разработки программных средств из рассматриваемого класса и необходимость использования специализированных технологий разработки. О важности создания специализированных технологий разработки сложного ПО, и в том числе ПО для математического моделирования, к примеру, говорится в [1].

Известным подходом, в том числе применяемым для разработки ПО указанного класса, является организация комплексных вычислений или описание алгоритма решения вычислительной задачи в форме ориентированного графа или «графовой модели» алгоритма (содержащего или не содержащего циклы). Далее с использованием интерпретатора осуществляется обход графовой модели.

Системы, реализующие указанный подход, носят название «систем управления рабочими процессами» (*англ*. Workflow management systems) [2]. Для описания рабочего процесса могут применяться специальные языки, например Common Workflow Language (CWL) [3].

Как правило системы из указанного класса применяются далеко не только для организации вычислений, а являются системами решения задач организации процессов обработки данных и используются в различных прикладных областях (например, описание бизнес-процессов, разработка систем реального времени и др.). Далее в статье указанный подход рассматривается только в контексте разработки вычислительных подсистем, которые в зарубежной литературе называют e-science workflow systems.

В [2], [4], [5] представлены подробные обзоры существующих известных программных систем (Taverna [6], Pegasus [7], Kepler, Triana, Vistrails и др.), применяемых для организации комплексных вычислений при решении сложных наукоемких задач. Следует отметить [8], где авторы детализировали базовые концепции таких систем и их примерный состав, а также привели особенности существующих реализованных систем.

Информации об отечественных системах рассматриваемого класса среди доступной в научных публикациях отечественных журналов очень мало. Среди обзорных публикаций можно отметить [2], однако и в ней представлены все те же зарубежные системы.

Одним из оригинальных подходов к созданию вычислительных библиотек является подход проф. В. Э. Малышкина [9]. Ключевой особенностью подхода является возможность автоматического распараллеливания программной реализации вычислительного метода, построенного с использованием специализированного языка и системы LuNA.

Развиваемым в настоящий момент является программный инструментарий Templet [10], включающий в свой состав: а) библиотеки параллельного программирования Templet SDK; 2) вебсервис для запуска и мониторинга задач на супер-

компьютере Templet Web; 3) подсистему мониторинга состояния кластера. Основным назначением инструментария является автоматизация параллельных вычислений на удаленном суперкомпьютере. Среди возможностей также следует отметить автоматическое распараллеливание вычислительной программы за счет использования DSL-языка Templet. Важной особенностью Templet является использование специального препроцессора исходных кодов программ, подготовленных на основе текстового шаблона на том или ином языке программирования (C++, Pascal и др.). Препроцессор позволяет разработать и отладить последовательную программу на локальной машине, а затем запустить код на исполнение на суперкомпьютере в параллельном режиме (параллельный код будет получен автоматически из последовательной версии, для параллельной работы будет использоваться API POSIX Threads).

Основной технологией, реализованной и примененной при разработке описываемой в настоящей статье вычислительной подсистемы, стал графоориентированный подход или графоориентированная программная инженерия (ГПИ) [11]. Особенностями подхода являются существенные различия в назначении узлов и ребер, а также использование ориентированных графов общего вида, допускающих наличие циклов, что, как правило, является ограничением многих известных систем, использующих ориентированные ациклические графы [4].

Применение подхода позволило систематизировать процесс разработки программных реализаций сложных вычислительных методов и обеспечило возможность формирования вычислительных библиотек группой разработчиков независимо. В [11], [12] представлен подробный обзор технологий, также предназначенных для упрощения разработки наукоемкого ПО, в том числе похожих на применяемый графоориентированный подход, среди которых специальные разработки компании Siemens, MatLab, LabView, FEniCS и др.

Фактическое отсутствие в открытой печати информации об отечественных системах рассматриваемого класса обусловлено, предположительно, закрытостью большинства из них и применением, как правило, в коммерческих целях (отметим, например, систему pSeven).

По мнению авторов, необходимым условием, определяющим принципиальную возможность создания ПО рассматриваемого типа, является участие специалистов, обладающих одновременно компетенциями как в области математического моделирования, так и в области программной инженерии. До 2006 г. специалистов по программной инженерии высшие учебные заведения России не готовили вовсе [13]. По рассматриваемому же профилю в России специалистов не готовят и сейчас даже в ведущих технических вузах страны. Как правило, специалисты в области математического моделирования не обладают достаточными навыками в области программной инженерии и наоборот. Текущее положение дел обусловлено, с одной стороны, существенным объемом материала и ограниченностью срока подготовки, а с другой – ориентированностью программ подготовки на решение учебных задач, отсутствием практического опыта у преподавателей, инертностью восприятия новых технологий и знаний в академической и производственной средах, потерей цепочек производственных связей между вузами и промышленностью за прошедшие 2–3 десятилетия и пр.

Помимо специализированных технологий принципиально важным при создании сложных программных комплексов является непосредственное участие руководителя проекта в отборе и подготовке кадров. Руководитель должен быть признанным лидером коллектива, что определяется его профессиональными компетенциями [14]. В указанных условиях такие специалисты стали редкостью.

Все перечисленные проблемы определили очевидное отставание и неконкурентоспособность России в рассматриваемой отрасли в сравнении с существующими в мире программными решениями.

Подтверждением этим выводам может служить фактическое отсутствие публикаций по указанному направлению в РИНЦ, тогда как там же легко могут быть найдены многочисленные публикации о практике применения низкоуровневых программных средств разработки параллельных программ (OpenMP, MPI, CUDA, OpenCL и пр.), включая методы программирования для Grid-систем [15], которые активно используются при научном программировании. Применение указан-

ных технологий возможно отдельными исследователями в рамках решения конкретных задач, но затруднительно при решении прикладных задач широкого класса, что характерно для задач промышленности и обычно требует использования многофункционального ПО и предметно-ориентированных вычислительных библиотек.

Настоящая статья является второй в цикле из трех статей о системе автоматизированного проектирования композиционных материалов (САПР КМ), целью которой является представить технологии, примененные при разработке *вычислительной подсистемы САПР КМ*. Основы всей системы были представлены в ч. 1 цикла.

Далее приведены требования, которым должна удовлетворять вычислительная подсистема, чтобы ее прикладное использование было возможным.

Требования к вычислительной подсистеме. Очевидно, что в состав вычислительной подсистемы должны входить программные средства, обеспечивающие решение различных классов вычислительных задач, среди которых можно выделить: инженерные задачи проектных расчетов, задачи научных исследований, образовательные вычислительные задачи и пр.

Замечание 1. Программное обеспечение, предназначенное для решения вычислительных задач из одного класса, допуская определенную нестрогость, будем называть *решателем*. Решатель является программной реализацией некоторого вычислительного метода или, в более общем случае, сложного вычислительного метода.

Определение 1. Вычислительный метод, предполагающий использование одного или нескольких других вычислительных методов, назовем сложным (далее *сложный вычислительный метод* (СВМ)).

Используя введенную терминологию, вычислительная подсистема, помимо прочего, должна объединять в своем составе множество программных реализаций сложных вычислительных методов, обеспечивающих решение вычислительных задач рассматриваемых классов, или проще – множество решателей.

Требования к вычислительной подсистеме в целом индуцируются требованиями к ее составным частям, т. е. к решателям. В свою очередь требования к решателям определяются особенностями классов вычислительных задач, для решения которых они предназначены. Напомним, что одному решателю соответствует один класс вычислительных задач, каждый из которых может носить прикладной, исследовательский или образовательный характер. Среди важных отличий класса вычислительных задач от частной вычислительной задачи, входящей в этот класс, можно выделить:

– существенно бóльший массив входных данных;

– частые уточнения постановки задачи, приводящие к изменениям перечня параметров, определяющих входные данные, включая как количество параметров, так и их типы;

– отдельные параметры входных данных могут быть неизвестны вовсе или известны неточно, информация о факте их неточности может стать известной в процессе решения;

– как правило, класс вычислительных задач может включать задачи с существенной вычислительной сложностью.

Отметим важные особенности, отличающие разработку программной реализации СВМ (решателя) от программной реализации вычислительного алгоритма решения узкоспециализированной задачи:

– существенный объем работ по написанию исходных кодов, ограничивающий возможности создания качественной реализации одним исследователем;

– существенный объем входных данных (скалярных и/или векторных), в том числе задаваемых неточно;

– существенное количество особых условий функционирования и согласованности выбранных численных методов выбранным алгоритмам и моделям;

– требование обеспечения масштабируемости (возможность задействовать доступные вычислительные ресурсы многопроцессорной техники при большой вычислительной сложности задачи);

– требование возможности запуска на различных платформах (так как решатель должен обеспечивать решение всех задач из соответствующего класса, то вполне возможно, что для текущей задачи необходим запуск решателя на одной платформе, а для следующей задачи – на другой, чтобы использовать соответствующие ресурсы этих платформ);

– необходимость обеспечения единого формата данных для постановки задачи;

– существенные трудозатраты на сопровождение.

В результате анализа можно сформулировать требования, которым должна удовлетворять *вычислительная подсистема (ВП)*:

1. ВП и ее архитектура не должны зависеть от требований к постановкам вычислительных задач, решаемых с ее помощью.

2. В состав ВП должны входить такие вспомогательные программные средства, которые бы позволили обеспечить технологичность (стандартизацию, унификацию, преемственность) процессов проектирования, разработки, сопровождения и внедрения решателей.

3. Создаваемые программные реализации решателей, входящие в состав ВП, должны учитывать указанные особенности классов вычислительных задач, для решения которых они предназначены.

4. С использованием ВП должны быть минимизированы трудозатраты на возможные обновления уже созданных решателей.

5. ВП должна обеспечивать унификацию процессов решения различных инженерных задач с использованием созданных решателей.

6. ВП должна содержать программные средства, упрощающие процессы изучения алгоритмов созданных решателей, в том числе средства построения документации, средства визуализации процессов решения, а также средства визуализации результатов расчетов.

7. ВП должна включать в себя программные средства регистрации проведенных вычислительных экспериментов, их атрибутов и получаемых результатов.

Вычислительная подсистема и ее состав. Несмотря на то, что описываемая в настоящей статье *вычислительная подсистема* разрабатывалась как одна из ключевых подсистем САПР КМ, еще на этапе проектирования было определено требование о более широкой области ее применения. Требование было обусловлено актуальностью и других прикладных задач, решаемых в рамках научного коллектива под руководством доцента Соколова А. П., к примеру: задач оптимизации в технических системах, химической кинетики, анализа данных в различных областях науки и техники и пр. Указанные задачи в том или ином виде требуется решать в рамках САПР различного назначения.

В ч. 1 статьи по основам разработки САПР КМ были представлены базовые классификации компонентов системы, а также введено понятие *функционального компонента*. В рамках данной терминологии *вычислительная подсистема является объединением таких компонентов системы, и в том числе*

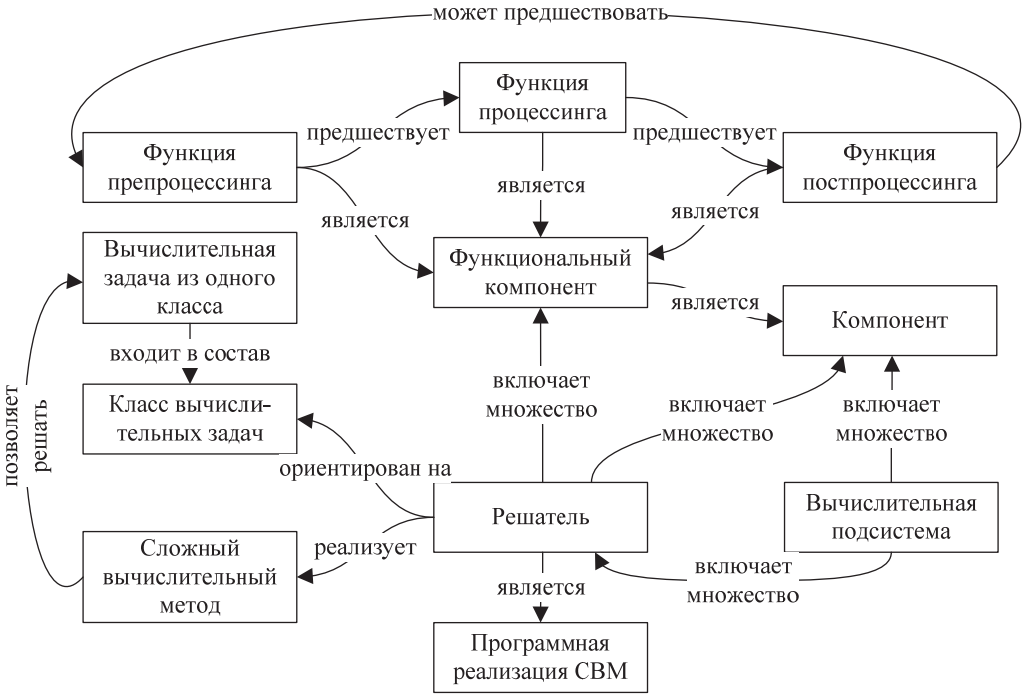

Рис. 1

функциональных компонентов, которые в совокупности реализуют сложные вычислительные методы (см. определение 1), обеспечивающие решение различных классов вычислительных задач. Схема отношений базовых понятий системы и понятий вычислительной подсистемы наглядно поясняет изложенное (рис. 1).

Приведем примеры некоторых классов вычислительных задач: а) задачи анализа эффективных упругих характеристик КМ; б) задачи идентификации упругопрочностных характеристик компонентов КМ; в) прямые задачи химической кинетики; г) обратные задачи химической кинетики и пр.

Замечание 2. Отметим, что многие методы решения прикладных или исследовательских вычислительных задач обычно относятся к классу СВМ. Например, задача автоматизированного проектирования КМ предполагает необходимость решения как прямой, так и обратной задач микромеханики КМ. Для решения прямой задачи часто используется тот или иной метод гомогенизации (МГ), а он, в свою очередь, может предполагать использование метода конечных элементов (МКЭ) и т. д. Обратные задачи микромеханики КМ предполагают необходимость использования методов глобальной оптимизации, а они, в свою очередь, могут потребовать использования МГ, МКЭ и пр.

Для построения вычислительной подсистемы в рассматриваемом универсальном смысле были введены специальные типы ее *функциональных компонентов* (в рамках терминологии, введенной в ч. 1 статьи) и были созданы стандартные методы разработки компонентов соответствующих типов. Например, в [16] представлен один из таких методов, обеспечивающий автоматическое построение компонентов, каждый из которых должен обладать графическим пользовательским интерфейсом.

Функциональные компоненты, определяющие основу вычислительной подсистемы, были сгруппированы (аналогичные группы используются различными авторами, например, [17]): а) функции по подготовке данных («препроцессинг»); б) функции по обработке данных («процессинг»); в) функции по постобработке («постпроцессинг») данных; г) комплексные функции, включающие множество циклов: «препроцессинга», «процессинга» и «постпроцессинга». Этапность выполнения функций разных типов при решении многих вычислительных инженерных задач, как правило, заранее определена (рис. 2). Указанная классификация в рамках представляемой САПР КМ была реализована добавлением атрибута «*группа типов компонентов по общему назначению*» к типу компонента по предметному назначению (ч. 1).

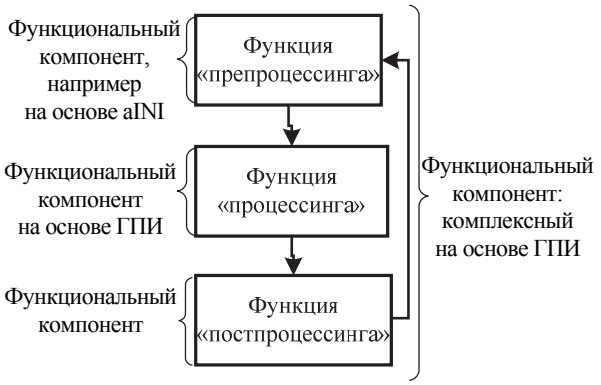

Рис. 2

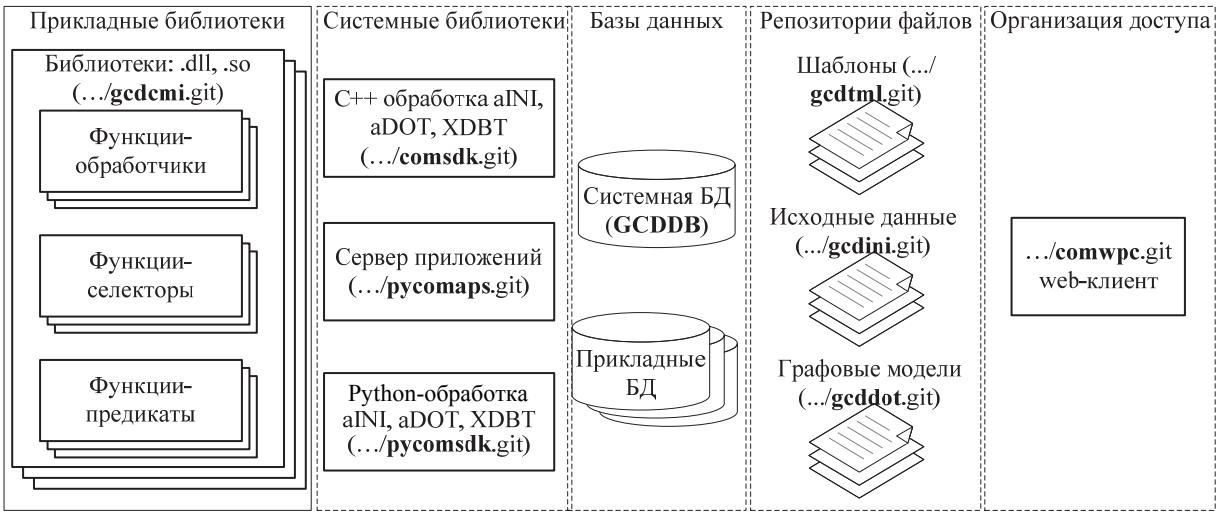

Рис. 3

К задачам препроцессинга можно отнести: а) ввод и определение входных данных (предоставление графического пользовательского интерфейса, запрос параметров в консоли, запрос файла входных данных в некотором формате); б) прием и обработку потока входных данных (например, по сети с использованием заранее определенного протокола передачи данных). Решение обеих задач может быть автоматизировано. В рамках представляемой САПР КМ решение первой задачи было автоматизировано применением специального языка определения входных данных aINI [16], [18], интерпретируя который можно создавать функциональные компоненты соответствующего подтипа.

Функциональные компоненты, обеспечивающие решение задач процессинга в рамках вычислительной подсистемы (см. рис. 1), реализуются с помощью графоориентированного подхода, базовые теоретические основы которого представлены в [11]. Особенности разработки таких компонентов и принципы построения графовых моделей вычислительных методов будут описаны далее.

Функциональные компоненты постпроцессинга реализуются по-разному, к ним можно отнести: функции визуализации данных (графических зависимостей и поверхностей); функции рендеринга графических изображений на основе, к примеру, результатов расчетов, полученных в трехмерном пространстве, и пр. В настоящей статье они не рассматриваются.

Состав вычислительной подсистемы. Помимо непосредственно функциональных компонентов предметного назначения в состав вычислительной подсистемы входят и вспомогательные компоненты: библиотеки, базы данных, программные средства (рис. 3). Все представленные компоненты в совокупности обеспечили работу тех самых *функциональных компонентов предметного назначения, реализующих СВМ для решения конкретных прикладных вычислительных задач*.

Неполный перечень прикладных библиотек вычислительной подсистемы с описанием представлен в таблице. Разработка каждой библиотеки ведется в одноименном git-репозитории.

Системная база данных GCDDB в контексте вычислительной подсистемы используется для хранения настроек подсистемы, а также идентификационных данных следующих объектов и их

№	Библиотека	Описание
1	gcdcmi	Библиотека функций-обработчиков, функций-селекторов и функций-предикатов широкого спектра приложений
2	gcdfes	Библиотека базовых функций конечно-элементных методов решения задач механики сплошных сред
3	gcdhom	Библиотека функций, обеспечивающих работоспособность методов гомогенизации в рамках задач механики композиционных материалов
4	gcdopt	Библиотека функций, обеспечивающих работоспособность методов оптимизации
5	gcddot	Библиотека графовых моделей решателей задач
6	gcdini	Библиотека входных данных (постановок) задач
7	hom3d	Библиотека классов для решения задач прямой и обратной трехмерной гомогенизации композитов
8	gcdnrg	Программная библиотека функций по автоматизации в области электроэнергетики для оптимизации работы ТЭЦ, ТЭС, ГЭС, ГРЭС, АЭС
9	gcdchm	Библиотека функций для решения задач химической кинетики

атрибутов: а) функциональных компонентов (таблица sys.aitms), в том числе: субплагинов для запуска решателей, функций-обработчиков, функций-предикатов, функций-селекторов; б) графоориентированых решателей (таблица com.slvrs); в) постановок задач (таблица tsk.tasks); г) вычислительных экспериментов (таблица srg.cmput); д) результатов расчетов (таблицы srg.cmpat, srg.cmpbt) и пр.

Разработка функциональных компонентов обработки данных. Как уже отмечалось, в основе вычислительной подсистемы лежит применение графоориентированного подхода [11], [12]. Подход обеспечивает комплексную систематизацию и упрощение процессов проектирования, разработки, тестирования, использования и сопровождения программных реализаций СВМ. В основе лежит формализация понятия «алгоритм решения задачи» с помощью определения понятий: графовая модель; входные данные; состояние данных и данные СВМ; функция перехода, функция-обработчик, функция-предикат, функция-селектор и пр. Теоретические основы подхода описаны в [11].

На рис. 4 представлена стандартная схема решения некоторой вычислительной задачи при условии реализации алгоритма ее решения с ис-

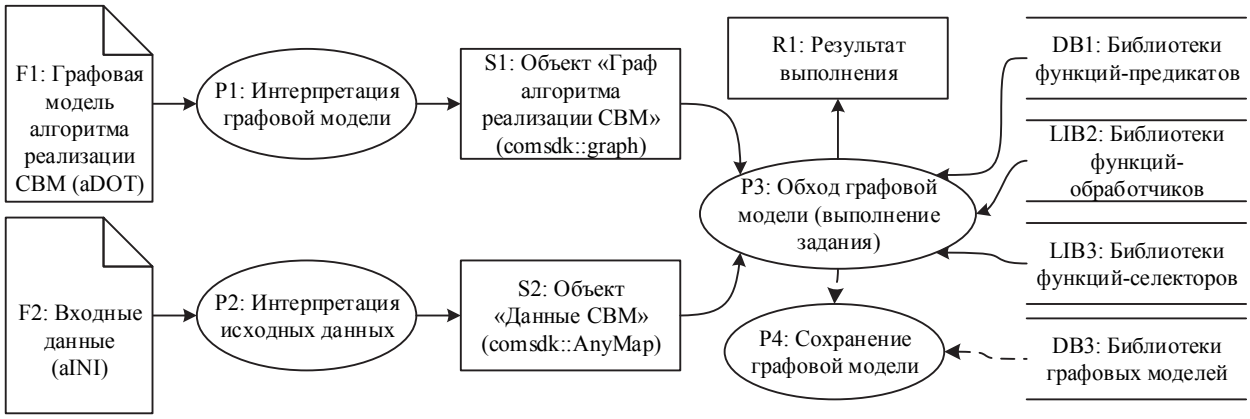

Рис. 4

пользованием графоориентированного подхода. Графовая модель представляет собой ориентированный граф. С каждым ребром графовой модели алгоритма связывается функция перехода, а каждому узлу ставится в соответствие состояние данных. Каждая функция перехода представляет собой пару: функция-предикат и функция-обработчик. Графовая модель описывается в формате aDOT [19], а входные данные представляют в формате aINI [18]. Совокупность указанных объектов подают на вход интерпретаторам P1 и P2: первый формирует объект, определяющий очередь(и) вызовов S1, а второй – объект входных данных S2. Последующий автоматический обход графа с вызовом соответствующих функций перехода [11] обеспечивает реализацию СВМ и решение соответствующей вычислительной задачи.

На рис. 5 представлена детализация процедуры запуска графоориентированного функционального компонента (числами обозначена последовательность обработки данных), где (1) – вызов функции редактирования; (2) – запрос атрибутов вызываемой функции; (3) – определение идентификатора решателя, который необходимо использовать для текущей задачи; (4) – определение имени файла в формате aDOT, соответствующего выбранному решателю, и обращение к git-репозиторию решателей; (5) – загрузка aDOT-файла; (6) – обращение к git-репозиторию вход-

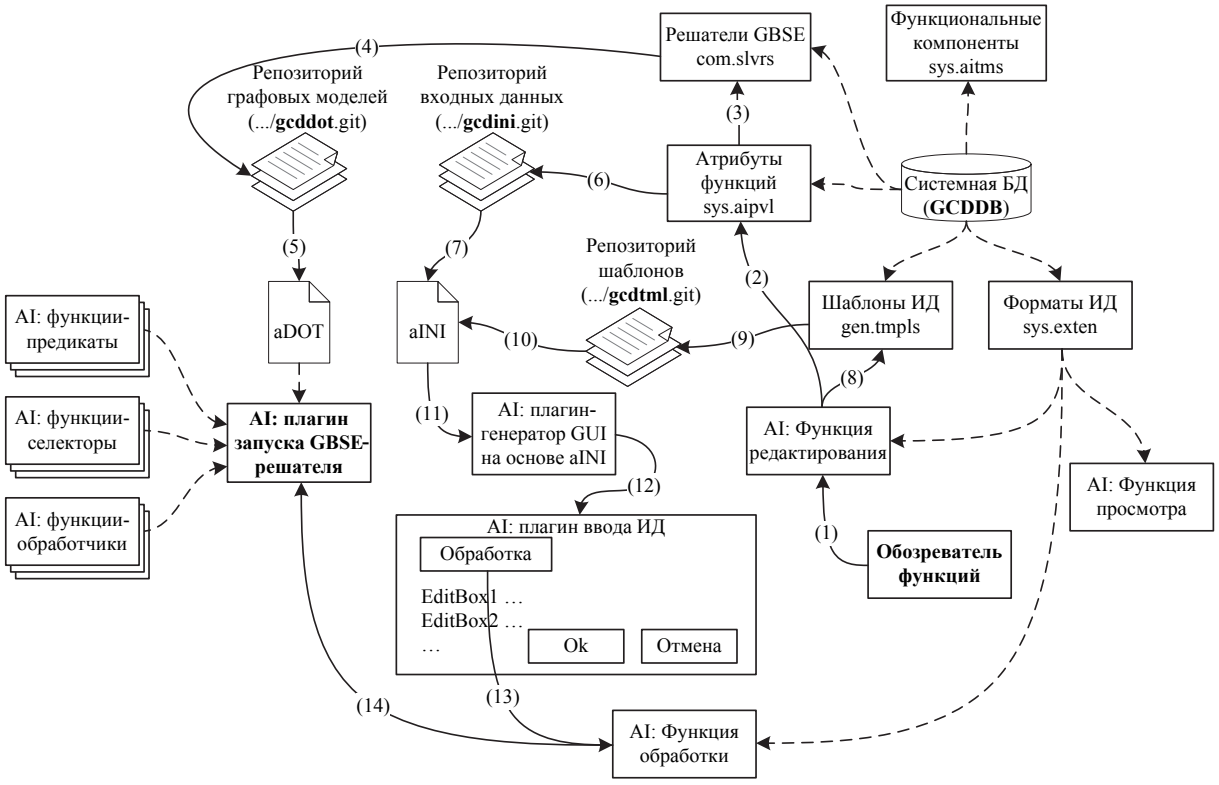

Рис. 5

ных данных в формате aINI; (7) – загрузка нужного aINI-файла (или формирование нового aINI-файла на основе шаблона, если нужного aINI-файла не существует в репозитории входных данных); (8) – определение имени файла шаблона входных данных; (9) – обращение к git-репозиторию шаблонов, в том числе входных данных для различных задач; (10) – формирование нужного aINI-файла; (11) – обращение к плагину-генератору графического пользовательского интерфейса (GUI) на основе полученного aINI-файла; (12) – интерпретация aINI-файла и генерация GUI [16]; (13) – вызов функции обработки; (14) – обращение к плагину, обеспечивающему обход графовой модели (Р3 на рис. 4), и непосредственный запуск процесса обхода.

Построение графовых моделей сложных вычислительных методов. Алгоритм создания графоориентированной реализации СВМ сводится к выполнению следующих процедур:

1. Определить перечень входных данных и представить их в текстовом виде с использованием языка определения входных данных aINI [18].

2. Определить множество состояний данных $\{S_i\}_1^n$, каждому из которых должен соответствовать момент успешного завершения очередной процедуры обработки данных (рис. 6, *а*). Среди состояний выделить начальное S_1 и конечное S_n. Начальное состояние данных соответствует моменту, когда входные данные, определенные в формате aINI, были загружены в оперативную память в формате объекта общих данных (многомерный ассоциативный массив).

3. Определить множество функций-обработчиков данных $\{f_s\}_1^S$ согласно реализуемому СВМ.

4. Каждому состоянию S_i сопоставить узлы N_i будущей графовой модели (рис. 6, *а*).

5. Связать узлы N_i ребрами $\left\{e_{ij}^k : i, j \in \{0, \ldots, (n-1)\},\ k \in \mathbf{N}\right\}$, связывающими состояния данных S_i и S_j, тогда как k – уникальный номер связи между этими состояниями данных (рис. 6, *б*).

6. Каждому ребру поставить в соответствие функцию перехода $F_{ij}^k = \langle f_s, p_t \rangle$, определяемую парой f_s – функция-обработчик, p_t – функция-предикат [11].

7. Определять или дополнять множество функций-предикатов $\{p_t\}_1^T$ и множество функций-селекторов $\{h_g\}_1^G$, также связывая их с соответствующими ребрами или узлами согласно их назначению [11] (рис. 6, *б*).

Перечисленные пункты алгоритма следует выполнять с использованием языка определения графовых моделей aDOT [19]. В результате проведенных процедур будет построена графовая модель необходимого СВМ (рис. 6, *в*) в формате aDOT.

Замечание 3. Отметим, что функции-обработчики $\{f_s\}_1^S$, функции-предикаты $\{p_t\}_1^T$ и функции-селекторы $\{h_g\}_1^G$ следует группировать по назначению и представлять в виде программных библиотек (рис. 6, *б*). Примеры таких библиотек представлены в таблице на с. 55.

Замечание 4. В рамках представляемой вычислительной подсистемы непосредственная разработка функций-обработчиков, функций-предикатов и функций-селекторов может осуществляться на двух языках программирования: C++ и Python, для каждого из которых были определены свои стандартные сигнатуры функций указанных трех типов (детализация выходит за рамки настоящей статьи).

Интерпретация (обход и выполнение) графовой модели (согласно рис. 4), в свою очередь, возможна только при предварительной реализации всех необходимых функций-обработчиков, функций-предикатов и функций-селекторов, их размещении в библиотеках согласно определению графовой модели и организации к ним доступа интерпретатору.

Замечание 5. Представляемый подход позволяет создавать в определенном смысле «универсальные» графовые модели, реализующие одновременно различные СВМ. Другими словами, сопоставляя с элементами графовой модели различные комбинации функций-обработчиков, функций-предикатов и функций-селекторов можно формировать реконфигурируемые графовые модели с единой топологией.

Замечание 6. Следует отметить, что графовые модели можно вкладывать друг в друга, что позволяет, описав 2 разных СВМ, встраивать один в другой, сохраняя возможность доработки каждого независимо друг от друга. С технической точки зрения это реализуется определением соответ-

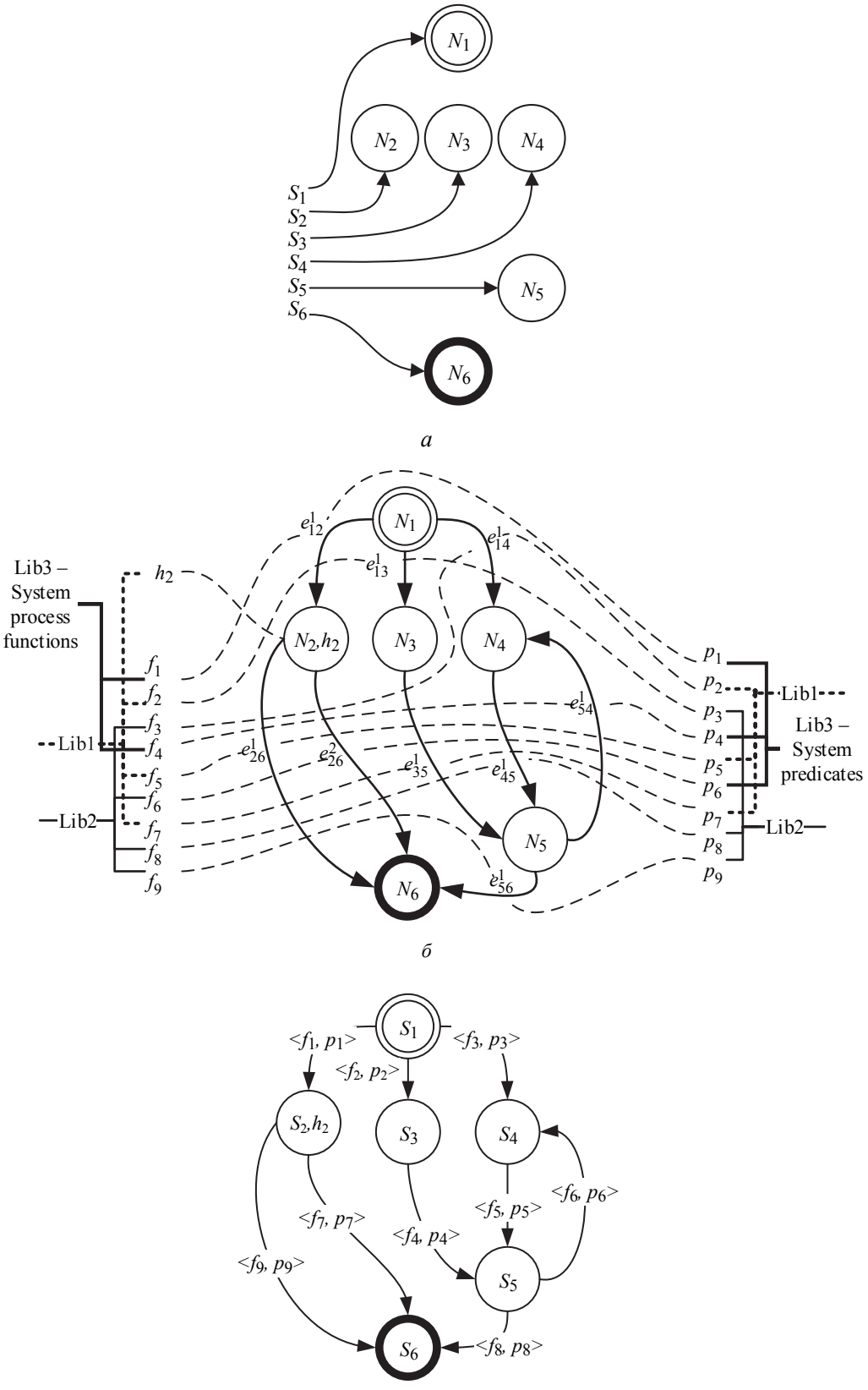

Рис. 6

ствующего свойства узла: «вложенная графовая модель, вкладываемая внутрь текущей». Такие возможности актуальны, например, при реализации методов оптимизации, когда алгоритм вычисления целевой функции реализуется одной графовой моделью, а сам метод оптимизации – другой, использующей первую.

Особенности организации распараллеливания и ветвления. Параллельные ветви графовой модели по умолчанию имеют одинаковый приоритет и могут выполняться параллельно. Однако существуют ситуации, когда параллельная обработка невозможна или нецелесообразна. Например, при отладке разработанного графоориентированного решателя удобен последовательный обход графа (рис. 7, *б*). В то же время, параллельная обработка зависит от наличия доступных аппаратных ресурсов, а также от программной поддержки на уровне интерпретатора графовых моделей. В связи с тем, что графовые модели можно вкладывать друг в друга (замечание 6) и каждую из них могут разрабатывать разные исследователи, очевидным становится тот факт, что может потребоваться одну графовую модель обходить последовательно, а вложенную в нее – параллельно. Более того, бывают ситуации, когда требуется обеспечить не распараллеливание, а ветвление (рис. 7, *в*). Таким образом, в процессе обхода графовой модели, «дойдя» до некоторого узла, из которого «выходят» несколько ребер, на уровне интерпретатора должна быть возможность принять решение о методе дальнейшего обхода. В рамках представляемой реализации графоориентированного подхода принятие такого решения осуществляется: а) определением стратегии распараллеливания или ветвления в узле; б) связыванием с узлом функции-селектора.

Обозначение 1. Если данные СВМ D находятся в состоянии S [11], то будем использовать краткое обозначение: $D \bullet\!\!-\!\circ S$.

Обозначение 2. Если $\{S_j\}_{j=1}^{m}$ – семейство состояний СВМ, являющихся разбиением состояния S [11], то будем использовать краткое обозначение: $S \rightrightarrows \{S_j\}_{j=1}^{m}$.

Далее используется терминология, введенная в [11].

Определение 2 (функция-селектор). Пусть $S \in C$ – некоторое состояние СВМ, $\mathbb{D} =$

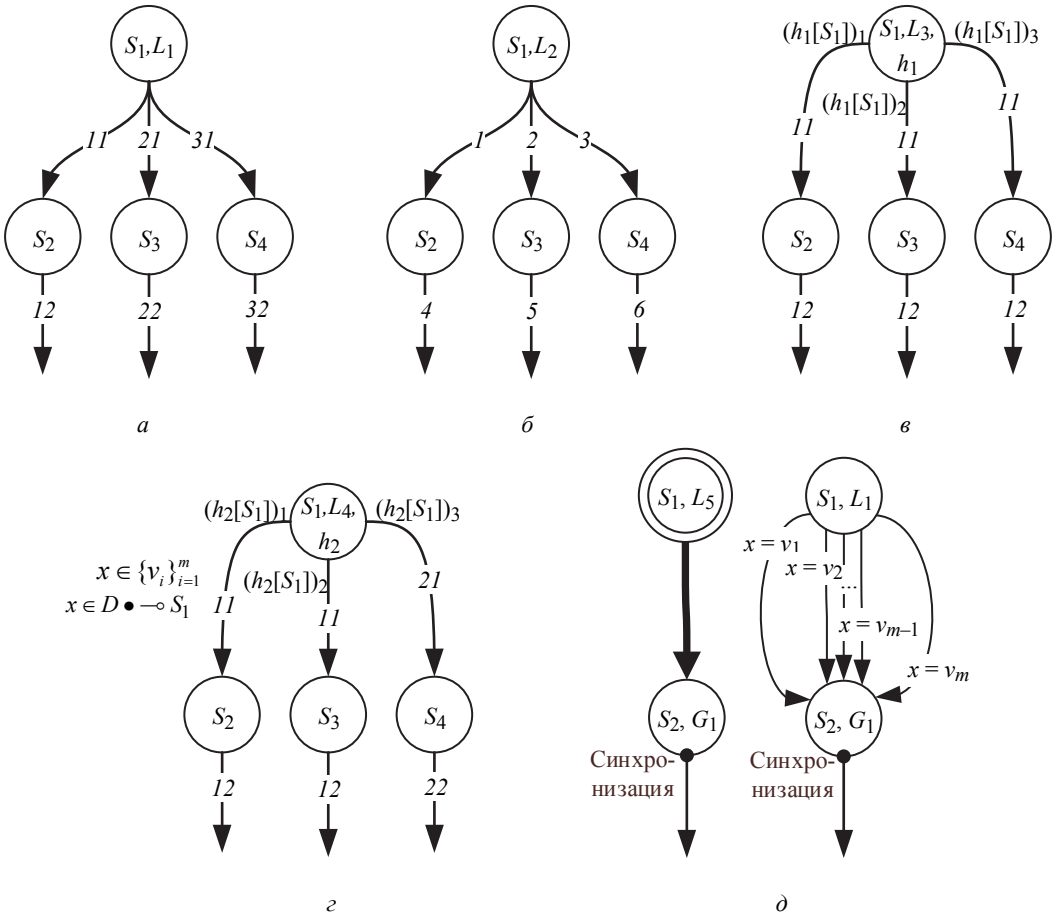

Рис. 7

$= \left\{ \{D_i\}_{i=1}^{n} : D_i \bullet\!\!-\!\!\circ S, n \in \mathbb{N} \right\}$ – семейство всех возможных данных СВМ, находящихся в состоянии S, а также $S \rightrightarrows \{S_j\}_{j=1}^{m}$. Тогда функцией-селектором будем называть такое отображение $h : \mathbb{D} \to \mathbb{B}^m$, где $\mathbb{B} = \{\text{TRUE}, \text{FALSE}\}$, $\mathbb{B}^m = \underset{k=1}{\overset{m}{\times}} \mathbb{B}$, $\forall D_i \in \mathbb{D} : D_i \bullet\!\!-\!\!\circ S \Rightarrow \exists! \mathbf{b} \in \mathbb{B}^m : h^{-1}(\mathbf{b}) = D_j \bullet\!\!-\!\!\circ S_j$.

Учитывая введенное определение, среди возможных стратегий распараллеливания L_i, связываемых с узлами, можно выделить (рис. 7):

а) L_1 – истинный параллелизм (рис. 7, *а*) – обход всех исходящих ребер (запуск связанных функций перехода) в независимых потоках/процессах (используется по умолчанию, функция-селектор не задана);

б) L_2 – псевдопараллелизм (рис. 7, *б*) – обход осуществляется последовательно согласно указанному порядку в одном потоке;

в) L_3 – ветвление (рис. 7, *в*) – обход лишь одного из исходящих ребер, выбор обеспечивается связанной с узлом функцией-селектором;

г) L_4 – общий случай, который соответствует необходимости распараллеливания и ветвления одновременно (для примера, на рис. 7, *г* в зависимости от результата вычисления функции-селектора h_2 будет выбрана одна из двух первых ветвей и третья, полученные ветви должны выполняться параллельно);

д) L_5 – неявный параллелизм (на рис. 7, *д* представлены эквивалентные графы).

Распределенные вычисления с использованием графоориентированного подхода. Предложенный графоориентированный подход и его программная реализация позволяют производить вычисления не только на локальном устройстве, но и задействовать удаленные устройства, например суперкомпьютеры. В такой конфигурации предполагается, что интерпретатор графа запущен на локальном устройстве, которое взаимодействует с множеством удаленных устройств посредством того или иного средства коммуникации. Запуск исполняемого файла на удаленном устройстве и контроль за его выполнением в этом случае соответствует отдельной функции-обработчику в рамках единой графовой модели, имеющей обобщенную структуру и позволяющей описать запуск произвольной команды со значениями ключей и параметров, которые будут подставлены в процессе интерпретации графовой модели из ее входных данных.

Суперкомпьютеры обычно предоставляют специальный сервис операционной системы для запуска программ в требуемом режиме, называемый планировщиком задач. В связи с тем, что актуальные удаленные устройства чаще всего являются суперкомпьютерами, программная реализация библиотеки для создания графовых моделей включает в себя инструменты для взаимодействия с Sun Grid Engine, позволяющие формировать постановку задач в виде специализированных скриптов, отправлять их в очередь планировщика задач и отслеживать, когда и с каким результатом их выполнение завершается.

В зависимости от уровня доступа к портам и в целом к операционной системе на удаленном устройстве библиотека позволяет использовать как простейшие средства коммуникации (например, ssh), так и полноценное клиент-серверное взаимодействие с использованием протокола TCP/IP и библиотеки zeromq. Важным элементом во взаимодействии с удаленными устройствами также является взаимная передача данных с локального устройства на удаленное (например, передача входных данных для расчета) и обратно (например, передача результатов расчета). В силу потенциально большого объема передаваемых данных требуется использование специализированных протоколов. В частности, программная реализация библиотеки для создания графовых моделей предполагает использование для этих целей протокола для обмена файлами sftp или программы синхронизации файлов и каталогов rsync. Подобное взаимодействие реализуется интерпретатором графовой модели автоматически, при условии, что архитектор графовой модели заранее указал в обобщенном виде, какие файлы и каталоги требуется копировать с локального устройства на удаленное и обратно. Отправка входных данных на удаленное устройство, последующий запуск исполняемого файла и скачивание результатов на локальное устройство представляют собой подграф специального назначения, где каждая из перечисленных операций реализуется с помощью соответствующей функции перехода.

Наличие возможности параллельного исполнения ребер графовой модели в контексте распределенных вычислений позволяет организовать не только параллельный запуск множества исполня-

емых файлов или одного исполняемого файла с различными аргументами командной строки, но и использовать для подобных целей одновременно несколько удаленных устройств. Интерпретатор графовой модели будет отслеживать статус выполнения команд на каждом из них независимо от остальных удаленных устройств.

Опыт использования. Графоориентированный подход в контексте распределенных вычислений был использован для анализа течения вязкой несжимаемой жидкости между двумя параллельными стенками, движущимися в противоположных направлениях вдоль продольной координаты с одинаковой и постоянной скоростью и осциллирующими вдоль поперечной координаты с амплитудой A и частотой ω (публикация в процессе подготовки). Фазовое пространство X динамической системы, определяемой оператором эволюции φ^t, порождаемым из решения дискретизированных уравнений Навье–Стокса с соответствующими граничными условиями для времени t, локально разделяется сепаратрисой на подпространство Φ быстро затухающих возмущений ламинарного течения и подпространство возмущений Σ, вызывающих фазовый переход из ламинарного в турбулентное состояние всего течения. Целью анализа было нахождение решений уравнений Навье–Стокса $u(t) = \varphi^t u_0$, где $u_0 \in X$ – начальное условие, в пределе сходящихся к сепаратрисе, для различных значений амплитуды A и частоты ω и дальнейшее нахождение A и ω, максимизирующих значение предельной кинетической энергии решения.

Алгоритм нахождения подобных решений, известный как edge tracking [20], [21], основан на методе бисекции, где в качестве пары начальных точек используются произвольные точки $u_1 \in \Phi$ и $u_2 \in \Sigma$. На отрезке $f(\lambda) = \lambda u_1 + (1-\lambda)u_2$, соединяющем эти точки в бесконечномерном пространстве, выбирается N равномерно распределенных точек $\tilde{u}_1, \ldots, \tilde{u}_N$. Далее, с помощью оператора классификации Λ, использующего внутри себя оператор φ^t, каждая из этих точек помечается либо как принадлежащая подпространству Φ, либо как принадлежащая подпространству Σ, после чего среди них выбирается такая пара точек $\tilde{u}_i$, $\tilde{u}_{i+1}$, что $\tilde{u}_i \in \Phi$ и $\tilde{u}_{i+1} \in \Sigma$. Переобозначив $u_1 = \tilde{u}_i$ и $u_2 = \tilde{u}_{i+1}$, получаем рекурсивный алгоритм, в пределе сходящийся к такому возмущению $\overline{u} = (u_1 + u_2)/2$, что решение $u(t) = \varphi^t \overline{u}$ лежит на сепаратрисе. Свойства рассматриваемой сепаратрисы таковы, что решение $u(t)$ при $t \to \infty$ стремится либо к стационарному, либо к периодическому, либо к хаотическому решению в зависимости от значений амплитуды A и частоты ω. В зависимости от типа решения значение предельной кинетической энергии вычисляется, используя тот или иной оператор осреднения во времени. Конечным шагом является вычисление предельных решений для множества пар A и ω, что позволяет выбрать их квазиоптимальные значения.

С вычислительной точки зрения наиболее требовательной частью алгоритма является вычисление траекторий $u(t) = \varphi^t u_0$, реализованное в специально предназначенном для этого решателе. Подобные вычисления выполнялись на удаленном компьютере, где располагался исполняемый файл решателя. Соответствующая логика инкапсулирована в графовой модели G_3 (TimeIntegrationGraph) (рис. 8). Приводим ее без описания и представления связанных функций перехода.

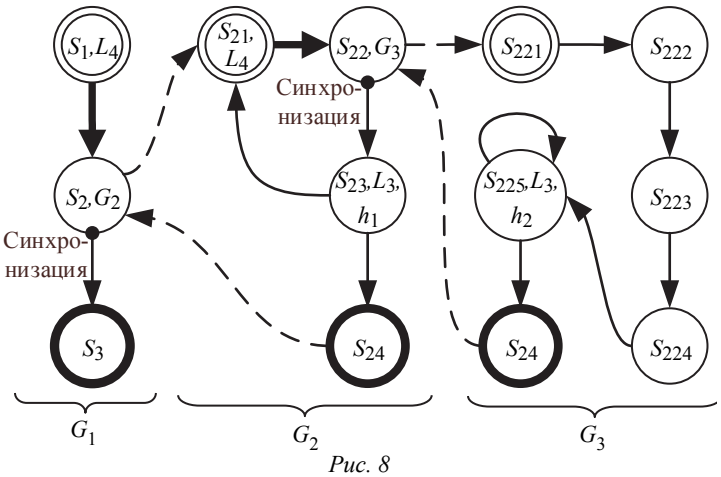

Рис. 8

Граф G_3 входит в качестве подграфа в граф G_2 (EdgeTrackingGraph), который реализует рекурсивный алгоритм edge tracking, описанный ранее, посредством цикличного переопределения входных данных для подграфа G_3. Граф G_2 также выполняется на удаленном компьютере. Наконец, перебор различных значений амплитуд A и частот ω, а также выбор квазиоптимальной пары значений происходит в основном графе G_1 (см. рис. 8) на локальном компьютере, который включает в себя G_2 как подграф при запуске алгоритма edge tracking для различных значений A и ω.

Следует отметить, что разделение логики алгоритма на локальную (т. е. исполняемую на локальном устройстве) и удаленную (т. е. исполняемую на удаленном суперкомпьютере) части позволило использовать решатель как black box и избежать его отдельной доработки. В то же время использование подграфов позволило создать обобщенный алгоритм с легко заменяемыми компонентами. Так, например, текущий алгоритм edge tracking может быть адаптирован для другой предметной области с минимальными трудозатратами заменой подграфа G_3 на соответствующий граф, реализующий вычисление траекторий $u(t) = \varphi^t u_0$. Если же подобное вычисление производится решателем, расположенным на удаленном суперкомпьютере, создавать отдельный граф не надо – достаточно заменить наименование решателя и список входных и выходных данных в подграфе G_3. С другой стороны, сам граф G_3 может быть включен в иные графы для решения сторонних задач (например, для исследования статистических свойств неустойчивых турбулентных течений, что требует многократного вычисления траекторий $u(t) = \varphi^t u_0$).

Использование описанной программной технологии позволило систематизировать процессы разработки вычислительных библиотек, обеспечило технологичность процессов разработки программных реализаций сложных вычислительных методов.

Универсальность представленного подхода позволяет использовать его при создании вычислительных подсистем различного назначения, что подтверждено на практике. Результатом работы можно считать обоснованную необходимость применения специальных подходов при разработке сложного наукоемкого программного обеспечения прикладного назначения.

Материал настоящей статьи дополняет теоретические основы графоориентированного подхода, впервые представленные в [11]: введено понятие *функция-селектор*, на основе которого стало возможным реализовать механизмы распараллеливания и ветвления.

Разработка архитектуры вычислительной подсистемы, создание прототипа ее программной реализации (comsdk, pycomsdk, comaps, gcdtml, pycomaps, GCDDB, gcddot), в том числе интеграция в рамки САПР КМ (PBC GCD v.4), включая разработку теоретических основ подсистемы и графоориентированного подхода, осуществлены авторами в МГТУ им. Н. Э. Баумана на инициативной основе.

Создание отдельных предметных вычислительных библиотек, разрабатываемых в рамках представленных программных технологий, осуществлялось в разные годы при следующей финансовой поддержке: госзадание 1.6260.2011 (gcdfes, gcdini), гранты Президента РФ МК-6421.2012.9 (gcdhom), МК-765.2012.8 (gcdhom, gcdopt), ФЦП «Исследования и разработки» 14.577.21.0135 (comwpc), 14.574.21.0158 (comwpc).

Разработка вычислительной библиотеки gcdchm выполняется при финансовой поддержке РФФИ в рамках научного проекта № 18-07-00341. Разработки библиотек gcdcmi, gcdfes, gcdnrg продолжаются в настоящее время.

СПИСОК ЛИТЕРАТУРЫ

1. Ильин В. П., Скопин И. Н. Технологии вычислительного программирования // Программирование. 2011. Т. 37, № 4. С. 53–72.

2. Лазарев И. В., Сухорослов О. В. Использование Workflow–методологии для описания процесса распределенных вычислений // Тр. Ин-та системного анализа Российской акад. наук. 2005. Т. 14. С. 26–70.

3. Common Workflow Language (CWL). URL: https://www.commonwl.org/ (дата обращения 22.04.2020)

4. Yu J., Buyya R. A. taxonomy of scientific workflow systems for grid computing // SIGMOD Record. 2005. Vol. 34, №. 3. P. 44–49.

5. Talia D. Workflow Systems for Science: Concepts and Tools // ISRN Software Engineering. 2013. Vol. 2013. P. 1–15.

6. De Roure D., Goble C. Software design for empowering scientists // IEEE Software. 2009. Vol. 26, № 1. P. 88–95.

7. Pegasus: a workflow management system for science automation / E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, K. Wenger // Future Generation Computer Systems. 2015. Vol. 46. P. 17–35.

8. Workflows and e–Science: An overview of workflow system features and capabilities / E. Deelman, D. Gannon, M. Shields, I. Taylor // Future Generation Computer Systems. 2009. Vol. 25. P. 528–540.

9. Malyshkin V. E., Perepelkin V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run–Time Subsystem // Lecture Notes in Computer Science. 2011. Vol. 6873. P. 53–61.

10. Артамонов Ю. С., Востокин С. В. Инструментальное программное обеспечение для разработки и поддержки исполнения приложений научных вычислений в кластерных системах // Вестн. Сам. гос. техн. ун–та. Сер.: Физ.-мат. науки. 2015. Т. 19, № 4. С. 785–798.

11. Sokolov A. P., Pershin A. Yu. Graph–Based Software Framework for Implementation of Complex Computational Methods // Programming and Computer Software. 2019. Vol. 45, № 5. P. 257–267.

12. Пат. RU 2681408. Способ и система графо–ориентированного создания масштабируемых и сопровождаемых программных реализаций сложных вычислительных методов / А. П. Соколов, А. Ю. Першин. Опубл. 22.02.2019.

13. Каменев А. В., Ишакова Е. Н. Специфика инженерного образования в области разработки программного обеспечения // Интеллект. Инновации. Инвестиции. Спец. выпуск. 2011. С. 78–81.

14. Липаев В. В. Инженерная психология при производстве программных продуктов // Программная инженерия. 2011. Т. 1. С. 7–15.

15. Многоуровневый подход к разработке алгоритмического и программного обеспечения экзафлопсных суперЭВМ / Б. М. Глинский, И. М. Куликов, А. В. Снытников, И. Г. Черных, Д. В. Винс // Вычислительные методы и программирование: новые вычислительные технологии. 2015. Т. 16, № 4. С. 543–556.

16. Соколов А. П., Першин А. Ю. Программный инструментарий для создания подсистем ввода данных при разработке систем инженерного анализа // Программная инженерия. 2017. Т. 8, № 12. С. 543–555.

17. Колбин И. С. Программный комплекс для решения задач математического моделирования с использованием нейросетевой методологии // Программная инженерия. 2013. № 2. С. 25–30.

18. Соколов А. П., Першин А. Ю. Описание формата данных aDOT (advanced DOT). URL: https://sa2systems.ru/nextcloud/index.php/f/403526 (дата обращения 05.03.2020)

19. Соколов А. П. Описание формата данных aINI (advanced INI). URL: https://sa2systems.ru/nextcloud/index.php/f/403527 (дата обращения 05.03.2020)

20. Skufca J. D., Yorke J. A., Eckhardt B. Edge of chaos in a parallel shear flow // Physical review letters. 2006. Vol. 96, № 17. P. 174101.

21. Laminar–turbulent boundary in plane Couette flow / T. M. Schneider, J. Gibson, M. Lagha, F. De Lillo, B. Eckhardt // Physical Review E. 2008. Vol. 78, №. 3. P. 037301.

A. P. Sokolov, A. Yu. Pershin
Bauman Moscow state technical university

COMPUTER-AIDED DESIGN OF COMPOSITE MATERIALS. PART 2: COMPUTATIONAL SUBSYSTEM, DISTRIBUTED COMPUTATIONS USING GRAPH-BASED SOFTWARE ENGINEERING

The paper presents a computational subsystem and its architecture, which is the part of the prototype of computer-aided design system for composite materials (CAD CM). A review of the literature was conducted and well-known software approaches to create e-science software, including methodologies for implementing computational methods, is presented. The importance of using special approaches while developing e-science software is proven. The paper focuses on methods for developing extensible and supportable computing libraries that implement complex computational methods (CCM). Features that distinguish the development of a software implementation of the CCM from the software implementation of a computational algorithm for solving a specific problem is presented. The list of technical requirements for the computational subsystem of CAD CM, as well as principles which has formed the basis for this subsystem, is given. The concept of «solver» is introduced and its relationship with other system concepts as «functional component», «numerical subsystem», «SVM» and others which were presented in the first part of the series of articles is presented. It is presented a detailed procedure for solving a computational problem based on using of corresponding graph model of computational method. The list of parts of the developed computational subsystem is given. The principles of developing libraries of computational functions using a graph-based approach are presented. The concept of «selector function» is introduced, as well as a description of parallelization and branching strategies involving the use of such functions. An example of using a graph-based approach in solving an applied problem of analyzing the turbulent flow of a viscous incompressible liquid is presented.

Technologies of development of engineering software, graph-based software engineering, development of computational libraries, computer-aided design of composite materials, computer-aided engineering, extension modules development, distributed computing technologies, workflow systems, e-science tools