ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

ПрИн 1/2013

Учредитель: Издательство "НОВЫЕ ТЕХНОЛОГИИ"

Издается с сентября 2010 г.

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана, ОАО "Концерн "Сириус".

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru E-mail: prin@novtex.ru

Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

УДК 004.4, 004.9

И. Е. Пелепелин[1], канд. физ.-мат. наук, вед. науч. сотр., e-mail: IPelepelin@blogic20.ru,
В. В. Феклистов[1], стар. науч. сотр., e-mail: vfeklist@mail.ru,
С. В. Карандин[1], науч. сотр., e-mail: skarandin@samelogic.ru,
Д. Р. Башкирцев[2], инженер, e-mail: DBashkirtsev@it.ru,
А. С. Зиннатуллин[3], стажер, e-mail: artem.zinnatullin@gmail.com,
[1]Национальный исследовательский университет "Высшая школа экономики", г. Москва,
[2]Группа компаний "АйТи", г. Уфа,
[3]Группа компаний "АйТи", г. Санкт-Петербург

Интеграция Metasonic Suite и Alfresco при разработке реестра сервисов с использованием принципов S-BPM

На примере субъектно-ориентированного подхода к разработке реестра сервисов и сопутствующего бизнес-процесса рассматриваются вопросы интеграции программных продуктов Metasonic Suite и Alfresco.

Ключевые слова: *интеграция, реестр сервисов, Metasonic Suite, Alfresco, S-BPM, CMIS, SOA*

Введение

Для компаний-разработчиков программного обеспечения всегда было актуальным добиться такого положения вещей, когда однажды разработанный для какого-либо проекта программный код можно было бы использовать повторно в других проектах. Такие повторно используемые программные компоненты традиционно представляются в виде библиотек кода и имеют ряд недостатков, основными из которых являются:

- слабая структурированность библиотек программного кода;
- отсутствие возможности организации перекрестных ссылок в описании элементов библиотеки;
- трудности поиска нужного компонента по его функциональным параметрам и по эффективности использования для выполнения того или иного нового проекта.

Устранить указанные недостатки призван Реестр сервисов (далее Реестр), который реализован на базе системы управления информационными ресурсами предприятия (*Enterprise content management* — ECM). Однако мир несовершенен и, как следствие, избавившись от одних недостатков, мы тут же получаем ряд новых. Поскольку Реестр становится централизованным ресурсом, необходим определенный регламент по добавлению новых компонентов или по изменению существующих. Другими словами, нужен механизм, с одной стороны, препятствующий превращению Реестра в "свалку" а с другой стороны, не слишком сложный, чтобы не отпугнуть потенциальных "инвесторов". Под инвесторами здесь будем понимать тех сотрудников, которые предоставляют свои разработки для внесения их в Реестр.

Исследование, результаты которого представлены в настоящей статье, посвящено повышению эффективности разработок внутри компании, в частности, за счет ориентации на сервис-ориентированную архитектуру (*Service-Oriented Architecture* — SOA) с использованием исследований и результатов консорциума OASIS [1]. При этом проблемы, связанные с созданием стимулов для наполнения Реестра, остаются за рамками данной статьи.

Выбор методов и средств ведения Реестра

В парадигме SOA понятия сервис и компонент не тождественны. Согласно референсной модели SOA [2], сервис — это совокупность сведений о предоставляемой возможности. А программный компонент — это логически обособленная программа или библиотека, которая может быть оформлена как сервис, при условии соответствия компонента спецификациям сервиса. Иными словами, сервис реализуется компонентом. Создание сервиса процесс творческий, требующий участия на разных этапах разработки специалистов различ-

ной квалификации, взаимодействие которых в процессе его создания должно быть строго регламентировано. Вместе с тем сам процесс разработки хотя и поддается регламентации, может быть построен различными способами, в зависимости от квалификации участников, инструментария, которым они владеют, а также от используемых ими технологий. Наибольший эффект достигается в том случае, если процесс создания на каждой стадии настраивается самим исполнителем, при условии выполнения им обязательств по предоставлению результатов своей деятельности другим участникам в стандартизованном виде.

Исходя из представленных выше соображений, в качестве систем управления процессом создания сервиса в наибольшей степени подходят системы, основанные на субъектно-ориентированном управлении. В таких системах исполнитель рассматривается не как ресурс, а как творческая единица, способная к самоорганизации и оптимизации своей деятельности в процессе исполнения заданий. Одной из таких систем является Metasonic Suite, которая позволяет упорядочить процесс взаимодействия участников разработки, оставляя им практически полную свободу действий в рамках решения конкретной задачи на разных этапах проектирования сервисов и разработки реализующих их компонентов.

Кроме стандартизации взаимодействия в процессе разработки компонентов необходимо также определить стандарты описаний сервисов при их проектировании. Такие стандарты должны быть понятны всем участникам процесса, а также потребителями сервиса, которые будут использовать его в своих бизнес-решениях. Для хранения сведений о сервисах и реализующих их компонентах служит Реестр, рассматриваемый в настоящей статье. В хранилище этого Реестра должны содержаться описания бизнес-требований, функциональные спецификации, контракты, содержащие нефункциональные характеристики сервисов, а также описания интерфейсов и реализаций сервисов.

В настоящее время существует три подхода к организации хранения и обработки информации о программных компонентах, использующихся при создании систем в парадигме сервис-ориентированной архитектуры:

- репозитории сервисов;
- реестры сервисов;
- комплексные решения по управлению архитектурой предприятия.

Репозитории сервисов предназначены для хранения информации об объектах SOA-систем, а именно — сервисах, их реализации, контрактах и другой подобной информации. По факту, это база данных или хранилище неструктурированной информации [3]. Как вырожденный случай, это может быть просто ресурс в файловой системе.

Реестры сервисов представляют собой решения, которые предоставляют возможность пользоваться репозиторием сервисов упорядоченным, контролируемым способом. Реестры сервисов различаются по назначению и области применения. Например, для решения задач по управлению инфраструктурой предприятия актуальны функции обнаружения с применением стандартов UDDI/WSDL. Такие репозитории могут применяться для автоматизации приемочного тестирования, мониторинга, инвентаризации, вывода сервисов из эксплуатации. Примеры таких реестров:

- Apache: jUDDI (http://uddi.xml.org/apache-juddi);
- Ruddy (http://www.ruddi.org/);
- WebSphere Service Registry and Repository (http://www-01.ibm.com/software/integration/wsrr/index.html).

В случае если речь идет о более ранних этапах создания программного сервиса, таких как проектирование и разработка, становятся актуальными функции совместной разработки, согласования описания, контрактов и программного кода, поиск сервисов по назначению и другие подобные им. Примеры таких реестров:

- Membrane (http://www.membrane-soa.org/soa-registry/);
- WSO2 Governance Registry (http://wso2.com/products/governance-registry/);
- SOA service manager (http://www.soa.com/products/service_manager/).

К общим решениям по управлению программными комплексами на основе SOA относятся приложения, обладающие целым спектром назначений. Они, как правило, используются в больших организациях, для которых актуальны вопросы управления архитектурными компонентами и активами предприятия в целом. В спектр задач таких решений входит не только архитектура приложений, но и технологическая, бизнес-архитектура. Ниже приведен перечень продуктов и вендоров, согласно аналитическому отчету Kristian Steenstrup из компании Gartner за 2010 г.

- IBM Maximo Asset Management
- IFS
- Invensys Operations Management (Avantis)
- Mainsaver
- Mincom
- Oracle E-Business Suite, Oracle
- SAP
- Ventyx, ABB Company.

Создатели этих продуктов в первую очередь ориентируются на организации, являющиеся потребителями программного обеспечения, а не его разработчиками. По этой причине они неэффективны в решении вопросов, рассматриваемых в настоящей статье.

Хранилище Metasonic Suite не может в достаточной степени решить задачи, стоящие перед Реестром в силу целого ряда ограничений, основным из которых является ненормализованная модель хранения данных. Кроме того, хранилище данных Metasonic Suite требует дополнительных усилий по проектированию интерфейса для потребителей сервисов, что вызывает необходимость выбора специального хранилища для реализации Реестра.

В результате проведенных исследований выбор авторов был остановлен на хранилище ECM-плафформы Alfresco. Основными преимуществами этого хранилища является режим его свободного распространения, а также возможность обращения к нему через стандартный де-факто протокол CMIS (*Content Management Interoperability Services* — сервисы взаимодействия при управлении контентом) [1].

Существует несколько способов, поддерживающих механизмы взаимодействия системы субъектно-ориентированного управления разработкой программного обеспечения и хранилища данных. Такое хранилище предназначено для упорядоченного хранения описаний сервисов и реализующих их программных компонентов в парадигме сервис-ориентированной архитектуры. В том числе существуют способы, эффективно реализующие эти механизмы для систем Metasonic-Alfresco.

Один из упомянутых выше способов заключается в построении модели управления разработкой программного обеспечения и бизнес-объектов, хранящих информацию о создаваемых компонентах на базе Metasonic Suite, и разработке сервисов, обеспечивающих взаимодействие экземпляра процесса разработки с Реестром описаний сервисов и реализующих их компонентов, который хранится в Alfresco.

Другой способ — на базе Metasonic Suite строится только управление процессом разработки. Работа с хранилищем при этом осуществляется в интерфейсе Alfresco с использованием его стандартных интерфейсов. При таком подходе в хранилище будут также накапливаться сведения о процессе разработки.

Третий способ представляет собой комбинацию первых двух, в этом случае управление процессом осуществляется на базе Metasonic Suite. Там же создаются бизнес-объекты, отвечающие за ход исполнения процесса разработки, а сущности, хранящие сведения о сервисах, заполняются в интерфейсе Alfresco.

Основным критерием при выборе эффективного способа рассматриваемого взаимодействия является трудоемкость его реализации. Во втором и третьем из упомянутых выше случаев она оказывается выше, так как требует разработки большего числа сервисов, обеспечивающих взаимодействие.

В первом случае необходимо создание сервисов, поддерживающих механизмы синхронизации значений атрибутов бизнес-объектов и сущностей хранилища данных. К ним относятся сервисы установления связи с хранилищем данных, создания и наполнения сущности в хранилище Alfresco, модификации сущности, установления связи между сущностями.

Во втором и третьем случаях необходимость в сервисах синхронизации данных отпадает, так как заведение и редактирование атрибутов сущностей ведется в стандартном интерфейсе Alfresco. Однако при этом возникает необходимость в создании сервисов, обеспечивающих передачу управления от Alfresco в Metasonic Suite. К ним относятся сервисы запуска процесса, работы с экземплярами процессов, работы с описаниями сущностей, связывания экземпляра сущности с экземпляром процесса (выполняются на стороне Metasonic Suite), работы с заданиями, работы с описаниями сущностей (выполняются на стороне Alfresco).

Выбор способа интеграции

Современные средства моделирования и автоматизации бизнес-процессов в отдельной организации развиваются в направлении большей адаптируемости к постоянным изменениям и усложнениям бизнес-окружения. Такие средства обеспечивают более высокий уровень понимания (восприятия) и доступности описаний используемых моделей для самих участников (субъектов) бизнес-процессов в рассматриваемой организации. В последние годы из общей концепции процессного управления организацией выделилось направление, основанное на субъектно-ориентированном подходе к описанию бизнес-процессов *Subject-oriented Business Process Management* (S-BPM) [4]. В S-BPM основное внимание сосредоточено на действующих лицах бизнес-процессов — субъектах, и все вопросы рассматриваются с позиции субъекта. В 2011 г. S-BPM попал в отчет аналитиков компании Gartner "Цикл зрелости технологий" (в оригинале *Hype Cycle for Business Process Management*) как новая развивающаяся технология [5]. В частности, S-BPM реализован в продукте Metasonic Suite разработки компании Metasonic AG.

В большинстве организаций, особенно в крупных, очень остро стоят вопросы интеграции между различными информационными ресурсами и системами. Как правило, в бизнес-процессы бывают вовлечены информационные ресурсы от разных производителей. Это обстоятельство приводит к необходимости решения задач по организации их взаимодействия как на уровне используемой системной архитектуры, так и на уровне отдельных функций конкретного бизнес-процесса. В данной статье ограничимся только рассмотрением вопроса интеграции системы S-BPM Metasonic Suite с другими системами.

В крупных организациях, особенно в Германии, часто используются продукты немецкой компании SAP AG и вопросы интеграции именно с этой системой на текущий момент являются наиболее проработанными. Так, например, в статье [6] подробно описаны способы интеграции Metasonic Suite с системами SAP ERP и SAP CRM. По поводу интеграции с другими системами, например, класса Enterprise content management (ECM), информации пока практически нет.

Наиболее удобным и гибким средством для интеграции процессов, запущенных в среде Metasonic, с другими системами является механизм, именуемый "Refinements" [7]. В рамках оболочки построения бизнес-процессов Metasonic Build этот механизм позволяет добавлять программный код сторонних разработчиков, определяя тем самым действия, которые будут выполнены в соответствующем состоянии субъекта бизнес-процесса. Таким образом, используя "Refinements", внутреннее поведение объекта бизнес-процесса может быть существенно изменено. Это дает в руки разработчика гибкий механизм, в том числе и для целей интеграции [8].

Задача интеграции информационных систем возникла одновременно с появлением собственно информационных систем и имеет несколько классических способов решения, описанных в литературе и активно применяемых на практике. Интеграция разнородных, особенно гетерогенных, систем может осуществляться на различных уровнях, в частности: на уровне пользовательских интерфейсов, на уровне информационных ресурсов, на уровне корпоративных приложений, на уровне сервис-ориентированной архитектуры [9].

Способы интеграции различаются в зависимости от структуры взаимодействия, метода интеграции и типа обмена данными.

Структурой взаимодействия в случае решения задачи интеграции двух систем (в нашем случае — Metasonic Suite и Alfresco) может являться только способ типа "точка-точка". При этом метод интеграции может быть выбран также только один — метод интеграции по данным. Типов же обмена данными может быть несколько: файловый обмен, совместное использование базы данных, обмен сообщениями, удаленный вызов процедур.

Файловый обмен предполагает наличие процедур экспорта-импорта данных из одного хранилища (базы данных) в другое. Преимуществом такого типа обмена данными является его независимость от процессов, происходящих в системах. Обмен может осуществляться по расписанию, в то время, когда нагрузка на локальную сеть минимальна, или по команде администратора, ответственного за взаимодействие. Существенным недостатком такого типа обмена является неизбежная потеря сведений о промежуточных состояниях процессов и объектов учета, создаваемых, модифицируемых и используемых в процессах различных систем.

Совместное использование базы данных является способом обеспечения взаимодействия в реальном времени, но накладывает серьезные ограничения на возможности изменения связанных систем.

Обмен сообщениями, который может осуществляться синхронно с изменениями, происходящими в ходе реализации процессов, лишен этого недостатка. Однако такой подход требует гораздо более сложной инфраструктуры организации взаимодействия и использования таких средств как общая интеграционная шина предприятия.

Наиболее простым и надежным является синхронный удаленный вызов процедур, реализованных в виде web-сервисов системы, поддерживающей хранилище данных (в нашем случае — Alfresco). Обращение к web-сервисам при этом должно осуществляться в момент завершения, создания и/или модификации экземпляров объектов учета.

Этапы процесса наполнения реестра

Поскольку наполнение Реестра данными предполагает автоматизацию определенных бизнес-процессов, рассмотрим отдельные этапы этого процесса в привязке к процедуре разработки сервиса:

- формирование бизнес-требований;
- экспертиза бизнес-требований;
- разработка спецификаций;
- разработка контракта;
- разработка реализации;
- тестирование.

Так как ни один из стандартов, в том числе государственных, не определяет модель жизненного цикла сервисов и реализующих их компонентов, авторами предлагается модель, аналогичная широко распространенной каскадной или последовательной модели. Ее характерной особенностью является необходимость формирования полной и согласованной документации на каждом этапе проекта.

В процессе проектирования сервисов и разработки реализующих их компонентов могут принимать участие пользователи, обладающие следующими ролями: заказчик, эксперт, аналитик, архитектор, разработчик, тестировщик. В роли заказчика в ходе разработки может выступать как собственно представитель заказчика, так и архитектор, который в случае крупного проекта осуществляет консолидацию составных его частей. При этом началом работы над новыми бизнес-требованиями по реализации того или иного сервиса (компонента) должен являться поиск его описания в Реестре.

В качестве эксперта в процессе разработки, как правило, выступает коллегиальный орган, осуществляющий надзор за разработкой проектов и их компонентов в организации или в группе организаций.

Роль аналитика предполагает создание спецификаций, понятных как разработчикам, так и экспертам, которые утверждают эти спецификации, отвечая за их соответствие требованиям реального заказчика.

Архитектор обеспечивает разработку контракта сервиса или компонента и надзор за реализацией сервиса или компонента в соответствии с контрактом, утвержденным коллегиальным органом. Утвержденный контракт становится доступным другим разработчикам, которые могут на него подписаться.

Разработчик — это программист, реализующий компоненты, соответствующие контракту.

Тестировщик тестирует разработанную реализацию.

Регламент, обеспечивающий взаимодействие участников процесса, накладывает ограничения на правила взаимодействия, а также на внутреннее поведение субъектов, участвующих в процессе. Однако каждый из них может самостоятельно менять модель своего внутреннего поведения, используя привычные ему методы и средства решения задач, что является одним из основных преимуществ предлагаемого в настоящей работе субъектно-ориентированного подхода к управлению процессом разработки.

Модель взаимодействия субъектов (регламент), участвующих в процессе разработки, представлена на рис. 1.

В процессе исполнения регламента сведения, которые попадут затем в хранилище в соответствии с моделью, необходимо вводить и преобразовывать к формату хранилища.

Ввод сведений в основном должен осуществляться вручную, так как эти данные появляются в результате регламентированной деятельности субъектов на различных стадиях этапа разработки сервисов и реализующих их программных компонентов. Для хранения сведений в процессе исполнения регламентов служит следующий набор бизнес-объектов:

- бизнес-требования (*Requirement*);
- контракт (*Contract*);
- реализация (*Realization*);
- спецификация (*Specification*);
- запрос (*Inquiry*);
- уведомление (*Notice*);
- задание (*Task*).

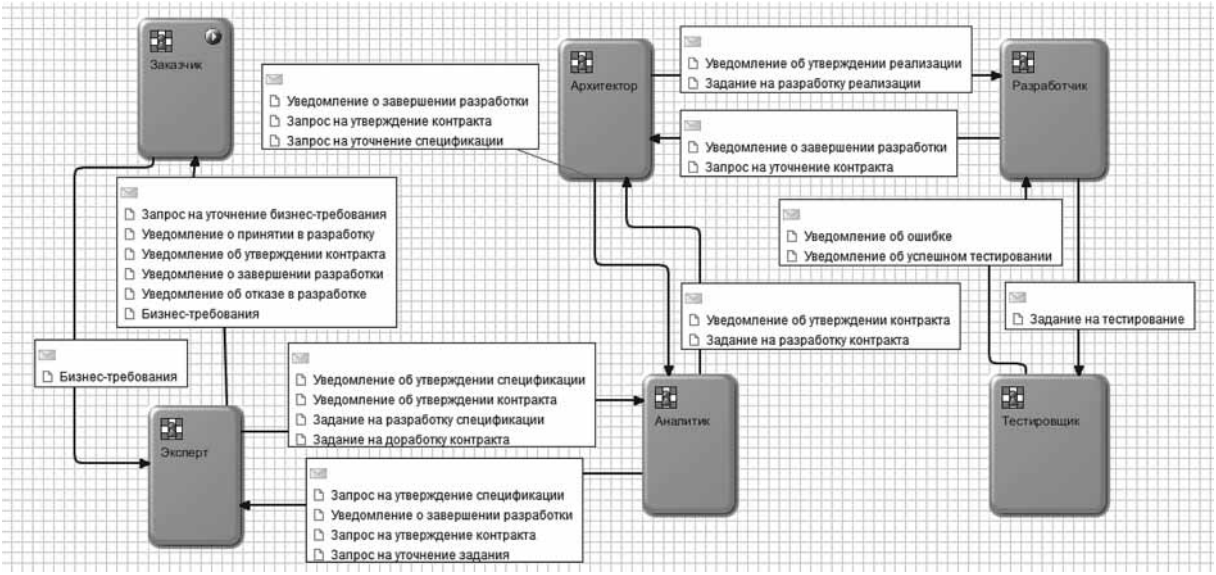

Рис. 1. Модель взаимодействия участников процесса разработки

Каждый из бизнес-объектов содержит набор атрибутов, необходимый как для учета свойств регламентирующих их сервисов и компонентов, так и для учета действий участников процесса, а также состояний бизнес-объекта в ходе исполнения регламента.

Модель данных

За основу модели данных Реестра описаний сервисов и реализующих их компонентов авторами взяты рекомендации консорциума OASIS [10], из которых выделены для включения в состав описаний сервисов следующие основные сущности:

- реестр сервисов (*ServiceRegistry*), содержащий краткое описание реестра сервисов;
- домен сервисов (*ServiceDomain*), содержащий краткое описание домена, в котором расположен сервис;
- описание сервиса (*ServiceDescription*);
- контракт (*Contract*), содержащий нефункциональные требования;
- реализация сервиса (*Realisation*), содержащая описание компонента, реализующего сервис;
- дополнительные наименования сервиса (*Sinonims*);
- предметная область (*ObjectArea*), содержащая краткое описание одной или нескольких предметных областей, например, Управление персоналом, Организационное развитие, Финансовый учет и контроль, Сбыт и др.;
- целевые пользователи (*User*), такая сущность содержит описания ролей на основе ролевой модели системы, для обладателей которых сервис предназначен;
- обеспечиваемые процессы (*Process*), эта сущность содержит описания бизнес-процессов, при автоматизации которых используется сервис, с указанием статуса (используется, планируется к использованию);
- бизнес-сущности (*BusinessEntity*), содержащие описания сущностей предметной области, с которыми связаны возможности, предоставляемые сервисом;
- атрибуты бизнес-сущности (*Attribute*);
- бизнес-операции (*BusinessMethod*), содержащие описания операций в предметной области, которые обеспечиваются возможностями, предоставляемыми сервисом;
- входные данные бизнес-операции (*InputData*);
- выходные данные бизнес-операции (*OutputData*);
- коды возврата бизнес-операции (*Return*);
- типовые решения SOA (*Pattern*), эта сущность содержит перечень паттернов, примененных при проектировании сервиса;
- соответствие стандартам (*Standart*), такая сущность содержит перечень стандартов (технические стандарты и рекомендации (W3C, OASIS, IETF и др.), стандарты и регламенты предприятия, государственные и отраслевые стандарты), которым соответствует сервис;
- ключевые индикаторы сервиса (KPI), эта сущность содержит описания измеримых критериев и их значения, при достижении которых принимается, что сервис эффективен.

Средства и методы взаимодействия

Интеграция информационных ресурсов подсистем, обеспечивающих автоматизацию разработки сервисов и реализующих их компонентов (подсистема управления потоками работ и подсистема хранения информации), основана на сервисах. Для сохранения данных в хранилище Alfresco используются коллекции Java-библиотек, фреймворков и механизмов Apache Chemistry OpenCMIS [11]. Эти инструментальные средства позволяют реализовать разработку программного продукта в соответствии с пакетом стандартов, состоящим из набора web-сервисов для совместного использования информации, хранимой в несвязанных между собой хранилищах контента. Модель данных реализована согласно рекомендациям по созданию пользовательских моделей в хранилище Alfresco [12].

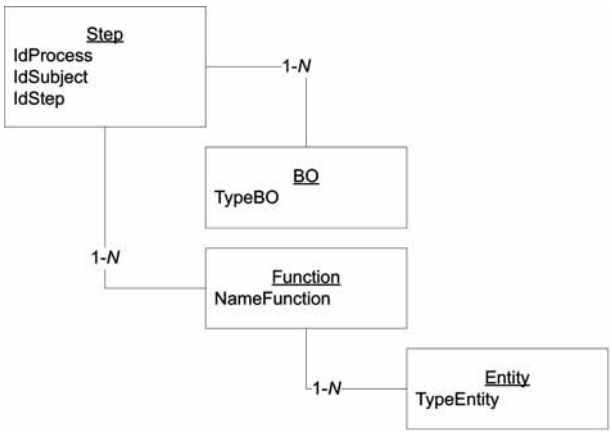

Рис. 2. Модель схемы процесса:
1—*N* — связь "один ко многим"

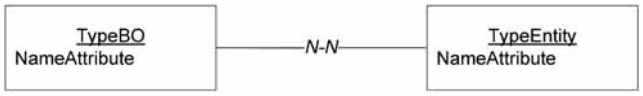

Рис. 3. Модель схемы преобразования:
N—*N* — связь "многие ко многим"

В целях рассматриваемой в контексте настоящей статьи интеграции создан следующий набор сервисов:

- установления связи с сервером Alfresco и открытия сессии взаимодействия;
- создания экземпляров сущностей и связей;
- сохранения экземпляров сущностей и связей.

Вызовы сервисов осуществляются в среде Metasonic в разделе Refinments для определенных состояний субъектов в процессе выполнения регламента. Чтобы обеспечить многократное использование сервисов, независящее ни от модели бизнес-объекта, ни от модели хранилища данных, ни от модели бизнес-процесса, авторами предложены следующие средства для настройки сервисов (настроечные таблицы):

- схема процесса;
- схема преобразований.

Смысл настроечных таблиц заключается в следующем. Схема процесса обеспечивает настройку сервиса на процесс и содержит сведения о том, какой бизнес-объект на каком шаге бизнес-процесса должен быть сохранен в хранилище. Схема преобразований содержит правила, по которым осуществляется преобразование сведений из экземпляра бизнес-объекта в экземпляр сущности хранилища данных. Установление связей между экземплярами сущностей в хранилище Alfresco осуществляется в соответствии с моделью хранилища.

Модель схемы реализации процесса представлена на рис. 2, где

- Step — описание шага регламента (IdProcess — идентификатор описания процесса, IdSubject — идентификатор описания субъекта, IdStep — идентификатор описания шага процесса);
- BO — описание бизнес-объекта (TypeBO — тип бизнес-объекта);
- Function — описание метода сервиса взаимодействия (NameFunction — имя метода);
- Entity — описание сущности (TypeEntity — тип сущности).

Модель схемы упомянутых выше преобразований представлена на рис. 3, где

- TypeBO — тип бизнес-объекта (NameAttribute — имя атрибута);
- TypeEntity — тип сущности (NameAttribute — наименование атрибута).

Конвертирование объекта Metasonic в сущность Alfresco. Результаты тестирования

Для получения ответа на вопрос о возможности хранения объектов Metasonic в хранилище Alfresco был проведен ряд тестов. По их результатам сделаны выводы о скорости выполнения конвертации, а также о возможных путях оптимизации данного процесса. Работа по проведению серии тестовых испытаний включала в себя развертывание инфраструктуры двух удаленных стендов (узлов) — Alfresco и Metasonic Suite. На сервере Metasonic Suite также была развернута локальная версия Alfresco.

В ходе тестовых экспериментов проводилось конвертирование десяти полей бизнес-объекта Requirement в поля сущности ServiceDescription. В каждом тесте создавалось 10, 100 и 200 однотипных объектов с передачей данных всех десяти полей. Затем рассчитывалось среднее время выполнения конвертации для одного объекта.

Вызов процедур Alfresco проводился с сервера Metasonic. При этом использовалась библиотека Apache Chemistry [11]. Время, которое затрачивалось на реализацию каждого из тестов, измерялось в миллисекундах. Рассматривалось два варианта тестирования. В первом варианте, который назван исходным, для каждого обращения к Alfresco создавался экземпляр соединения. Во втором варианте, который назван оптимизированным, использовалось одно и то же статическое соединение.

Два узла, на которых проводилось тестирование, представляют собой виртуальные серверы, параметры которых представлены в таблице.

На рис. 4 представлены результаты тестирования производительности для исходного и оптимизированного вариантов для случая, когда хранилище Alfresco установлено на том же сервере, что и Metasonic Suite. Для каждого из трех тестов показано по два результата: первый столбец — среднее время выполнения в исходном варианте; второйя столбец — среднее время выполнения в оптимизированном варианте.

Хорошо видно, что оптимизация позволяет ускорить процесс конвертирования объектов почти в 3 раза.

Параметр	Сервер Metasonic Suite	Сервер Alfresco
Операционная система	Windows 7 Домашняя расширенная x64	Windows Server 2008 Standard x32
Оперативная память	8 Гбайт	2 Гбайта
Процессор	AMD A8-3510MX APU 1.8 GHz 4 ядра	Intel Xeon X5660 1.8 GHz 2 процессора
Прикладное ПО	Metasonic Suite, вер. 4.0, Alfresco 4.0.e x64	Alfresco 4.0.e 32-bit

Эффект оптимизации еще более заметен, если протестировать тот же самый процесс, но с использованием хранилища Alfresco, расположенного на удаленном сервере. Во время тестирования сервер Metasonic Suite располагался в г. Уфа, а сервер с хранилищем Alfresco — в г. Москва (рис. 5). В этом случае оптимизация позволила сократить среднее время выполнения операции конвертирования в 6—15 раз. Следует также отметить, что обращение к удаленному серверу Alfresco, по сравнению с локальным увеличивает время в среднем в 4 раза.

Необходимо также отметить тот факт, что если все поля объекта конвертируются с помощью одного обращения к хранилищу Alfresco, то скорость обработки увеличивается еще в 4—5 раз.

Заключение

Предложенный субъектно-ориентированный подход к управлению стадиями этапа разработки в совокупности со способом ведения Реестра сервисов в специальном хранилище данных обладают следующими достоинствами:

- гибкость настройки процесса проектирования сервисов и разработки реализующих их компонентов на каждой стадии этапа разработки;

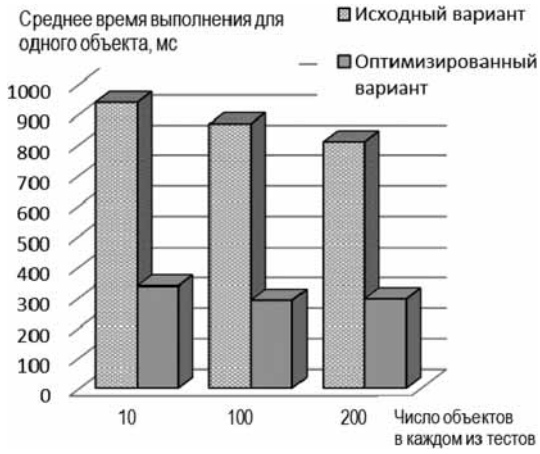

Рис. 4. Результаты тестов для локального сервера Alfresco

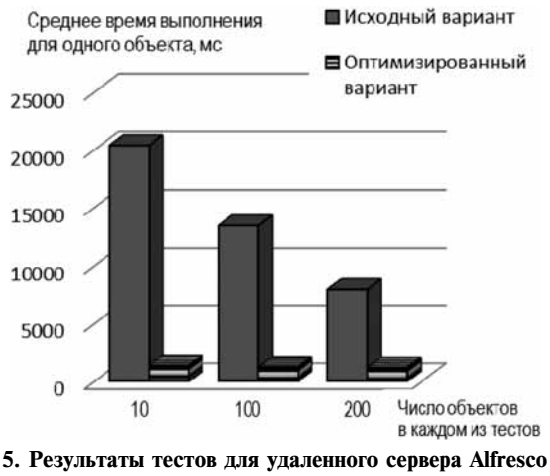

Рис. 5. Результаты тестов для удаленного сервера Alfresco

- простота реализации, основанная на вызовах web-сервисов.

Предложенный подход позволяет создать условия для повторного использования написанного программного кода в процессе создания новых приложений в парадигме SOA.

Существенным преимуществом предлагаемого в настоящей работе подхода является вовлечение в процесс наполнения Реестра представителей руководства организации. Им представляется возможность оперативно контролировать деятельности внутри компании по обновлению старых и внесению новых сервисов в Реестр за счет автоматизации бизнес-процесса и формирования ключевых показателей. Однако следует отметить, что недостатком предложенного способа является отсутствие автоматизированных средств синхронизации изменений моделей данных.

Дальнейшие исследования направлены на устранение указанного недостатка, а именно на создание интегрированных средств, обеспечивающих синхронизацию изменений моделей данных, описывающих бизнес-объекты. Такие модели используются в процессе разработки целевого программного продукта в качестве промежуточного хранилища данных и сущностей хранилища данных Реестра описаний сервисов и компонентов.

Данное исследование проводилось при финансовой поддержке Правительства Российской Федерации (Минобрнауки России) в рамках договора № 13.G25.31.0096 о "Создании высокотехнологичного производства кросс-платформенных систем обработки неструктурированной информации на основе свободного программного обеспечения для повышения эффективности управления инновационной деятельностью предприятия в современной России".

Список литературы

1. **Content** Management Interoperability Services (CMIS). OASIS Standard Specification. Version 1.0. 2010. URL: http://docs.oasis-open.org/cmis/CMIS/v1.0/os/cmis-spec-v1.0.pdf.

2. **Reference** Model for Service Oriented Architecture 1.0. OASIS Standard, 12 October 2006. URL: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf.

3. **Stephens R. T.** The Repository vs. The Registry. URL: http://www.information-management.com/news/1025672-1.html?zkPrintable=1&nopagination=1.

4. **Fleischmann A.** What Is S-BPM? // Communications in Computer and Information Science. 2010. Vol. 85. P. 85—106.

5. **Hype** Cycle for Business Process Management. Gartner. 25 July 2011. URL: http://www.adeptia.com/products/Hype_cycle_BPM_ 2011.pdf.

6. **Hufgard A., Gerhardt E.** Consolidating Business Processes as Exemplified in SAP ERP Systems // Communications in Computer and Information Science. 2011. Vol. 213. P. 155—171.

7. **Martin W.** Active Compliance Management with Subject-oriented Business Process Management. URL: http://www.metasonic.de/sites/default/files/editor/PDF/Metasonic_whitepaper_e_finsec.pdf.

8. **Busse C., Stein G., Fackelmeier B.** Metasonic Build Modeler Manual. Version: T9.0. 08/01/2011. URL: http://www.metasonic.de/en/downloads.

9. **Кусов А. А.** Проблемы интеграции корпоративных информационных систем // Управление экономическими системами: электронный научный журнал. 2011. № 4 (28). URL: http://uecs.ru/uecs-28-282011?start=20.

10. **Web** Services Base Notification 1.3 (WS-BaseNotification). OASIS Standard. 1 October 2006. URL: http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf.

11. **Apache** Chemistry. Apache Software Foundation. URL: http://incubator.apache.org/projects/chemistry.html.

12. **Cei U., Lucidi P.** Alfresco 3 Web Services. Birmingham: Packt Publishing Limited, 2010.

УДК 621.039.51 + 004.272.26

О. О. Казьмин, мл. науч. сотр., НИИ механики МГУ им. М. В. Ломоносова,
И. А. Капацкая, инженер, ОАО ДЖЭТ («Дженерал Энерджи Технолоджиз»),
С. А. Карпов, инженер, ОАО ВНИИАЭС,
e-mail: nutok@yandex.ru

Моделирование нейтронно-физических процессов активной зоны реактора АЭС в реальном времени с применением параллельных вычислений

Представлены результаты адаптации существующих алгоритмов и программных средств моделирования нейтронно-физических процессов активной зоны реактора атомной электростанции для работы на суперкомпьютерных платформах. Проанализирована эффективность использующейся реализации модели в целях выявления ее наиболее ресурсозатратных частей. Исследованы отдельные программные компоненты модели и связи между ними для определения возможности распараллеливания. Разработана и реализована распределенная версия алгоритмов моделирования. Реализованные методы апробированы, исследована их эффективность и предложены направления дальнейшей оптимизации.

Ключевые слова: *распараллеливание программ, эффективность распараллеливания, нейтронная физика, моделирование нейтронно-физических процессов*

Введение

Задача обеспечения безопасности существующих и проектируемых атомных электростанций (АЭС) становится еще более актуальной в связи с рядом чрезвычайных ситуаций, произошедших на АЭС в последние десятилетия. Принимая во внимание цену каждой ошибки при эксплуатации АЭС и отсутствие реальной альтернативы атомной энергетике, следует самое серьезное внимание уделить моделированию различных внештатных ситуаций и их вероятных последствий, в первую очередь чрезвычайного характера.

В настоящее время во Всероссийском научно-исследовательском институте атомных электростанций (ВНИИАЭС) создается суперкомпьютерная полномасштабная модель атомной электростанции, именуемая "Виртуальной АЭС с водо-водяным энергетическим реактором (ВВЭР)", с помощью которой можно серьезно продвинуться в решении следующих задач:

- расчет штатных режимов эксплуатации существующих и проектируемых АЭС;
- качественная подготовка персонала, в том числе с помощью симуляторов управления АЭС в режиме реального времени;
- разработка оптимальных сценариев устранения и минимизации последствий внештатных ситуаций на существующих и проектируемых АЭС;
- упреждающее моделирование в процессе принятия решений по выходу из опасных внештатных ситуаций.

Полномасштабная модель создается на основе прошедших апробацию и хорошо зарекомендовавших себя на практике алгоритмов и их программных реализаций на ЭВМ традиционной последовательной архитектуры. Среди них немаловажную роль играет нейтронно-физический код ТРЕК, предназначенный

для моделирования процессов в активной зоне реактора. Такой программный код моделирует изменения пространственного распределения плотности потока нейтронов, энерговыделения, а также остаточного энерговыделения в активной зоне. В настоящее время он повсеместно используется на практике. Однако повышение точности расчетов, увеличение расчетной сетки и необходимость упреждающего моделирования требуют высокопроизводительной реализации таких расчетных кодов. Для этого в настоящее время ведется их адаптация к современным суперкомпьютерным вычислительным установкам, особенностью которых является массовый параллелизм и распределенная по узлам оперативная память. Компактные варианты подобных систем могут в перспективе позволить оснастить высокоточными моделирующими программными комплексами все заинтересованные организации, в первую очередь ситуационные и учебные центры.

Однако современным суперкомпьютерам присущи некоторые ограничения, которые делают распараллеливание данного программного комплекса достаточно сложной задачей. В настоящей статье рассмотрены следующие вопросы, связанные с его переносом на суперЭВМ:

- проведен анализ компонентов системы, занимающих наибольшее количество времени, для определения возможности их распараллеливания;
- представлена использующаяся в настоящее время схема распараллеливания;
- проведен анализ эффективности и выявлены механизмы, ограничивающие ее производительность;
- предложены перспективные методы оптимизации расчетов с учетом целевой платформы, механизмов распараллеливания и особенностей модели.

1. Общее описание модели

Существует большое число алгоритмов моделирования нейтронно-физических процессов в активной зоне атомного реактора, программные реализации которых могут существенно отличаться друг от друга. При адаптации программной модели к суперкомпьютерным вычислительным установкам следует учитывать особенности как алгоритма, так и его программной реализации. По этой причине прежде чем переходить к описанию распараллеливания, необходимо дать описание использующихся алгоритмов и особенностей их программной реализации. В данном разделе представлена методика нейтронно-физического расчета и особенности структуры ее реализации, которые оказывают существенное влияние на адаптацию расчетов к суперкомпьютерным установкам.

1.1. Методика нейтронно-физического расчета

В нейтронно-физическом расчете используют трехмерное, двухгрупповое приближение расчета. Решается уравнение переноса нейтронов в диффузионном приближении с разделением переменных с шестью группами запаздывающих нейтронов. Амплитуду изменения мощности определяют в точечном приближении.

При расчете остаточного энерговыделения учитывают 11 групп осколков деления, распределенных по объему активной зоны (АЗ). Из продуктов деления рассматривают только ксенон, самарий, йод и прометий, так как существенное влияние на временное поведение реактора оказывают только эти элементы. Остальные продукты деления учтены в сечениях, рассчитанных для определенного момента времени. Зависимость среднего времени жизни нейтрона от плотности воды и выгорания незначительна, поэтому при расчетах используют усредненное по реактору время жизни нейтрона.

Программа обеспечивает расчет объемного распределения энерговыделения в стационарных и нестационарных режимах работы реактора до 10 раз в секунду, в режиме реального времени.

Время выгорания топлива не моделируется. Поведение реактора в различные моменты времени моделируется через использование сечений, зависящих от выгорания топлива.

Исходная система уравнений, отвечающая диффузионной двухгрупповой модели, выглядит следующим образом:

$$
\begin{cases}
\dfrac{1}{v^{(1)}}\dfrac{\partial \varphi^{(1)}(\mathbf{r}, t)}{\partial t} = \nabla D^{(1)} \nabla \varphi^{(1)}(\mathbf{r}, t) + (1-\beta) \times \\
\times \left[v_f^{(1)} \Sigma_f^{(1)}(\mathbf{r}) \varphi^{(1)}(\mathbf{r}, t) + v_f^{(2)} \Sigma_f^{(2)}(\mathbf{r}) \varphi^{(2)}(\mathbf{r}, t) \right] - \\
- \Sigma_a^{(1)}(\mathbf{r}) \varphi^{(1)}(\mathbf{r}, t) - \Sigma_d^{(1 \to 2)}(\mathbf{r}) \varphi^{(1)}(\mathbf{r}, t) + \\
+ \sum_{m=1}^{6} \lambda_m C_m(\mathbf{r}, t) + S^{(1)}(\mathbf{r}, t), \\
\dfrac{1}{v^{(2)}}\dfrac{\partial \varphi^{(2)}(\mathbf{r}, t)}{\partial t} = \nabla D^{(2)} \nabla \varphi^{(2)}(\mathbf{r}, t) - \\
- \Sigma_a^{(2)}(\mathbf{r}) \varphi^{(2)}(\mathbf{r}, t) + \Sigma_d^{(1 \to 2)}(\mathbf{r}) \varphi^{(1)}(\mathbf{r}, t) + \\
+ S^{(2)}(\mathbf{r}, t), \\
\dfrac{\partial C_m(\mathbf{r}, t)}{\partial t} = \beta_m [v_f^{(1)} \Sigma_f^{(1)}(\mathbf{r}) \varphi^{(1)}(\mathbf{r}, t) + \\
+ v_f^{(2)} \Sigma_f^{(2)}(\mathbf{r}) \varphi^{(2)}(\mathbf{r}, t)] - \lambda_m C_m(\mathbf{r}, t),
\end{cases}
\tag{1}
$$

где $v^{(g)}$ — модуль скорости нейтронов[1]; $\varphi^{(g)}(\mathbf{r}, t)$ — функция потока нейтронов; $\mathbf{r}$ — радиус-вектор, задающий точку в пространстве; t — время; $D^{(g)}$ — коэффициент диффузии; β — эффективная доля запаздывающих нейтронов; β_m — доля запаздывающих ней-

[1] Здесь и далее верхний индекс (g) означает, что значение определяется для группы нейтронов с номером g. В данном случае он может принимать значения только 1 и 2.

тронов группы m; $\upsilon_f^{(g)}$ — число нейтронов деления; $\Sigma_f^{(g)}$ — макросечение деления; $\Sigma_a^{(g)}(\mathbf{r})$ — макросечение поглощения; $\Sigma_d^{(g)}(\mathbf{r})$ — макросечение увода нейтронов; $\Sigma_d^{(1\to 2)}(\mathbf{r})$ — макросечение замедления нейтронов; λ_m — постоянная распада ядер группы запаздывающих нейтронов m; $C_m(\mathbf{r}, t)$ — концентрация ядер эмиттеров группы запаздывающих нейтронов m; $S^{(g)}(\mathbf{r}, t)$ — источники нейтронов.

Функция потока нейтронов представляется как $\varphi^{(g)}(\mathbf{r}, t) = T(t)\Phi^{(g)}(\mathbf{r}, t)$, где $T(t)$ — амплитудная функция; $\Phi^{(g)}(\mathbf{r}, t)$ — функция пространственного распределения, предполагается слабое ее изменение по отношению к $T(t)$.

Особую сложность представляет алгоритм нахождения функции пространственного распределения нейтронов $\Phi^{(g)}(\mathbf{r}, t)$. Остановимся на нем подробнее. После интегрирования первых двух уравнений системы (1) по объему ячейки V_k (k — номер ячейки), запишем их в конечно-разностной форме:

$$\begin{cases} \dfrac{1}{\upsilon^{(1)}}\dfrac{\Phi_{n+1}^{(1)}}{\Delta t} = \dfrac{1}{\upsilon^{(1)}}\dfrac{\Phi_n^{(1)}}{\Delta t} + \dfrac{1}{V_k}\int\limits_{V_k}\nabla D_{n+1}^{(1)}\nabla\Phi_{n+1}^{(1)}(\mathbf{r})dV + \\ + \dfrac{(1-\beta)\left[\upsilon_f\Sigma_{f,\,n+1}^{(1)}\Phi_{n+1}^{(1)} + \upsilon_f\Sigma_{f,\,n+1}^{(2)}\Phi_{n+1}^{(2)}\right]}{\tilde{K}} - \\ - \left[\Sigma_{a,\,n+1}^{(1)} + \dfrac{1}{\upsilon^{(1)}}\dfrac{1}{T_{n+1}}\dfrac{dT}{dt}\right]\Phi_{n+1}^{(1)} - \\ - \Sigma_d^{(1\to 2)}\Phi_{n+1}^{(1)} + \\ + \dfrac{1}{T_{n+1}}\sum\limits_{m=1}^{6}\lambda_m C_{m,\,n+1} + \dfrac{1}{T_{n+1}}S^{(1)}, \\ \dfrac{1}{\upsilon^{(2)}}\dfrac{\Phi_{n+1}^{(2)}}{\Delta t} = \dfrac{1}{\upsilon^{(2)}}\dfrac{\Phi_n^{(2)}}{\Delta t} + \dfrac{1}{V_k}\int\limits_{V_k}\nabla D_{n+1}^{(2)}\nabla\Phi_{n+1}^{(2)}(\mathbf{r})dV - \\ - \left[\Sigma_{a,\,n+1}^{(2)} + \dfrac{1}{\upsilon^{(2)}}\dfrac{1}{T_{n+1}}\dfrac{dT}{dt}\right]\Phi_{n+1}^{(2)} + \\ + \Sigma_d^{(1\to 2)}\Phi_{n+1}^{(1)} + \dfrac{1}{T_{n+1}}S^{(2)}. \end{cases} \quad (2)$$

Здесь и далее нижний индекс $n + 1$ или n означает шаг по времени, на котором берется соответствующий параметр.

Из этих уравнений можно определить искомые средние потоки нейтронов $\Phi_{n+1}^{(g)}$, для каждой области, если считать известными перетечки нейтронов между расчетными элементами — нодами. Для этого необходимо определить следующие значения:

$$\int\limits_{V_k}\nabla D_{n+1}^{(g)}\nabla\Phi_{n+1}^{(g)}(\mathbf{r})dV.$$

Здесь есть неизвестные распределения $\Phi_{n+1}^{(g)}$, поэтому искусственно принимаем, что:

$$\int\limits_{V_k}\left[\nabla D_{n+1}^{(g)}\nabla\Phi_{n+1}^{(g)}(\mathbf{r})\right]_k dV = \\ = -\sum_{l\in N(k)}\chi_{lk}^{(g)}\left(\Phi_{k,\,n+1}^{(g)} - \Phi_{l,\,n+1}^{(g)}\right), \quad (3)$$

где $\Phi_{l,\,n+1}^{(g)}$ означает $\Phi_{n+1}^{(g)}$ в ячейке с номером l.

Под $N(k)$ понимаем множество ячеек, которые являются соседними для ноды с номером k. При гексагональной расчетной сетке таких соседних ячеек будет восемь — шесть радиальных и две аксиальных. Более подробно использующаяся расчетная сетка описана в разд. 2.

$\chi_{lk}^{(g)}$ является константой, определяемой в зависимости от способа моделирования и типа реактора. В данном случае (одна расчетная точка на тепловыделяющую сборку (ТВС) и типе реактора ВВЭР) оына равна

$$\chi_{lk}^{(g)} = \frac{4}{3p^2}\frac{D_k^{(g)}D_{lk}^{(g)}}{D_k^{(g)} + D_{lk}^{(g)}} \quad \text{для радиальных перетечек;}$$

$$\chi_{lk}^{(g)} = \frac{2}{\Delta h_z}\frac{D_k^{(g)}D_{lk}^{(g)}}{D_k^{(g)} + D_{lk}^{(g)}} \quad \text{для аксиальных перетечек,}$$

где p — шаг гексагональной решетки; Δh_z — размер «под ключ»; D_k — коэффициент диффузии в ячейке k; D_{lk} — коэффициент диффузии между ячейками l и k.

Неточность этого интегрального баланса нейтронов в стационарном состоянии, обусловленная приближениями расчетной схемы и подготовки констант, компенсируется введением коэффициента $\tilde{K}$. Таким образом, выбранное определенное стационарное состояние реактора объявляется точно критичным.

Введем обозначения:

$$Q_k^{(1)} = \frac{1}{\upsilon^{(1)}}\frac{\Phi_n^{(1)}}{\Delta t} + \sum_{l\in N(k)}\chi_{lk}^{(1)}\Phi_{k,\,n+1}^{(1)} + \\ + \frac{(1-\beta)\upsilon_f\Sigma_{f,\,n+1}^{(2)}\Phi_{n+1}^{(2)}}{\tilde{K}} + \\ + \frac{1}{T_{n+1}}\sum_{m=1}^{6}\lambda_m C_{m,\,n+1} + \frac{1}{T_{n+1}}S^{(1)};$$

$$Q_k^{(2)} = \frac{1}{\upsilon^{(2)}}\frac{\Phi_n^{(2)}}{\Delta t} + \sum_{l\in N(k)}\chi_{lk}^{(2)}\Phi_{k,\,n+1}^{(2)} + \Sigma_d^{(1\to 2)}\Phi_{n+1}^{(1)}; \quad (4)$$

$$G_k^{(1)} = \frac{1}{v^{(1)}\Delta t} + \left[\Sigma_{a,\,n+1}^{(1)} + \frac{1}{v^{(1)}}\frac{1}{T_{n+1}}\frac{dT}{dt}\right] + \Sigma_d^{(1\to 2)} +$$

$$+ \sum_{l\in N(k)} \chi_{lk}^{(1)} - \frac{(1-\beta)\upsilon_f \Sigma_{f,\,n+1}^{(1)}}{\tilde{K}};$$

$$G_k^{(2)} = \frac{1}{v^{(2)}}\left[\frac{1}{\Delta t} + \frac{1}{T}\frac{dT}{dt}\right] + \Sigma_{a,\,n+1}^{(2)} + \sum_{l\in N(k)} \chi_{lk}^{(2)}.$$

Применяя (3) и (4) к системе (2), получим средние потоки нейтронов в нодах — $\Phi_{n+1}^{(g)}$. Процесс расчета средних потоков нейтронов должен происходить итерационно, так как для того чтобы найти поток нейтронов в какой-либо ноде необходимо знать потоки во всех нодах окружения. По этой причине применяется метод верхней релаксации (с использованием фактора ускорения ω_g):

$$\Phi_{k,\,n+1}^{(1)} = (1-\omega_1)\Phi_{k,\,n}^{(1)} + \omega_1 \frac{Q_{k,\,n+1}^{(1)}}{G_{k,\,n+1}^{(1)}};$$

$$\Phi_{k,\,n+1}^{(2)} = (1-\omega_2)\Phi_{k,\,n}^{(2)} + \omega_2 \frac{Q_{k,\,n+1}^{(2)}}{G_{k,\,n+1}^{(2)}}.$$

Как правило значения ω_g лежат в пределах $1{,}0 < \omega_g < 1{,}4$.

Для подготовки двухгруппового константного обеспечения используется многогрупповая программа GETERA [1], учитывающая влияние окружающих каналов. Для обеспечения высокого быстродействия расчета констант во время моделирования нестационарных процессов нейтронно-физические константы аппроксимируются полиномами. Вид полиномов оптимизируется для получения максимального быстродействия при заданной точности расчета. Во всех режимах работы блока используется единая система полиномов:

$$\Sigma(p_1, p_2, \ldots, p_K) =$$

$$= \sum_{i=1}^{N} \alpha_i \left(p_1 - p_1^{\min}\right)^{\mathbf{n}_i^1}\left(p_2 - p_2^{\min}\right)^{\mathbf{n}_i^2}\ldots\left(p_K - p_K^{\min}\right)^{\mathbf{n}_i^K},$$

где K — число параметров; N — число функций в полиноме; p_i — значение параметра; $p_i^{\min}$ — минимальное значение параметра, используемое в модели; α_i — i-й коэффициент полинома; $\mathbf{n}_i^K$ — матрица степеней полиномов размерностью (K, N).

Полиномиальные зависимости от параметров строятся для следующих величин:

- сечение деления нейтронов в двух группах;
- сечения поглощения в двух группах;
- сечение увода в первой энергетической группе;
- коэффициенты диффузии в двух группах;
- микросечения поглощения нейтронов ксеноном и самарием.

Аргументами полиномов в модели являются:

- плотность теплоносителя;
- температура воды;
- температура топлива;
- концентрация бора;
- глубина выгорания.

Точность восстановления констант определяется качеством выбора функций свертки, а быстродействие и объем хранимой информации — числом функций разложения. Для данного метода можно отметить отсутствие систематических погрешностей, присущих табличному методу. Метод обеспечивает достижение заданной точности в условиях расчета в реальном времени.

1.2. Структура программной модели

Нейтронно-физический расчетный код ТРЕК реализован в рамках базовой среды моделирования S3, которая допускает использование MPI-процессами всех вычислительных ядер суперкомпьютера. Запуск расчетного кода осуществляется в S3 в виде запуска контрольных модулей с определенной частотой, в которых уже вызываются расчетные модули. Все модули системы реализованы на языке Фортран. Описание всех модулей нейтронно-физического кода представлено в табл. 1.

Таблица 1

Структура нейтронно-физического кода ТРЕК

Число вызовов в секунду	Контрольные модули	Расчетные модули	Описание
10	cr4dy1	scrd01	Интерфейс с системами
		scrd02, в состав которого входят:	Нейтронно-физические константы для ТВС
		yscrstk	Полиномы для ТВС
		yscrsor	Полиномы для системы управления и защиты (СУЗ)
		scrd04	Реактивность. Амплитудная функция потока нейтронов
		scrd05	Остаточное тепловыделение
10	cr4dy2	scrd03	Расчет пространственной функции потока нейтронов
10	cr1dy1	scrd06	Расчет коэффициентов-утечек для пространственной функции потока нейтронов
1	cr1dy2	scrd07	Расчет отравления

Расчетные модули комплекса представляют собой относительно самостоятельные расчетные единицы. Как следствие, в большинстве своем они связаны друг с другом довольно небольшим объемом выходных данных. По указанной причине было принято решение проводить модификацию кода программного комплекса на уровне модулей. Архитектура программной реализации модели представлена на рис. 1.

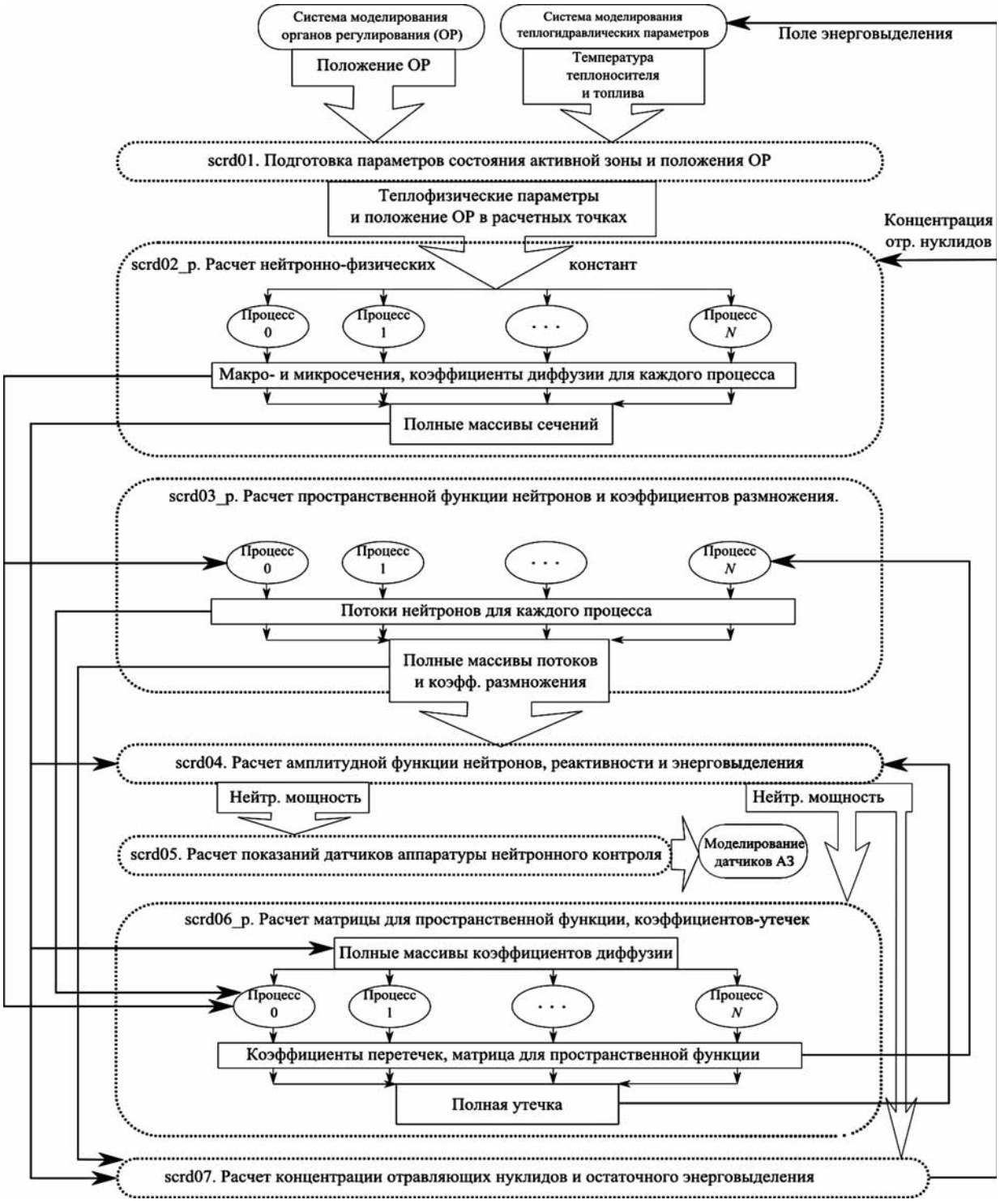

Рис. 1. Схема работы нейтронно-физического кода

1.3. Результаты профилировки расчетного кода ТРЕК

В целях выявления наиболее затратных компонентов программного комплекса была проведена профилировка его работы. Анализ проводился по результатам моделирования работы реактора в течение трех минут. Результаты представлены в табл. 2.

Таблица 2

Результаты профилировки расчетного кода ТРЕК

Контрольные модули	Подпрограммы	Затраченное время	
		Абсолютное, с	Относительное, %
cr4dy1	scrd01	Менее 0,01	Менее 0,01
	scrd02	0,35	0,47
	yscrstk	20,51	27,47
	yscrsor	23,21	31,10
	scrd04	0,05	0,07
	scrd05	0,13	0,17
cr4dy2	scrd03	25,78	34,54
cr1dy1	scrd06	4,24	5,68
cr1dy2	scrd07	0,37	0,50

Нетрудно заметить, что три модуля занимают в общей сложности более 90 % времени работы всей системы. Причем модули расчета полиномов для ТВС и СУЗ (yscrstk и yscrsor соответственно) являются подпрограммами модуля расчета нейтронно-физических констант (scrd02). На основе этих данных было принято решение ограничиться модификацией только указанных компонентов комплекса.

Однако модуль расчета матрицы для пространственной функции и коэффициентов-утечек (scrd06) связан с расчетом пространственной функции (scrd03) очень большим количеством данных. Для вычисления пространственной функции необходимы коэффициенты утечек в расчетной области и значения знаменателя в уравнении, которые вычисляются модулем scrd06. В условиях работы в реальном времени передача такого большого объема данных между процессорами сильно замедлит работу системы. Для обеспечения передачи указанных параметров через оперативную память было принято решение разработать параллельную версию данного компонента модели, несмотря на то, что он занимает всего 5 % времени работы всего комплекса.

2. Распараллеливание расчета нейтронно-физических констант

Расчет нейтронно-физических констант основывается на сеточном методе вычислений. Расчетная сетка является цилиндром высотой в 15 точек и с 163 точками в основании. Основание поделено на семь областей, а именно — центральную в виде круга и шесть секторов. По высоте расчетная сетка поделена на пять групп по три слоя в каждой. Схематичное изображение расчетной сетки представлено на рис. 2, где цифрами обозначено число точек расчетной сетки в области. На рис. 3 представлена расчетная сетка в горизонтальной проекции. В каждой точке указанной сетки выполняется расчет коэффициентов для органов регулирования систем управления защиты и для тепловыделяющей сборки, а затем пересчет по ним констант во всей области. Особенностью данных вычислений в плане возможности распараллеливания является то, что для каждой точки значения полиномов зависят только от теплогидравлических параметров в самой этой точке и не зависят от аналогичных параметров в других точках. Отсюда следует, что расчет в каждой области расчетной сетки может осуществляться независимо друг от друга, а следовательно, может быть распараллелен без дополнительных обменов данными, кроме рассылки начальных параметров расчета.

Были разработаны две схемы, использующие различные методы распараллеливания — статический и квазистатический для произвольного числа процессов. Данный подход обусловлен различной спецификой этих методов. При статическом распараллеливании снижаются затраты на накладные расходы за счет того, что область расчетов для каждого процесса определена на этапе реализации алгоритма. Однако при использовании большого числа процессов эффективность распараллеливания будет уменьшаться, как следствие, придется выбирать варианты кода, оптимальные с точки зрения отношения затраченные ресурсы/производительность. Квазистатическое распараллеливание, в отличие от статического, позволяет подобрать такое оптимальное число процессов. Далее представлены схемы предлагаемых методов.

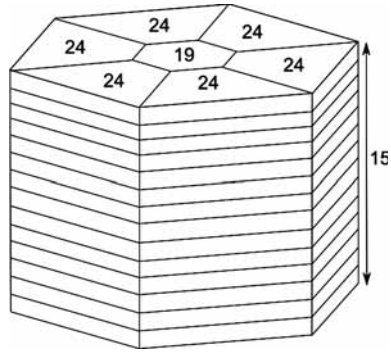

Рис. 2. Разбиение АЗ реактора

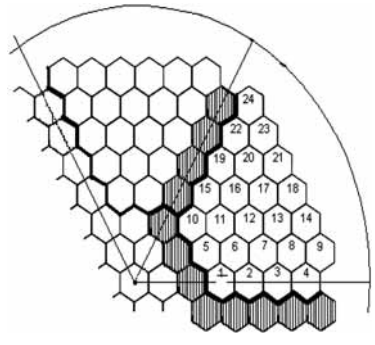

Рис. 3. Разбиение АЗ реактора в горизонтальной проекции

Для использования произвольного количества процессов вводится одномерная параметризация областей расчетной сетки. Каждая область нумеруется числом от 1 до 105, что сводит распределение вычислительной работы между процессорными элементами к разбиению множества 105 целых чисел на определенное число подмножеств. На данный момент используется схема, в которой процесс с рангом N вычисляет полиномы в областях с номерами от $start_N$ до end_N, которые рассчитываются по следующим формулам:

$$start_N = \left[\frac{105}{size}\right](N-1) + a_N + 1,$$

где $size$ — число всех процессов;

$$end_N = start_N + \left[\frac{105}{size}\right] + b_N - 1;$$

$$a_N = \begin{cases} N-1, & \text{если } N \leqslant \text{mod}(105, size); \\ \text{mod}(105, size), & \text{если } N > \text{mod}(105, size); \end{cases}$$

$$b_N = \begin{cases} 1, & \text{если } N \leqslant \text{mod}(105, size); \\ 0, & \text{если } N > \text{mod}(105, size). \end{cases}$$

Указанный метод гарантирует максимально равномерное распределение вычислительной нагрузки между произвольным числом узлов, однако он уступает по накладным расходам статическому распараллеливанию на несколько арифметических операций и одну передачу данных (параметры отображения трехмерной параметризации в одномерную). Следует также отметить тот факт, что подобный подход не зависит от способа нумерации расчетных областей. Следовательно, возможна оптимизация алгоритма с учетом архитектуры целевой вычислительной установки. В настоящее время используется одномерная параметризация, представленная на рис. 4.

Схема со статическим распараллеливанием является схемой расчета с минимальным количеством накладных расходов, которые обусловлены разнесением отдельных вычислительных процессов, которые появляются в ходе решения задачи, на несколько процессоров. Данная схема была реализована для 7 и 35 процессоров. При запуске на 7 процессорах каждый процесс считает области расчетной сетки, которым соответствуют одинаковые горизонтальные проекции. При запуске на 35 процессорах указанные области дополнительно делятся на пять об-

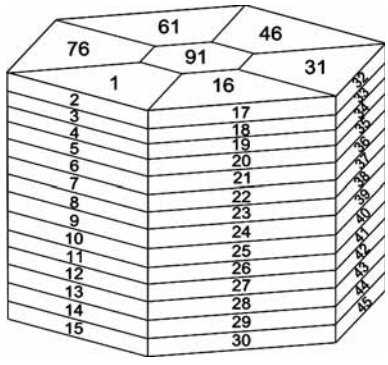

Рис. 4. Одномерная параметризация расчетной области

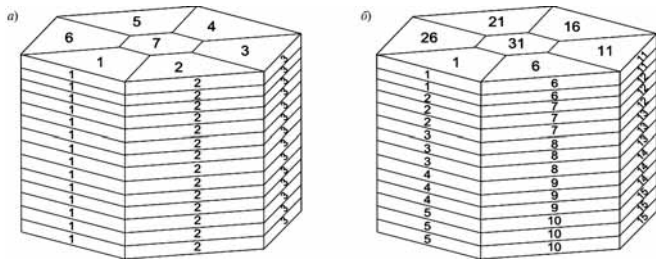

Рис. 5. Распределение расчетных областей по процессам:
а — для 7 процессоров; *б* — для 35 процессоров

ластей по вертикали. Распределение расчетных областей по процессорам схематично изображено на рис. 5, где числами указаны номера процессоров, рассчитывающих константы в точках данной области.

В целом необходимо отметить, что структура алгоритма и реализующего его программного модуля расчета нейтронно-физических констант имеют высокий внутренний параллелизм. Как следствие, этот модуль распараллеливается очень хорошо. Анализ эффективности реализации представленного метода распараллеливания можно найти в разд. 5.

3. Распараллеливание расчета пространственной функции

Расчет пространственной функции проводится методом итерации источников, суть которого изложена в работе [2]. Используется та же расчетная сетка, что и в расчете нейтронно-физических констант (см. разд. 2). В указанном методе поток нейтронов в точке определяется значениями потока в этой и восьми соседних точках (для гексагональной геометрии), а также источником нейтронов в этой точке. Источник нейтронов в точке определяется суммой потоков нейтронов по группам в этой точке. Потоки и источники нейтронов определяются внутри итерационного цикла. Итерации прекращаются, когда эффективный коэффициент размножения нейтронов перестает меняться с заданной точностью. На каждой итерации проводится нормировка источника нейтронов на суммарное энерговыделение по зоне. Таким образом, чтобы приступить к следующей итерации, вначале нужно узнать локальные потоки и источники нейтронов по всей активной зоне.

Расчет пространственной функции на настоящее время распараллелен статически для 7 и 35 процессоров, с использованием разбиения расчетной области, описанного в разд. 2. В отличие от модуля расчета нейтронно-физических констант, в модуле вычисления пространственной функции присутствуют зависимости по данным между гранулами параллелизма. По этой причине число пересылок увеличивается и, соответственно, эффективность распараллеливания снижается.

На рис. 6 схематично представлен алгоритм параллельного вычисления пространственной функции. На схеме можно заметить, что во время выполнения внутренних итераций не используются данные, получаемые другими процессами во время выполнения итераций внешних. Следовательно, весь цикл внутренних итераций может выполняться на удаленном вычислительном узле без дополнительных обменов данными. Однако для

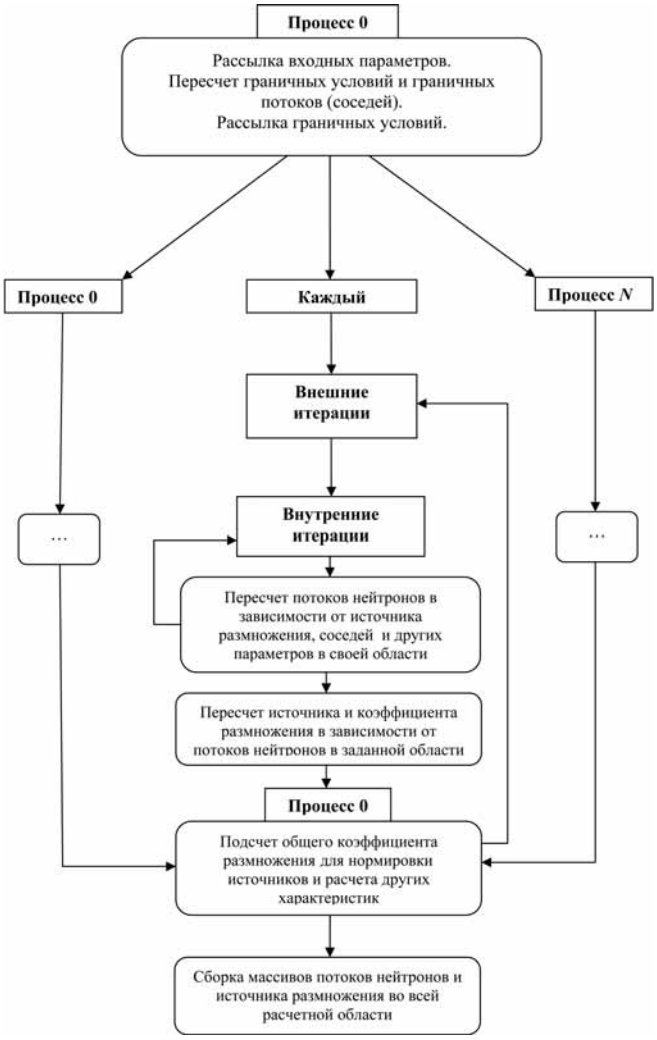

Рис. 6. Алгоритм вычисления пространственной функции

расчета потоков нейтронов требуются данные о всей расчетной области, полученные на прошлой внешней итерации, например, общий коэффициент размножения и поток нейтронов в соседних ячейках. По этой причине при инициализации новой внешней итерации реализован механизм коллективного обмена данными и синхронизация процессов.

Описанный выше алгоритм содержит коллективные обмены данными о потоках нейтронов в соседних ячейках. Они были реализованы по схеме "один-всем" и "все-одному". Указанный подход позволяет без существенных издержек поддерживать механизмы сохранения состояния расчета, которые накладывают условие "все данные при выходе из модуля должны содержаться в памяти нулевого процесса".

4. Интеграция параллельных модулей в расчетный код

В предыдущих разделах описана схема распараллеливания компонентов системы, которые занимают в сумме более 90 % времени работы расчетного кода. Однако в нем также присутствуют модули, работающие последовательно на главном процессоре. По этой причине возникает задача "стыковки" параллельных и последовательных частей расчета.

На рис. 1 (подразд. 1.2) можно видеть, что модуль расчета нейтронно-физических констант предоставляет данные для модулей, которые не подвергались модификации. В частности, в модуле scrd02 рассчитываются полные массивы сечений, которые используются при вычислении амплитудной функции нейтронов, реактивности и энерговыделения (scrd04, scrd07). Аналогичные зависимости можно отметить и у расчета пространственной функции. В силу этих обстоятельств, при завершении параллельного расчета необходим сбор указанных данных в памяти главного процесса, что несколько снижает эффективность распараллеливания.

Подобных издержек можно избежать, разработав параллельные версии для всех модулей системы. В таком случае большая часть данных будет распределена между процессами и необходимость частых пересылок исчезнет. Однако этому препятствуют несколько факторов. Во-первых, алгоритмы не всех компонентов комплекса имеют внутренний параллелизм и могут быть распараллелены. Во-вторых, нейтронно-физический расчетный код ТРЕК реализован в рамках базовой среды моделирования S3, которая накладывает некоторые ограничения на запускаемые расчеты. В рассматриваемом программном комплексе присутствует механизм сохранения и восстановления текущего состояния расчетов, что приводит к необходимости в определенные моменты собирать все параметры расчета на главном процессоре. В-третьих, данная задача достаточно трудоемка, что в совокупности со сложно прогнозируемым результатом делает ее решение нецелесообразным в рамках данного исследования.

5. Оценка производительности расчета нейтронно-физических констант

Как было обозначено в разд. 2, расчет нейтронно-физических констант имеет высокий внутренний параллелизм и распараллеливается с высокой эффективностью. Под эффективностью распараллеливания здесь и далее понимается отношение минимально возможного времени вычисления на заданном числе процессоров к полученному времени расчета. Результаты временных раскладок это подтверждают. Была проведена промежуточная оценка времени выполнения модели с распараллеленным модулем scrd02. В качестве объекта тестирования был принят процесс моделирования минуты работы реактора. Тестирование проводилось на сервере mpidev ВНИИАЭС. Особенностью данной установки является то, что она использует общую память. Ее наличие сильно ускоряет пересылки данных. Результаты тестирования и их сравнение с идеальным распараллеливанием представлены на рис. 7. Под идеальным распараллеливанием понимается распараллеливание без каких-либо накладных расходов, а именно — ускоряющее расчет в N раз, если N — число процессов. Для оценки времени использовался профилировщик gprof.

Представленные на рис. 7 данные указывают на эффективность распараллеливания при использовании не-

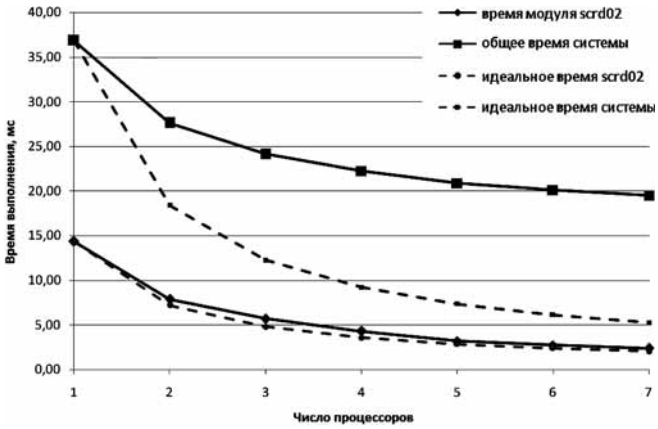

Рис. 7. Промежуточное тестирование параллельной версии расчета нейтронно-физических констант

Таблица 3

Время счета контрольного модуля cr4dy1

Время выполнения последовательного расчета, мс	Время выполнения на 7 процессорах, мс		Время выполнения на 35 процессорах, мс	
	Главный процесс	Второстепенный процесс	Главный процесс	Второстепенный процесс
17,92	3,43	3,27	3,45	1,39

большого числа процессоров. При работе на семи и меньше процессах эффективность распараллеливания колеблется между 80 и 90 %. Однако дальнейшие измерения на целевой универсальной компактной Супер-ЭВМ производства ФГУП "РФЯЦ-ВНИИЭФ" [3] выявили значительное уменьшение эффективности распараллеливания при увеличении числа процессов.

Измерения проводились при помощи встроенного механизма S3 показа временной статистики, который демонстрирует время работы только контрольных модулей. Однако модуль cr4dy1, в котором вызывается расчет нейтронно-физических констант, содержит модули, работающие последовательно. В силу этого обстоятельства замеры проводились не только в главном процессе, но и в одном второстепенном. Результаты времени счета второстепенного процесса могут быть несколько занижены в силу того, что во всех распараллеленных модулях присутствуют последовательные участки кода.

Результаты, представленные в табл. 3, отражают среднее время выполнения модуля cr4dy1 при моделировании трех минут работы реактора. В табл. 3 представлены результаты работы контрольного модуля cr4dy1 при моделировании в 1, 7 и 35 процессах. Результаты счета при параллельном моделировании представлены как на главном процессоре, так и на второстепенном.

Таблица 4

Модификация контрольного модуля cr4dy

Контрольные модули	Вызываемые расчетные модули	Описание
cr4dy1_p1		Рассылка начальных входных теплофизических параметров для модуля scrd02p
cr4dy1_p2	scrd01	Последовательный модуль интерфейса с системами (расчет идет только на главном процессе)
	scrd02p (scrd02p_1)	Расчет нейтронно-физических констант для ТВС (без обмена данными между процессами)
cr4dy1_p3		Сбор подсчитанных данных на главном процессе
cr4dy1_p4	scrd04	Последовательный модуль расчета реактивности и амплитудной функции потоков нейтронов
	scrd05	Последовательный модуль расчета остаточного тепловыделения

Время работы распараллеленной модели существенно сокращается по сравнению с временем последовательного расчета, однако полученное при 7 процессах ускорение сохраняется и при 35. Для более подробного анализа производительности было принято решение отделить сам параллельный расчет и пересылки данных. Вследствие этого контрольный модуль cr4dy1 был разделен на четыре других контрольных модуля, описание которых представлено в табл. 4.

Измерение времени выполнения всех указанных частей проводилось по тому же принципу (три минуты работы реактора, команда sho t). Результаты измерений приведены в табл. 5.

Исходя из данных, представленных в табл. 5, можно утверждать, что резкое снижение эффективности распараллеливания обусловлено все большей долей времени, затраченного на пересылки данных. Если при расчете в 7 процессах она составляла 20 % времени расчета, то в 35 — уже 82 %. В свою очередь, скорость параллельных вычислений без передачи данных увеличивается более чем в 4,3 раза, что довольно близко к идеальному.

Таблица 5

Детализованное время исполнения модуля cr4dy1

Модули	Время выполнения последовательного расчета, мс	Время выполнения на 7 процессорах, мс		Время выполнения на 35 процессорах, мс	
		Главный процесс	Второстепенный процесс	Главный процесс	Второстепенный процесс
cr4dy1_p1	17,92	0,12	0,31	0,24	0,74
cr4dy1_p2		2,69	2,62	0,62	0,53
cr4dy1_p3		0,51	0,34	2,48	0,12
cr4dy1_p4		0,11	0,00	0,11	0,00

Таким образом, для увеличения эффективности вычислений следует снизить долю времени, которое затрачивается на пересылку данных. Решение этой задачи является наиболее перспективным направлением дальнейшего исследования.

6. Производительность расчета пространственной функции

В табл. 6 представлены результаты распараллеливания модуля расчета пространственной функции (scrd03), проведенного по методике, которая изложена в предыдущем разделе.

С использованием 7 процессов было достигнуто ускорение всего в 2,5 раза, что составляет эффективность распараллеливания 35 %. В случае разбиения на 35 процессов ускорение немного ниже (2,33), что соответствует эффективности всего в 6 %. Данный факт обусловлен тем обстоятельством, что сам алгоритм расчета пространственной функции имеет довольно тесные связи между гранулами параллелизма. Как следствие, число пересылок увеличивается, доля времени, затраченного на пересылки, растет, в силу чего уменьшается производительность расчета на основе рассматриваемой методики.

К сожалению, вследствие структуры используемого алгоритма проанализировать долю времени, затраченного на пересылки, не представляется возможным. Однако по времени последовательного расчета и по числу пересылок в коде приблизительную оценку можно получить. Время последовательного исполнения модулей приблизительно одинаковое. Однако при расчете нейтронно-физических констант функции коллективного обмена вызываются 5 раз, а при расчете пространственной функции они вызываются более 40 раз, что больше на порядок. Исходя из этого можно предположить, что пересылки данных занимают почти все время расчета.

Таким образом, сокращение затрат на обмен данными становится основным способом оптимизации вычислений. Альтернативой обычному коллективному обмену данных является новый алгоритм передачи данных, который учитывал бы особенности разбиения расчетной сетки и алгоритма расчета. Основой для реализации такого подхода является тот факт, что для расчета каждому процессу нужны данные не во всей расчетной сетке, а только в ее части. Расчетная область каждого процесса граничит максимум с восемью областями других процессов (для разбиения на 35 процессов). По этой причине существует возможность применения децентрализованного обмена данными на этапе внешних итераций, который осуществляется только между процессами с соседними областями расчета.

Таблица 6

Время расчета пространственной функции

Время выполнения последовательного расчета, мс	Время выполнения на 7 процессорах, мс		Время выполнения на 35 процессорах, мс	
	Главный процесс	Второстепенный процесс	Главный процесс	Второстепенный роцесс
14,01	5,62	5,27	5,99	5,28

Данный подход более затратен, как с точки зрения числа пересылок, так и с точки зрения вычисления номеров процессов с граничащими областями. Однако он позволяет проводить расчет асинхронно в большей части времени исполнения программы, что может дать существенный выигрыш по времени, который покроет излишние расходы на пересылки и вычисление соседей. К тому же предлагаемый подход может быть существенно оптимизирован под целевую аппаратную платформу. Это обусловлено тем обстоятельством, что пропускная способность между процессорами разная, а следовательно, после минимизации количества общих границ расчетных областей "далеких" процессоров минимизируется среднее время одной пересылки. Указанный подход является довольно трудоемкой задачей, потому он не был реализован в рамках исследования, результаты которого представлены в настоящей статье. Однако его реализация и апробация на практике является перспективным направлением дальнейших исследований на рассматриваемом направлении.

Заключение

Исследования, представленные в настоящей публикации, являются лишь первым шагом к эффективной адаптации нейтронно-физического расчетного кода ТРЕК для использования на суперкомпьютерных платформах. В первую очередь необходима дальнейшая оптимизация механизмов обмена данными в системе. Снижение затрат времени на передачи данных позволит использовать массовый параллелизм, присущий архитектурам современных суперЭВМ, которые зачастую используют не только традиционные CPU, но и специализированные аппаратные ускорители. Поддержка массового параллелизма предоставляет возможность для дальнейшего безболезненного увеличения точности расчетов за счет использования более "тонкой" расчетной сетки. Как правило, чем объемнее задача, тем лучше она решается распределенными вычислительными методами.

В случае отсутствия возможности повысить эффективность параллельной версии нейтронно-физического расчетного кода, целесообразно будет рассмотреть возможность перехода от итерационного метода поиска решения к вычислению точного решения. Такой подход не рассматривался в рамках данной работы, однако с учетом возможностей современных высокопроизводительных вычислительных установок он имеет право на жизнь.

Список литературы

1. **Пряничников А. В.** Описание программы GETERA // ВАНТ. Сер. Физика ядерных реакторов. 2009. Вып. 3. С. 63—76.

2. **MPI-2:** Extensionsto the Message-PassingInterface. Message PassingInterface Forum. URL: http://www.mpi-forum.org/docs/mpi-20.ps.

3. **Компактные** Супер-ЭВМ производства ФГУП "РФЯЦ-ВНИИЭФ". URL: http://www.vniief.ru/directions/grazrab/catalog/inf-prod/ razrabotki/ super- evm.html

УДК 004.925

А. А. Антонченков, аспирант,
В. В. Шилов, канд. техн. наук, зав. каф.,
МАТИ — РГТУ имени К. Э. Циолковского,
e-mail: A.A.Antonchenkov@gmail.com

Комплекс алгоритмов отсечения невидимых поверхностей для трехмерной визуализации пластовых моделей

Представлен комплекс алгоритмов отсечения невидимых поверхностей для трехмерной визуализации пластовых моделей. Дано описание подхода с двумя стадиями подготовки исходных данных для использования предлагаемых алгоритмов. Приведены формулировки алгоритмов с описанием их ключевых особенностей и назначения. Показан порядок применения алгоритмов для достижения максимальной эффективности их работы.

Ключевые слова: *трехмерная визуализация, пластовая модель, отсечение невидимых поверхностей, алгоритмы отсечения*

Компьютерное моделирование сегодня широко применяется в подавляющем большинстве областей научных исследований. Важным направлением компьютерного моделирования является трехмерное моделирование и визуализация. Они включают построение вершинной модели объекта и ее визуализацию — создание трехмерного изображения объекта на основании полученной модели. Трехмерное изображение моделируемого объекта является наиболее удобной для человека формой представления данных. Для некоторых областей исследований трехмерная визуализация играет особую роль, например, для архитектуры, авиа- и кораблестроения, геологии [1].

В разных областях исследований существует класс моделей, служащих для изучения структурных особенностей слоистых структур [2]. Такие модели, представляющие каждый слой изучаемой структуры в виде отдельного пласта, обладающего характерными отличительными свойствами, называются пластовыми. Пласты с разными характеристиками для большей наглядности обычно отображают разными цветами, как показано на рис. 1 (см. вторую сторону обложки).

В большинстве случаев слой модели представляется набором точек, каждая из которых описывается тремя координатами, толщиной слоя в данной точке, и набором значений (качественных характеристик), описывающих физико-химические свойства слоя в данной точке. Точки являются узлами регулярной сетки, единой для всех пластов одной модели. Для визуализации пласт представляется набором из трех поверхностей: верхней (кровли), нижней (подошвы) и боковой. Для получения кровли и подошвы пласта проводится триангуляция поверхности пласта по узлам сетки. Боковая поверхность строится соединением крайних точек кровли и подошвы по периметру сегмента для отображения его толщины.

Крайне важной представляется задача трехмерной визуализации пластовых моделей, содержащих большие объемы данных. Известные подходы к решению этой задачи базируются в основном на уменьшении количества данных, используемых для построения трехмерного изображения модели [3].

Высокую эффективность имеют алгоритмы, предусматривающие динамическое изменение уровня детализации модели [4, 5]. Однако для трехмерной визуализации пластовых моделей алгоритмы этого класса не подходят, поскольку динамическое изменение уровня детализации может приводить к существенным изменениям рельефа пластов, неточности отображения, увеличению времени проведения анализа и появлению ошибок.

Класс алгоритмов, основанных на отсечении невидимых поверхностей, имеет наибольшую эффективность при визуализации моделей, содержащих большие объемы данных [6]. Однако они требуют при построении каждого кадра длительных предварительных вычислений и многошаговой обработки, что увеличивает время подготовки к работе с моделью и, при значительном увеличении количества данных, может значительно снижать скорость отрисовки. Особое место среди алгоритмов отсечения невидимых поверхностей занимает алгоритм, использующий пирамиду видимости, применимый для широкого круга задач. Этот алгоритм также применим и для задачи визуализации пластовых моделей, содержащих большие объемы данных. Предлагаемый в работе комплекс алгоритмов включает в себя модифицированную под специфику задачи версию одной из известных реализаций данного алгоритма.

Предлагаемый подход к решению рассмотренной задачи заключается в алгоритмическом отсечении невидимых поверхностей, образующихся в результате перекрытия объектами модели друг друга, на основе углов обзора. Под отсечением подразумевается исключение информации о невидимых поверхностях из данных, передаваемых на видеокарту для отрисовки. За счет чего возрастает скорость отрисовки, обеспечивается плавность визуализации и увеличивается эффективность работы с моделью [7].

Для работы алгоритмов отсечения невидимых поверхностей необходимо реализовать логическое разбиение модели на части, каждая их которых упрощенно представляет часть поверхности пласта, что позволяет сократить число вычислений при принятии решения об отсечении. Для логического разделения пласта координатное пространство модели делится на квадранты размером $N \times N$ точек сетки пласта, каждый из которых содержит информацию обо всех точках и поверхностях пласта, попадающих в него (N может задаваться пользователем в зависимости от производительности аппаратного обеспечения). Эти квадранты назовем сегментами. Сегменты бывают двух видов: полные и неполные. В неполном сегменте часть ячеек пласта отсутствует. На рис. 2 (см. вторую сторону обложки) границы полных сегментов показаны белым цветом, а неполных сегментов — красным цветом.

Разделение на сегменты сквозное для всей модели. Это означает, что сегменты разных пластов с одинаковыми координатами ширины и долготы располагаются друг над другом, как показано на рис. 3 (см. вторую сторону обложки). Таким образом модель можно логически разбить на наборы сегментов с одинаковыми координатами. Данные наборы называются вертикалями.

Для работы комплекса алгоритмов отсечения невидимых поверхностей предлагается разбить обработку данных модели на две фазы.

Первая фаза — подготовительная. Во время нее проводятся загрузка и анализ данных, в ходе которых собирается информация о структурных особенностях пластов в модели. Первую фазу обработки данные проходят один раз на начальной стадии работы комплекса моделирования, при подготовке модели к визуализации. Полученные в ходе анализа данные сохраняются для последующего использования на второй фазе обработки.

Подготовительная фаза включает несколько этапов обработки входных данных и сбора информации для дальнейшей работы алгоритмов.

1. Входные данные загружаются из файлов на диске.

2. Осуществляется пропорциональное масштабирование координат вершин из входных данных для получения значений, пригодных для рендеринга.

3. Вершины, описывающие пласт модели, группируются по координатам широты и долготы, тем самым пласт разделяется на набор независимых однотипных объектов — сегментов. Для каждого сегмента проводится триангуляция по координатам вершин, попадающих в него (подошвы, кровли и боковой поверхности, если она для данной части пласта существует), что позволяет получить данные для рендеринга части поверхностей пласта. По полученным данным поверхностей пласта строится ограничивающий объем этой группы вершин [8].

4. Полученный набор сегментов пластов модели группируется в вертикали по совпадению координат широты и долготы крайних точек ограничивающего объема. Внутри вертикали для каждого сегмента вычисляются критические углы обзора, при которых он загораживается соседними сегментами. Для каждого сегмента также вычисляется критический угол обзора, при котором одна из поверхностей пласта загораживается другой.

Рассмотрим более подробно этап 4 подготовительной фазы обработки, а именно расчет критических углов обзора. Дальнейшие пояснения будем вести из предположения, что пользователь находится в области пространства над верхним пластом. Для того чтобы утверждать, что сегмент нижнего пласта загораживается от пользователя верхним пластом, следует выбрать некоторый перекрывающий его объект [9]. Предлагается перекрывающим объектом принять группу сегментов верхнего пласта, состоящую из расположенного в той же вертикали сегмента верхнего пласта и его окружения. Под окружением следует понимать восемь соседних сегментов. При этом надо учитывать, что гарантированную непрозрачность такого составного объекта может обеспечить лишь условие, что все сегменты в нем полные. Таким образом, перекрывающий объект будет иметь квадратную форму и линейный размер в 3 раза больше, чем перекрываемый сегмент нижнего пласта. Пример перекрывающего объекта приведен на рис. 4, см. вторую сторону обложки.

В силу того, что толщина модели гораздо меньше, чем ее протяженность, пространство модели можно разбить на три области: верхнюю, нижнюю и центральную. Проверка видимости основана на расчете области пространства, находясь в которой пользователь не будет видеть перекрываемый сегмент из-за перекрывающего объекта. Для расчета этой области не-

обходимо определить углы обзора перекрывающего объекта, при которых пользователь гарантированно не увидит лежащий под ним сегмент нижнего пласта. Эти углы рассчитываются по границам ограничивающих объемов перекрываемого и перекрывающих сегментов и на рис. 5 обозначены цифрами (см. третью сторону обложки).

Для сокращения вычислений выбирается самый пессимистический случай, т. е. меньший из восьми углов, который и называется критическим углом обзора. Для упрощения расчетов угол обзора перекрываемого пласта считается от центра верхней грани ограничивающего объема перекрываемого сегмента, которую назовем базовой. Текущее положение камеры вида в таком случае определяется вектором, соединяющим ее с базовой точкой перекрываемого сегмента. Таким образом, искомое положение камеры вида принадлежит конусу с вершиной в базовой точке, осью, совпадающей с центральной вертикалью перекрываемого сегмента, и углом раствора равным двум критическим углам обзора. Если текущее положение пользователя в пространстве находится внутри этого конуса, то можно утверждать, что при любом направлении взгляда пользователь не сможет увидеть перекрываемый сегмент, и он может быть исключен из процесса формирования кадра. Аналогичным образом рассчитывается критический угол для отсечения внутри сегмента, в качестве отсекаемой поверхности берется одна из сторон сегмента, а в качестве перекрывающего объекта — противоположная сторона и соответствующие ей части из соседних сегментов.

Вторая фаза — отсечение и отображение. Визуализация проводится в интерактивном режиме и основывается на покадровой анимации. Для формирования очередного кадра изображения алгоритмы отсечения используют данные, полученные на первой фазе обработки, и текущее положение пользователя в пространстве модели для формирования набора данных для рендеринга. Информация о полученном наборе данных сохраняется в текущее состояние комплекса и используется для отрисовки последующих кадров. Формирование набора данных происходит только при изменении пользователем своего положения в пространстве модели [10].

Формирование набора данных для отображения осуществляется комплексом алгоритмов последовательной обработкой общего набора данных. Каждый следующий алгоритм учитывает результаты работы предыдущих алгоритмов и работает на меньшем объеме данных в целях исключения повторного рассмотрения уже отсеченных объектов. Алгоритмы применяются к исходному набору данных в следующей последовательности.

1. ***Отсечение невидимых поверхностей пирамидой видимости.*** На основании текущего положения пользователя в пространстве вычисляются плоскости пирамиды видимости. Проводится проверка взаимного расположения ограничивающих объемов объектов модели и пирамиды видимости. Объекты, ограничивающие объемы которых не попали внутрь пирамиды видимости, отсекаются.

2. ***Отсечение перекрываемых сегментов.*** Внутри каждой вертикали по полученным в подготовительной фазе данным о критических углах обзора определяются сегменты, которые в данный момент перекрываются другими сегментами соседних пластов. Эти объекты считаются невидимыми и отсекаются.

3. ***Отсечение перекрывающихся поверхностей внутри сегмента.*** Для каждого сегмента по критическим углам обзора определяется видимость поверхностей в данный момент и при необходимости одна из поверхностей отсекается.

4. Данные о поверхностях полученного усеченного набора объектов передаются на видеокарту для рендеринга, в ходе которого на ранних этапах конвейера происходит дополнительное отсечение обратных сторон поверхностей, определяемое по направлению нормали.

Рассмотрим каждый из перечисленных выше алгоритмов.

Отсечение невидимых сегментов пирамидой видимости. Первым шагом фазы отсечения является отсечение невидимых сегментов пирамидой видимости. Пирамида видимости представляется усеченной призмой, стороны которой служат плоскостями отсечения, меньшее основание называется ближней плоскостью отсечения, большее — дальней. Для отсечения частей модели, не попадающих в пирамиду видимости, используется модификация одной из самых эффективных реализаций этого алгоритма [11, 12]. Модификация заключается в исключении проверки по дальней плоскости отсечения. В подготовительной фазе параметры пространства модели и проекции вида задаются таким образом, что при любом положении камеры вида в пространстве никакие сегменты модели не могут оказаться за дальней плоскостью пирамиды видимости и не могут быть ею отсечены. С учетом индексирования плоскостей отсечения и исключения дальней плоскости отсечения из проверки алгоритм имеет следующий вид:

```
Для каждого индекса плоскости отсечения в массиве из пяти элементов:
   1. По знаку составляющих нормали плоскости отсечения выбрать соответствующую дальнюю точ-
      ку ограничивающего объема.
   2. Рассчитать расстояние от точки ограничивающего объема до плоскости отсечения.
   3. Если расстояние меньше 0 — ограничивающий объем находится вне пирамиды видимости:
         a. поменять местами индекс текущей плоскости отсечения с первым индексом в массиве;
         b. закончить проверку.
   4. Иначе ограничивающий объем пересекает плоскость или находится внутри пирамиды видимости,
      продолжить проверку для оставшихся плоскостей отсечения.
```

Модифицированная реализация требует провести вычисление от одного до пяти скалярных произведений (оригинальная версия требует вычислить от одного до шести скалярных произведений) для каждого объекта модели для определения его видимости в текущем кадре. В худшем случае предлагаемый алгоритм по количеству необходимых вычислений уступает в пять раз алгоритму, использующему в качестве ограничивающего объема сферу, что компенсируется точностью отсечения сегментов на границе пирамиды видимости. Пример работы предлагаемого алгоритма отсечения приведен на рис. 6, см. третью сторону обложки.

Для большей наглядности объяснения остальных алгоритмов предположим, что пользователь рассматривает содержащую 3,6 млн треугольников модель с некоторой позиции, модель для него выглядит так, как показано на рис. 7, см. третью сторону обложки.

Для отображения результатов работы алгоритма отсечения представим, что можем заглянуть на обратную сторону модели и будем использовать принудительную заливку цветом и расчет освещенности для обратных сторон примитивов (данный прием требует дополнительных вычислительных затрат и не используется при нормальной работе комплекса визуализации для увеличения скорости работы). До применения каких-либо алгоритмов вид с обратной стороны модели показан на рис. 8, см. третью сторону обложки.

Отсечение перекрытых сегментов. Этот алгоритм используется для расчета отсечения в видимой части модели сегментов вследствие перекрытия пластов, не видимых пользователю из текущего положения. Проверка основана на расчете области пространства, при нахождении в которой пользователь не будет видеть перекрываемый сегмент из-за перекрывающего составного объекта. Эта область определяется по углам обзора перекрывающего объекта, при которых пользователь гарантированно не увидит лежащий за объектом сегмент нижнего пласта. Алгоритм рассчитывает угол обзора перекрываемого сегмента и сравнивает его с критическим углом обзора, рассчитанным по нижним границам ограничивающих объемов перекрываемого сегмента и нижним границам перекрывающих сегментов. Таким образом определяется конус пространства, при нахождении в котором пользователь не может видеть загораживаемый сегмент. Алгоритм формулируется следующим образом:

```
Для каждого сегмента каждого пласта:
   1. Если сегмент помечен как невидимый, то перейти к следующему сегменту.
   2. Иначе:
        a. Определить часть пространства, в котором находится камера вида;
        b. Выбрать соответствующий список перекрывающих сегментов соседних пластов. Для кри-
           тического угла из списка:
             i. Если угол обзора сегмента меньше критического угла обзора — пометить сегмент
                как невидимый и перейти к следующему сегменту;
            ii. Иначе продолжить проверку для остальных углов.
```

Результат работы этого алгоритма показан на рис. 9 (см. четвертую сторону обложки), на котором видно, что часть сегментов пластов, соответствующих критерию алгоритма, не отображается.

Отсечение обратных сторон сегментов 1. Алгоритм перекрытия сегментов можно адаптировать на уровень пласта. По аналогии с перекрывающими сегментами вышележащего пласта, части кровли пласта смежных сегментов могут являться перекрывающими объектами для частей подошвы пласта тех же сегментов. По аналогии с перекрывающими сегментами соседних пластов, для каждой поверхности сегмента следует определить критический угол обзора и в фазе отсечения исключать одну из поверхностей сегмента, если она не видна пользователю из данного положения в пространстве модели.

Преимущество данного алгоритма заключается в существенном приросте скорости отрисовки модели. Как правило, соседние сегменты пласта имеют довольно пологий рельеф и малую толщину. Критический угол обзора при этом получается очень большим, что позволяет отсекать обратную сторону сегмента в достаточно широком диапазоне положений пользователя в пространстве модели. Недостатком алгоритма является то, что он охватывает не все сегменты пласта, поскольку для сегментов, находящихся на краях пласта, перекрывающий объект отсутствует.

Алгоритм отсечения имеет следующий вид:

```
Для каждого сегмента каждого пласта:
   1. Если сегмент неполный или один из соседних сегментов неполный, то перейти к следующему
сегменту.
   2. Иначе:
        a. Определить полупространство, в котором находится камера вида;
        b. Если угол обзора сегмента меньше критического угла обзора — пометить обратную сто-
           рону сегмента как невидимую.
```

Результат работы этого алгоритма показан на рис. 10, см. четвертую сторону обложки.

На этом рисунке у части сегментов, имеющих наиболее пологий рельеф и соответствующих критерию алгоритма, отсутствует нижняя по отношению к пользователю поверхность, при этом видна верхняя поверхность с неестественной освещенностью (разница хорошо заметна при сравнении с рис. 8, см. третью сторону обложки).

Отсечение обратных сторон сегментов 2. Этот алгоритм также предназначен для определения возможности отсечения обратной относительно положения пользователя в пространстве поверхности сегмента. Однако в отличие от алгоритма перекрытия поверхностей сегмента, он позволяет работать и с крайними сегментами пласта, так как затрагивает лишь свойства поверхностей внутри одного сегмента.

Для определения отсечения обратной стороны сегмента текущий угол обзора сегмента сравнивается с определенным максимальным углом уклона рельефа сегмента. Таким образом определяется положение пользователя внутри конуса в пространстве над сегментом, в котором обратная сторона сегмента не видна.

Алгоритм имеет следующий вид:

```
Для каждого сегмента каждого пласта:
   1. Определить часть пространства, в котором находится камера вида.
   2. Выбрать соответствующий части пространства критический угол обзора.
   3. Если угол обзора сегмента меньше критического угла обзора — пометить обратную сторону
      сегмента как невидимую.
```

Результат работы этого алгоритма показан на рис. 11, см. четвертую сторону обложки.

Критерий отсечения невидимых поверхностей данного алгоритма охватывает довольно большое число сегментов с пологим рельефом, поэтому на данном рисунке мы практически не видим обратных по отношению к положению пользователя сторон сегментов (это хорошо заметно при сравнении рельефа поверхностей и их освещенности с рис. 8, см. третью сторону обложки). В дальней левой части модели расположены пласты чашеобразной формы с сегментами, имеющими большой перепад высот, можно заметить, как часть нижних сторон сегментов все-таки отображается, и видно их границы.

Применение предлагаемого комплекса алгоритмов отсечения для выбранной позиции позволяет сократить число треугольников для отрисовки с 3,6 млн до 1,4 млн.

Предлагаемые алгоритмы отсечения невидимых поверхностей при трехмерной визуализации пластовых моделей обладают следующими свойствами:

- универсальность — алгоритмы применимы для любых пластовых моделей с регулярной сеткой;
- точность — алгоритмы не имеют ошибок определения невидимых поверхностей и не допускают отсечения видимых частей модели;
- вычислимость параметров — алгоритмы автоматически подстраиваются под конкретную модель;
- инвариантность — алгоритмы не используют специфических возможностей аппаратного окружения (таких как расширения OpenGL, встраиваемых производителями видеокарт).

Список литературы

1. **Антонченков А. А.** Задачи трехмерной визуализации в ГИС и программный прототип для их решения // Информационные технологии. 2010. № 3. С. 2—5.
2. **Ажгирей Г. Д.** Структурная геология. М.: Изд-во МГУ, 1956. 494 с.
3. **Cohen-Or D., Fibich G., Halperin D., Zadicario E.** Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes // Computer Graphics Forum. 1998. Vol. 17. Is. 3. P. 243—253.
4. **Aila T., Miettinen V.** dPVS: An Occlusion Culling System for Massive Dynamic Environments // IEEE Computer Graphics and Applications. 2004. Vol. 24. № 2. P. 86—97.
5. **Luebke D., Erikson C.** View-Dependent Simplification of Arbitrary Polygonal Environments // Proceedings of the 24th annual conference on Computer graphics and interactive techniques. 3—8 August 1997. Los Angeles, California, USA. ACM Press/Addison-Wesley Publishing, 1997. P. 198—208.
6. **Hoff K.** Faster 3D game graphics by not drawing what is not seen // Crossroads — Special issue on computer graphics. 1997. Vol. 3. Is. 4. P. 20—23.
7. **Brownlow K.** Silent Films: What Was the Right Speed? // Sight and Sound. Summer 1980. P. 164—167.
8. **Assarsson U., Moller T.** Optimized View Frustum Culling Algorithms // Technical Report 99-3. Department of Computer Engineering, Chalmers University of Technology, 1999.
9. **Coorg S., Teller S.** Real-Time Occlusion Culling for Models with Large Occluders // Symposium ACM on Interactive 3D Graphics, 27—30 April 1997. Providence, RI, USA., NY: ACM New York, 1997. P. 83—90.
10. **Wonka P., Wimmer M., Schmalstieg D.** Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs // Rendering Techniques 2000. Proceedings Eurographics Workshop on Rendering. 26—28 June 2000. Brno, Czech Republic. Springer, 2000. P. 71—82
11. **Sykora D., Jelinek J.** Efficient View Frustum Culling // In Proceedings of the 6th Central European Seminar on Computer Graphics. 22—24 April 2002. Budmerice, Slovakia. Springer, 2002. P. 55—64.
12. **Assarsson U., Moller T.** Optimized View Frustum Culling Algorithms for Bounding Boxes // Journal of graphics tools. 2000. Vol. 5. № 1. P. 9—22.

УДК 519.246

Л. Ю. Бараш, канд. физ.-мат. наук, мл. науч. сотр.,
Л. Н. Щур, д-р физ.-мат. наук, вед. науч. сотр.,
Институт теоретической физики им. Л. Д. Ландау РАН, г. Черноголовка
e-mail: shchur@chg.ru

О генерации параллельных потоков псевдослучайных чисел

Предложен подход к разработке алгоритмов и созданию программных средств генерации параллельных потоков псевдослучайных чисел, при использовании которого удается создавать большое число некоррелированных потоков. Изложены также подходы к генерации параллельных потоков псевдослучайных чисел с использованием наиболее распространенных генераторов, основанные на специальных алгоритмах инициализации таких генераторов. Приведен пример реализации генераторов с использованием графических ускорителей. Все алгоритмы воспроизводят эталонную последовательность.

Ключевые слова: *метод Монте-Карло, псевдослучайные числа, графические ускорители, гибридные суперкомпьютеры*

Введение

Псевдослучайные числа являются важным элементом моделирования методом Монте-Карло [1]. Используемые в настоящее время для этих целей генераторы таких чисел показывают хорошие результаты при тестировании на наборах из сотен статистических тестов (результаты тестирования можно найти в статье [2]). Однако эффективное применение метода Монте-Карло на современных суперкомпьютерных системах ограничено в силу отсутствия генераторов, способных производить большое число параллельных (и некоррелированных) потоков псевдослучайных чисел. В настоящей статье обсуждаются возможные подходы к решению отмеченной выше задачи.

1. Генераторы псевдослучайных чисел. Достижения и неудачи

Известны два основных типа генераторов. Первые представляют собой устройства, использующие стохастические свойства некоторых физических процессов, которые могут подвергаться оцифровыванию и преобразованию в машинное слово (см., например, [3]). Такие устройства носят название генераторы случайных чисел (ГСЧ), в англоязычной литературе — TRNG (*True Random Number Generators*). Вторые — это устройства, использующие способ преобразования машинных слов. Такие устройства именуются генераторами псевдослучайных чисел (ГПСЧ), в англоязычной литературе — PRNG (*Pseudo Random Number Generators*) [4].

Существующие ГСЧ производят последовательность случайных чисел, которая не является воспроизводимой, т. е. не может быть повторена в том же виде. Это делает их неприменимыми для целей отладки и верификации программных систем и программно-аппаратных комплексов. Именно поэтому они не применяются при моделировании методом Монте-Карло.

Нежелательные корреляции в последовательности псевдослучайных чисел могут приводить к существенному искажению численных результатов [5]. Показательны следующие далее два примера.

В 1967 г. были обнаружены существенные дефекты линейно-конгруэнтных генераторов, используемых в системе IBM 360/370 [6]. Например, при заполнении n-мерного куба последовательностью псевдослучайных чисел, генерируемой такими генераторами, точки

заполняют равномерно не куб, а серию параллельных $(n-1)$-мерных гиперповерхностей внутри куба [7]. Использование таких генераторов для многомерного численного интегрирования приведет к существенному искажению результатов.

В 1992 г. были обнаружены серьезные искажения в результатах моделирования методом Монте-Карло спиновых систем, в котором использовались генератор псевдослучайных чисел R250 (сдвиговый регистр) и кластерный метод Вольфа [8]. Этот факт был воспринят очень серьезно, он получил резонанс не только в научной литературе, о нем сообщила газета Нью-Йорк Таймс в разделе "Технологии" за 12 января 1993 г.

Таким образом, на каждом витке качественного развития вычислительных систем обнаруживаются дефекты используемых генераторов псевдослучайных чисел. Генераторы, применение которых на текущем этапе приводит к хорошим результатам моделирования, оказываются непригодными для использования на следующем поколении более высокопроизводительной вычислительной техники. При проведении расчетов методом Монте-Карло на суперкомпьютерных системах производительностью более сотни терафлоп ожидается, что в существующих генераторах случайных чисел проявятся дефекты. Необходим поиск новых методов генерации случайных чисел и реализация эффективных алгоритмов и методов для использования на современных суперкомпьютерных системах, для которых типично использование не менее 10^{15} чисел в одном расчете.

Еще одна задача, требующая решения, возникает при проведении расчетов на параллельных вычислительных установках, включая гибридные суперкомпьютерные системы. В таких расчетах необходим эффективный метод генерации некоррелированных параллельных потоков случайных чисел, а также его реализация в виде программного обеспечения и библиотек генераторов. Эта задача до сих пор не решена.

2. Основные методы генерации псевдослучайных чисел

Наиболее широко используются методы генерации псевдослучайных чисел, которые целесообразно разделить на два основных класса, а именно линейно-конгруэнтный метод и метод сдвиговых регистров.

В *линейно-конгруэнтном методе* (ЛКМ) последовательность псевдослучайных чисел $(x_0, x_1,..., x_n,...)$, не превосходящих модуль M, вычисляется после задания начального значения x_0 по следующей формуле [9]:

$$x_{n+1} = (ax_n + c)(\mathrm{mod}M), \qquad (1)$$

где a и c — некоторые константы.

Метод ЛКМ имеет два существенных недостатка. Первый заключается в алгоритмическом ограничении на максимальный период, который не может превосходить машинную длину целого числа. Например, последовательность длины $2^{32} \approx 4 \cdot 10^9$ исчерпывается на современных персональных компьютерах в течение нескольких секунд. Второй недостаток связан с тем обстоятельством, что ЛКМ не следует использовать в приложениях, имеющих дело со случайными векторами в многомерном пространстве, что было отмечено нами в предыдущем разделе.

Метод генерации с помощью сдвиговых регистров (ГСР) основан на свойствах линейных операций по модулю 2 над битами x_n. Рассмотрим последовательность

$$x_n = (a_1 x_{n-1} + ... + a_k x_{n-k})(\mathrm{mod}2). \qquad (2)$$

Характеристическим полиномом этой последовательности является $P(z) = z^k - a_1 z^{k-1} - ... - a_k$. Это линейно-рекуррентное соотношение в поле $\mathbb{Z}_2$, состоящем из двух элементов, нуля и единицы. Такое рекуррентное соотношение называется сдвиговым регистром и имеет период длины $p = 2^k - 1$ тогда и только тогда, когда P является примитивным полиномом ([10]). Генератором на сдвиговом регистре называется генератор с выходной последовательностью

$$u_n = \Sigma_{i=1}^{L} x_{ns+i-1} 2^{-i},$$

где размер шага s и длина слова L — целые положительные числа.

Из представленного определения следует, что генераторы, основанные на сдвиговом регистре, быстрые. Они обладают очень большим периодом при условии правильного выбора примитивных триномов. Однако в генераторах этого класса были обнаружены корреляции, которые могут привести к систематическим ошибкам в вычислениях на основе метода Монте-Карло. Эти свойства были подробно изучены в работах [5, 8, 11—16].

После обнаружения упомянутых выше дефектов были предприняты попытки модификации и обобщения методов ЛКМ и ГСР для достижения хороших статистических свойств последовательности. В качестве примеров можно привести перечисленные далее: генератор RANLUX [17]; генератор MT19937 [18]; комбинированные ЛКМ-генераторы MRG32K3A [19]; комбинированные генераторы из сдвиговых регистров LFSR113 [20, 21]. Понятие генератора случайных чисел может быть формализовано следующим образом: генератор — это структура $G = (S, s_0, T, U, G)$, где S — конечное *множество состояний*; $s_0 \in S$ — начальное состояние, преобразование $T: S \to S$ — *функция перехода*; U — конечное множество выходных символов; $G: S \to U$ — *выходная функция* генератора [22]. Таким образом, состоянием генератора изначально является s_0, и генератор меняет свое состояние на каждом шаге, вычисляя на шаге n значения $s_n = T(s_{n-1})$ и $u_n = G(s_n)$. Значения u_n на выходе генератора называются *наблюдаемыми значениями* или просто *псевдослучайными числами* на выходе генератора.

Как известно, последовательность битов обладает *свойством равнораспределения вероятностей* в размерности l, если для каждого $n \leqslant l$ все комбинации из последовательных n битов появляются одно и то же число

раз в битовой последовательности и имеют соответствующую вероятность $1/2^n$. Как правило, на выходе генератора существует последовательность 32-битовых чисел. В этом случае можно составить последовательность из ν-битовых блоков при помощи выделения ν последовательных битов (например, ν старших битов) в каждом из чисел выходной последовательности. Выходная последовательность генератора имеет свойство ν-*битового равнораспределения вероятности* в размерности l, если для каждого $n \leqslant l$ все комбинации из n последовательных ν-битовых блоков появляются одно и то же число раз в выходных последовательностях генератора и имеют соответствующую вероятность $1/2^{\nu n}$. Можно ввести наиболее естественную для каждого конкретного генератора псевдослучайных чисел меру вероятности на множестве последовательностей ν-битовых блоков, что позволяет рассматривать свойства равнораспределения в деталях со строгой математической точки зрения и доказывать соответствующие утверждения (см., например, [20, 23]). Свойство многомерного равнораспределения вероятностей является одним из важнейших свойств, характеризующих качество последовательности псевдослучайных чисел (см., например, [18, 20, 23—26]).

Генератор RANLUX используется для расчетов в области физики высоких энергий и популярен среди сообщества, участвующего в моделировании процессов при столкновении частиц в большом адронном коллайдере. Он основан на ГСР короткой длины и эмпирическом подходе по модификации выходной последовательности за счет пропуска части чисел. Этот прием, однако, не избавляет от отсутствия корреляций, которые могут существенно повлиять на результаты моделирования [27].

В генераторе MT19937 [18] наблюдается точное равнораспределение вероятности в размерности до 623. При этом период генерируемой последовательности порядка 10^{6001}.

Для комбинированного ЛКМ-генератора MRG32K3A точное равнораспределение не выполняется. Однако по результатам изучения так называемых коэффициентов его доброкачественности (*figures of merit*) утверждается [19], что он ведет себя близко к равнораспределению в размерности до 45, а период генерируемой им последовательности порядка 10^{57}.

Для комбинированного генератора из четырех сдвиговых регистров LFSR113 [20,21] точное равнораспределение наблюдается в размерности не выше 30, а период последовательности порядка 10^{34}. Статистические тесты выявляют дополнительные корреляции в выходных последовательностях MT19937 и LFSR113, но они не оказывают существенного влияния на точность расчетов с использованием суперкомпьютеров на сегодняшний день [28].

Принимая изложенные выше обстоятельства, в работах [29, 30] был предложен алгоритм построения генераторов псевдослучайных чисел, основанный на параллельной эволюции ансамбля преобразований тора. Множеством состояний такого генератора является $S = L^s$, где $L = \{0, 1,..., g-1\}^2$, g и s — целые числа: $s \leqslant 32$, g является произведением достаточно большого простого числа и некоторой степени двойки. Таким образом, состояние содержит s пар целых чисел из множества $\{0, 1,..., g-1\}$. Обозначим эти числа на шаге n процесса генерации следующим образом: $x_i^{(n-1)}$, $x_i^{(n-2)} \in \{0, 1,..., g-1\}$, $i = 0, 1,..., s-1$. Тогда функция перехода генератора определяется рекуррентным соотношением

$$x_i^{(n)} = kx_i^{(n-1)} - qx_i^{(n-2)} (\mathrm{mod} g),\ i = 0,1,\ ...,\ s-1. \quad (3)$$

Здесь k и q — целые числа. Значения $x_i^{(n)}$, $i = 0,1,\ ...,\ s-1$ можно рассматривать как абсциссы s точек $\left(x_i^{(n)},\ y_i^{(n)}\right)^{\mathrm{T}}$, $i = 0,1,..., s-1$, лежащих на решетке $g \times g$ на двумерном торе. Тогда каждое рекуррентное соотношение описывает динамику точки двумерного тора:

$$\begin{pmatrix} x_i^{(n)} \\ y_i^{(n)} \end{pmatrix} = \mathbf{M} \begin{pmatrix} x_i^{(n-1)} \\ y_i^{(n-1)} \end{pmatrix} (\mathrm{mod} g),$$

где $\mathbf{M}\begin{pmatrix} m_1 & m_2 \\ m_3 & m_4 \end{pmatrix}$ — матрица с целыми элементами, при этом $k = \mathrm{Tr}\mathbf{M}$ и $q = \det\mathbf{M}$, т. е. след матрицы $\mathbf{M}$ и определитель матрицы $\mathbf{M}$ соответственно (см., например, [23]). Пусть $\alpha_i^{(n)}$ является 0 или 1 в зависимости от того, $x_i^{(n)} < g/2$ или $x_i^{(n)} \geqslant g/2$, т. е. $\alpha_i^{(n)} = \left\lfloor 2x_i^{(n)}/g \right\rfloor$. Тогда искомое псевдослучайное число на выходе генератора следующее: $a^{(n)} = \sum_{i=0}^{s-1} \left\lfloor 2^{\nu} x_i^{(n)}/g \right\rfloor 2^{i\nu}$, где ν бит берется из каждого рекуррентного соотношения.

В таблице указаны параметры генераторов, основанных на автоморфизме двумерного тора, и описанных в работах [2, 29, 30]. Параметры подбирались та-

Генератор	Параметры				
	k	q	g	ν	Период
GM19	15	28	$2^{19}-1$	1	$2{,}7 \cdot 10^{11}$
GM31	11	14	$2^{31}-1$	1	$4{,}6 \cdot 10^{18}$
GM61	24	74	$2^{61}-1$	1	$5{,}3 \cdot 10^{36}$
GM29.1	4	2	$2^{29}-3$	1	$2{,}8 \cdot 10^{17}$
GM55.4	256	176	$16(2^{51}-129)$	4	$\geqslant 5{,}1 \cdot 10^{30}$
GQ58.1	8	48	$2^{29}(2^{29}-3)$	1	$\geqslant 2{,}8 \cdot 10^{17}$
GQ58.3	8	48	$2^{29}(2^{29}-3)$	3	$\geqslant 2{,}8 \cdot 10^{17}$
GQ58.4	8	48	$2^{29}(2^{29}-3)$	4	$\geqslant 2{,}8 \cdot 10^{17}$

ким образом, чтобы для генератора выполнялось свойство равнораспределения вероятностей в размерности порядка логарифма величины g, а также чтобы период генератора был максимально возможным [30]. Точное равнораспределение наблюдается в размерности порядка 30. Практически такие генераторы ведут себя близко к равнораспределению в размерности близкой к 350. Генераторы имеют большой период.

3. Параллельные потоки псевдослучайных чисел

Основными требованиями, предъявляемыми к генераторам случайных чисел, являются следующие: статистическая устойчивость и отсутствие корреляций; большое значение периода генерируемой последовательности; эффективность реализации генератора; наличие теории, предсказывающей свойства генератора; воспроизводимость генерируемой последовательности; переносимость на разные программно-аппаратные платформы; возможность быстрого пропуска кусков генерируемой последовательности; наличие методов правильной инициализации (см., например, [31]). Генерация параллельных потоков псевдослучайных чисел для использования при вычислениях методом Монте-Карло на суперкомпьютерных системах имеет дополнительное требование — возможность генерации и одновременного использования большого числа параллельных потоков псевдослучайных чисел. Потоки псевдослучайных чисел должны быть некоррелированы между собой, иначе статистика, которая получается при решении задачи методом Монте-Карло, будет смещена. Этого можно добиться за счет подходящей инициализации псевдослучайных последовательностей. Теоретически можно провести представленную далее классификацию методов [32].

Случайный выбор начальных величин. Для генерации потоков используют один и тот же алгоритм PRNG. Каждый из потоков при этом имеет начальное значение, полученное с помощью другого генератора, PRNG или TRNG. У такого подхода нет обоснования и он основан на ожидании, что начальные значения будут соответствовать точкам последовательности, разведенным на большое расстояние. Применения такого "оптимистического" метода следует избегать.

Параметризация. Для генерации потоков используют один и тот же алгоритм PRNG, но с разными наборами параметров. К числу таких параметров для ЛКМ относятся различные приращения c или различные множители a в выражении (1), для ГСР это длина сдвигового регистра k в выражении (2). Следует отметить, что метод параметризации имеет слабое теоретическое обоснование. Для ЛКМ эмпирические тесты выявили серьезные корреляции между потоками [33]. Для ГСР число возможных потоков при использовании такого метода ограничено наборами известных параметров (см., например, [34]).

Расщепление блока. Пусть M — максимальное число вызовов генератора псевдослучайных чисел у одиночного процесса, p — число процессов. Выходная последовательность генератора псевдослучайных чисел нарезается на p последовательных блоков длины M. Необходим эффективный алгоритм для перехода к началу каждого из p блоков.

Чехарда. Метод параллелизации чехардой основан на том, что каждый из процессов, используя одно из значений генератора, каждый раз пропускает следующие p значений, где p — число процессов. Необходим эффективный алгоритм для "пропуска кусков".

Для целей генерации параллельных некоррелированных потоков псевдослучайных чисел, которые преследует настоящая работа, больше подходят методы расщепления блока и чехарды.

Процесс генерации случайных чисел должен учитывать особенности вычислительного процесса, использующего псевдослучайные числа. При этом важен учет аппаратной архитектуры, которая накладывает жесткие условия как на проведение вычислительного процесса исследовательской задачи, так и на реализацию процесса генерации псевдослучайных чисел.

Классификация реализации вычислительного процесса методом Монте-Карло может быть проведена при помощи одновременного учета способов реализации вычислительного процесса исследовательской задачи и способов реализации процесса генерации псевдослучайных чисел. По результатам рассмотрения различных вариантов для числа нитей вычислительного процесса, числа нитей процесса генерации псевдослучайных чисел, а также числа потоков псевдослучайных чисел, можно сделать вывод, что базовыми являются разработка эффективной реализации процесса генерации одного потока псевдослучайных чисел с использованием одной нити и одного потока псевдослучайных чисел с использованием N нитей. Любая реализация потоков псевдослучайных чисел для вычислительного процесса Монте-Карло может быть легко сведена к применению этих базовых вариантов. При этом может потребоваться передача к вычислительному процессу одного псевдослучайного числа, а также блока из N псевдослучайных чисел.

4. Генерация параллельных потоков псевдослучайных чисел с использованием GPGPU

Предлагаемый в настоящей работе подход состоит в использовании таких алгоритмов, для которых: доказаны хорошие свойства псевдослучайной последовательности; существует эффективный алгоритм инициализации параллельных потоков таким образом, чтобы псевдослучайные числа в потоках не коррелировали между собой; существует эффективная реализация на GPU общего назначения (GPGPU).

4.1. Эффективные алгоритмы пропуска кусков и инициализации генераторов GM19, GM31, GM61, GM29.1, GM55.4, GQ58.1, GQ58.3, GQ58.4

Пропуск кусков проводится при помощи следующих соображений. Пусть для некоторых целых n, k_n и q_n и для всех целых неотрицательных w имеет место соотношение $x^{(2n+w)} = k_n x^{(n+w)} - q_n x^{(w)}(\mathrm{mod} g)$. Например, из выражения (3) следует, что при $n = 1$ соотношение выполняется для $k_1 = k$, $q_1 = q$. Тогда $x^{(4n)} = k_n x^{(3n)} - q_n x^{(2n)}(\mathrm{mod} g) = k_n\left(k_n x^{(2n)} - q_n x^{(n)}\right) - 2q_n x^{(2n)} + q_n x^{(2n)}(\mathrm{mod} g) = \left(k_n^2 - 2q_n\right)x^{(2n)} - q_n^2 x^{(0)}(\mathrm{mod} g)$. Отсюда следует, что соотношение выполняется для $k_1 = k$; $q_1 = q$; $k_{2n} = k_n^2 - 2q_n(\mathrm{mod} g)$; $q_{2n} = q_n^2(\mathrm{mod} g)$. Это позволяет быстро вычислить коэффициенты, необходимые для пропуска кусков с числом элементов, равным степени двойки.

Эффективным способом пропуска куска произвольной длины n является разложение числа n в двоичную запись вида $n = 2^{i_0} + 2^{i_1} + ... + 2^{i_m}$ и последовательный пропуск кусков длин 2^{i_0}, 2^{i_1}, ..., 2^{i_m}. Можно также использовать следующие формулы:

$$k_0 = 2;\ k_1 = k;\ k_{n+1} = kk_n - qk_{n-1}(\mathrm{mod} g);$$
$$q_n = q^n(\mathrm{mod} g).$$

Докажем эти выражения по индукции. Пусть $x^{(2i)} = k_i x^{(i)} - q_i x^{(0)}(\mathrm{mod} g)$ для всех $i = 0, 1,..., n$. Тогда $x^{(2n)} = k_{n-1}x^{(n+1)} - q_{n-1}x^{(2)} = k_n x^{(n)} - q_n x^{(0)}(\mathrm{mod} g)$. Следовательно, $x^{(2n+2)} = k_n x^{(n+2)} - q_n x^{(2)} = kk_n x^{(n+1)} - qk_n x^{(n)} - q_n x^{(2)} = (kk_n - qk_{n-1})x^{(n+1)} + qk_{n-1}x^{(n+1)} - qk_n x^{(n)} - q_n x^{(2)} = k_{n+1}x^{(n+1)} - q_{n+1}x^{(0)}(\mathrm{mod} g)$, что и требовалось доказать. Используя выведенные здесь выражения для k_n, q_n, можно эффективно проводить пропуск кусков в выходной последовательности генератора.

Инициализация параллельных потоков осуществляется для генератора GQ58 при помощи следующих функций:

```
__device__ void GQ58_init_short_sequence(GQ58_state* state,unsigned SequenceNumber){
GQ58_init(state); // 0 <= SequenceNumber < 3*10^8; length of each sequence <= 8*10^7
GQ58_SkipAhead(state,0,82927047ULL*(unsigned long long)SequenceNumber);
}
__device__ void GQ58_init_medium_sequence(GQ58_state* state,unsigned SequenceNumber){
GQ58_init(state); // 0 <= SequenceNumber < 3*10^6; length of each sequence <= 8*10^9
GQ58_SkipAhead(state,0,8799201913ULL*(unsigned long long)SequenceNumber);
}
__device__ void GQ58_init_long_sequence(GQ58_state* state,unsigned SequenceNumber){
GQ58_init(state); // 0 <= SequenceNumber < 3*10^4; length of each sequence <= 8*10^11
GQ58_SkipAhead(state,0,828317697521ULL*(unsigned long long)SequenceNumber);
}
```

Функция GQ58_init_short_sequence позволяет инициализировать $3 \cdot 10^8$ последовательностей случайных чисел длины, не превосходящей $8 \cdot 10^7$; GQ58_init_medium_sequence позволяет инициализировать $3 \cdot 10^6$ последовательностей длины, не превосходящей $8 \cdot 10^9$; GQ58_init_long_sequence позволяет инициализировать $3 \cdot 10^4$ последовательностей длины, не превосходящей $8 \cdot 10^{11}$. Параллелизация проводится методом расщепления блока. Для пропуска соответствующих кусков используется функция GQ58_SkipAhead.

Несколько более сложной является процедура инициализации параллельных потоков для таких генераторов как GM55.4 и GM61. Дело в том, что период этих генераторов является настолько большим, что позволяет провести параллельную генерацию очень большого числа очень длинных последовательностей, поэтому простое умножение значения номера последовательности (*SequenceNumber*) на длину блока в методе расщепления блока может дать значение вплоть до 2^{128}, что превосходит прямые возможности 64-битной арифметики. Ниже показано, как можно эффективно реализовать инициализацию параллельных потоков случайных чисел методом расщепления блока для генератора GM61.

```
__device__ void GM61_init_sequence(GM61_state* state,lt SequenceNumber){
lt n1,n2; // length of each sequence < 10^10, 0 <= SequenceNumber < 4*10^18
GM61_init(state);
n1=SequenceNumber/892447987; n2=SequenceNumber%892447987;
GM61_SkipAhead(state,n1,n1*4193950067);
GM61_SkipAhead(state,0,n2*20669825409);
// thus we are skipping ahead (SequenceNumber*20669825409) numbers
}
__device__ void GM61_init_long_sequence(GM61_state* state,lt SequenceNumber){
// 0 <= SequenceNumber < 4*10^9, length of each sequence < 3*10^25
GM61_init(state);
GM61_SkipAhead(state,2000000*SequenceNumber,2699204111*SequenceNumber);
}
```

Функция GM61_init_sequence позволяет инициализировать $4 \cdot 10^{18}$ последовательностей случайных чисел длины, не превосходящей 10^{10}; GM61_init_long_sequence позволяет инициализировать $4 \cdot 10^{9}$ последовательностей случайных чисел длины, не превосходящей $3 \cdot 10^{25}$. При реализации функции GM61_init_sequence используется число $A = 892447987$ и находятся числа n_1, n_2 такие, что

$$SequenceNumber = A \cdot n_1 + n_2.$$

Длина блока в методе расщепления блока в данном случае равна $L = 20669825409$. При этом

$$SequenceNumber \cdot L =$$
$$= (A \cdot n_1 + n_2) \cdot L = (2^{64} + 4\ 193\ 950\ 067) \cdot n_1 + L \cdot n_2.$$

Здесь каждое из чисел $4\ 193\ 950\ 067 \cdot n_1$ и $L \cdot n_2$ не превосходит 2^{64}, поскольку $L \cdot (A - 1) < 2^{64}$. Этот факт позволяет двумя вызовами функции GM61_SkipAhead осуществить необходимый пропуск куска. При реализации функции GM61_init_long_sequence используем длину блока, превосходящую $3 \cdot 10^{25}$: $L = n_1 \cdot 2^{64} + n_2$, где $n_1 = 2\ 000\ 000$, $n_2 = 2\ 699\ 204\ 111$. При этом

$$SequenceNumber \cdot L =$$
$$= (SequenceNumber \cdot n_1) \cdot 2^{64} + (SequenceNumber \cdot n_2),$$

и, поскольку каждое из чисел *SequenceNumber*, n_1, n_2 в данном случае не превосходит 2^{32}, то необходимый пропуск куска можно выполнить одиночным вызовом функции GM61_SkipAhead.

4.2. Инициализация и генерация параллельных потоков для генератора MRG32K3A

Алгоритм генерации псевдослучайных чисел MRG32K3A можно записать следующим образом:

$$x_n = (ax_{n-2} + bx_{n-3}) (\mathrm{mod} m_1),$$
$$y_n = (cy_{n-1} + dy_{n-3}) (\mathrm{mod} m_2),$$
$$z_n = (x_n + y_n) (\mathrm{mod} m_1),$$

где $a = 1\ 403\ 580$, $b = -810\ 728$, $c = 527\ 612$, $d = -1\ 370\ 589$, $m_1 = 2^{32} - 209$, $m_2 = 2^{32} - 22\ 853$. Выходной последовательностью генератора является $\{z_0, z_1, ...\}$.

Чтобы описать метод генерации параллельных потоков псевдослучайных чисел и пропуска кусков для генератора MRG32K3A, удобно обозначить $\mathbf{X}_n = \begin{pmatrix} x_n \\ x_{n-1} \\ x_{n-2} \end{pmatrix}$, $\mathbf{Y}_n = \begin{pmatrix} y_n \\ y_{n-1} \\ y_{n-2} \end{pmatrix}$, что позволяет представить метод MRG32K3A в следующем виде:

$$\mathbf{X}_{n+1} = \mathbf{A}\mathbf{X}_n (\mathrm{mod} m_1),$$
$$\mathbf{Y}_{n+1} = \mathbf{B}\mathbf{Y}_n (\mathrm{mod} m_2).$$

Здесь $\mathbf{A} = \begin{pmatrix} 0 & a & b \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, $\mathbf{B} = \begin{pmatrix} c & 0 & d \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$. Состоянием генератора является пара векторов $\mathbf{X}_n$, $\mathbf{Y}_n$. Следовательно, для пропуска куска длиной n необходимо найти n-ую степень матриц $\mathbf{A}$ и $\mathbf{B}$. Это можно сделать, разлагая число n в двоичную запись вида $n = 2^{i_0} + 2^{i_1} + ... + 2^{i_m}$, а затем вычисляя матрицы $\mathbf{A}$, $\mathbf{A}^2 (\mathrm{mod} m_1)$, $\mathbf{A}^4 (\mathrm{mod} m_1)$, ..., $\mathbf{A}^{2^{i_m}} (\mathrm{mod} m_1)$, $\mathbf{B}$, $\mathbf{B}^2 (\mathrm{mod} m_2)$, $\mathbf{B}^4 (\mathrm{mod} m_2)$, ..., $\mathbf{B}^{2^{i_m}} (\mathrm{mod} m_2)$ и используя равенства $\mathbf{A}^n = \mathbf{A}^{2^{i_0}} \mathbf{A}^{2^{i_1}} ... \mathbf{A}^{2^{i_m}} (\mathrm{mod} m_1)$, $\mathbf{B}^n = \mathbf{B}^{2^{i_0}} \mathbf{B}^{2^{i_1}} ... \mathbf{B}^{2^{i_m}} (\mathrm{mod} m_2)$. Таким образом, пропуск куска длины n может быть проведен за $O(\log_2 n)$ операций.

Ниже показано, как проводится инициализация параллельных потоков для генератора MRG32K3A:

```
__device__ void MRG32K3A_init_sequence(MRG32K3A_state* state,unsigned long long SequenceNumber){
// 0 < SequenceNumber < 10^19; length of each sequence <= 10^38
unsigned long long a0, a32, b0=1029448176, b32=1712288060, b64=2074794764, b96=1914833239,t32,t96;
a32=SequenceNumber>>32; a0=SequenceNumber-(a32<<32);
MRG32K3A_init(state);
t32=a32*b0+a0*b32; t96=a32*b64+a0*b96;
MRG32K3A_SkipAhead(state,a32*b96,a32*b32+a0*b64,a0*b0);
MRG32K3A_SkipAhead(state,t96>>32,t32>>32,0);
MRG32K3A_SkipAhead(state,0,(t96^4294967295ULL)<<32,(t32^4294967295ULL)<<32);
}
```

Функция MRG32K3A_init_sequence позволяет инициализировать до 10^{19} независимых параллельных последовательностей случайных чисел длины, не превосходящей 10^{38}. Для пропуска кусков используется функция MRG32K3A_SkipAhead (MRG32K3A_state* state, unsigned long long offset128, unsigned long long offset64,unsigned long long offset0), работающая по методу, изложенному выше. Как видно из алгоритма, умножение *SequenceNumber* и константы $b = b_0 + 2^{32} b_{32} + 2^{64} b_{64} + 2^{96} b_{96}$, которая является размером блока в методе расщепления блока, осуществляется разбиением чисел на 32-битовые блоки и аккуратным учетом каждого слагаемого.

4.3. Инициализация и генерация параллельных потоков для генератора LFSR113

Генератор LFSR113 является комбинацией четырех следующих сдвиговых регистров:

$$x_n = x_{n-31} \oplus x_{n-25},\ s = 18,$$

$$x_n = x_{n-29} \oplus x_{n-27},\ s = 2,$$

$$x_n = x_{n-28} \oplus x_{n-15},\ s = 7,$$

$$x_n = x_{n-25} \oplus x_{n-22},\ s = 13.$$

Алгоритм пропуска куска для комбинированного генератора сводится к пропуску кусков для каждого из этих сдвиговых регистров. Каждый из этих сдвиговых регистров можно записать в виде

$$x_n = (x_{n-p} + x_{n-p+q})(\mathrm{mod}2),\ u_n = \Sigma_{l=1}^{32} x_{is+l-1} 2^{-l}.$$

Как было замечено в работе [10], если число k является степенью двойки: $k = 2^e$, то для сдвигового регистра $x_n = (x_{n-p} + x_{n-p+q})(\mathrm{mod}2)$ истинно выражение $x_n = (x_{n-kp} + x_{n-kp+kq})(\mathrm{mod}2)$. Действительно, для $k = 2$ имеем $x_n = x_{n-p} \oplus x_{n-p+q} = x_{n-2p} \oplus x_{n-2p+q} \oplus \oplus x_{n-p+q} = x_{n-2p} \oplus x_{n-2p+q} \oplus x_{n-2p+q} \oplus x_{n-2p+2q} = x_{n-2p} \oplus x_{n-2p+2q}$. Продолжая далее точно также по индукции, выражение $x_n = (x_{n-kp} + x_{n-kp+kq})(\mathrm{mod}2)$ обосновывается для любого $k = 2^e$.

Пусть теперь заданы числа $x_0, x_1, x_2, ..., x_{31}$. Построим таблицу, в n-й строке которой находятся следующие p чисел: $x_{2^n p}, x_{2^n(p+1)}, ..., x_{2^n(2p-1)}$. Строки таблицы нумеруются начиная с нуля. В нулевой строке также находятся p чисел, соответствующие $n = 0$, которые вычисляются напрямую из известных чисел $x_0, x_1, x_2, ..., x_{31}$. Все следующие числа в такой таблице вычисляются из предыдущих чисел таблицы согласно выражению $x_{2^n(p+l)} = x_{2^n l} \oplus x_{2^n(l+q)}$, которое сразу следует из доказанного выше выражения. Заметим, что для $n \geqslant 5$ число x_{2^n} обязательно принадлежит таблице, поскольку $32 \in \{p, p+1, ..., 2p-1\}$ для каждого из четырех рассматриваемых сдвиговых регистров. По этой причине алгоритм пропуска куска длины 2^n для комбинированного генератора LFSR113 сводится к построению предложенных таблиц для каждого из четырех сдвиговых регистров. Пропуск куска произвольной длины сводится к пропуску кусков длин, равных степени двойки, точно также, как и для генераторов, пропуск кусков которых описан выше в предшествующих подразделах.

Период генератора LFSR113 приблизительно равен 2^{113}, из рассмотренных выше генераторов этот период наиболее близок к периодам генераторов GM55.4 и GM61. Функции инициализации последовательности случайных чисел с номером *SequenceNumber* оказывается возможным реализовать аналогично соответствующим функциям генераторов GM55.4 и GM61. Функция LFSR113_init_sequence позволяет инициализировать $4 \cdot 10^{18}$ независимых последовательностей случайных чисел длины, не превосходящей 10^{10}; LFSR113_init_long_sequence позволяет инициализировать $4 \cdot 10^9$ независимых последовательностей случайных чисел длины, не превосходящей $3 \cdot 10^{25}$.

4.4. Инициализация и генерация параллельных потоков для генератора MT19937

Общий алгоритм генерации для MT19937 подробно описан в работе [18]. Алгоритм работы генератора MT19937 имеет линейную структуру и может быть представлен в виде $\mathbf{Y}_{n+1} = \mathbf{A}\mathbf{Y}_n$, $\mathbf{Y}_n$ — *состояние генератора*, а арифметические операции проводятся по модулю 2. По этой причине простейший алгоритм пропуска кусков для данного генератора заключается в нахождении матрицы $\mathbf{A}^n$. Однако для этого генератора матрица $\mathbf{A}$ имеет размер $19\,937 \times 19\,937$, поэтому вычисление $\mathbf{A}^n$ будет исключительно медленным и потребует значительного объема памяти, даже для случая, когда n является степенью двойки. Другой способ, предложенный в работе [35], основан на арифметике многочленов в поле $\mathbb{F}_2$. Авторы работы показывают, что для любого $\nu \in \mathbb{N}$ истинно выражение

$$\mathbf{A}^{\nu}\mathbf{Y}_0 = g_{\nu}(\mathbf{A})\mathbf{Y}_0 = a_k\mathbf{Y}_{k-1} + a_{k-1}\mathbf{Y}_{k-2} + ... + a_2\mathbf{Y}_1 + a_1\mathbf{Y}_0, \quad (4)$$

где $k = 19\,937$, коэффициенты $a_i \in \{0, 1\}$, $i = 1, ..., k$, и полином $g_{\nu}(x) = a_k x^{k-1} + ... + a_2 x + a_1$ в поле $\mathbb{F}_2$ зависит от ν. В работе [35] приведен способ вычисления g_{ν} для произвольного фиксированного ν. Это вычисление является однозадачным вычислением, в котором массивный параллелизм не будет полезен. Вычисление является довольно затратным по времени (несколько миллисекунд на современном CPU для вычисления коэффициентов одного полинома, что соответствует времени, достаточному для генерации примерно 13 млн случайных чисел) и поэтому не должно проводиться "на лету". В отличие от имеющихся ранее подходов, в предлагаемом нами алгоритме заранее вычисляются коэффициенты полинома g_{ν} для следующих значений ν: $\nu = n \cdot 2^{21}$, $n = 1, 2, ..., 512$; $\nu = 2^n$, $n = 0, 1, 2, ..., 512$. Все эти коэффициенты помещаются в память графического процессора, в которой для размещения каждого из 1024 рассматриваемых полиномов выделяется 2496 байт. При таком подходе для пропуска куска длиной ν достаточно вычислить на графическом процессоре выражение (4). Вычисление при этом проводится с использованием массивного параллелизма графического процессора. Если ν можно представить в виде $\nu = n \cdot 2^{21}$ для $n \leqslant 512$, тогда коэффициенты будут заранее известны и вычисление будет наиболее эффективным. Если этого не удается сделать,

то коэффициенты известны для полиномов g_{2^n}, n = = 0, 1, 2, ..., 512. При этом пропуск куска длиной v сводится к пропуску кусков длин, равных степеням двойки, соответствующих разложению числа v в двоичную запись.

Период последовательности MT19937, равный $2^{19937} - 1 \approx 4{,}3 \cdot 10^{6001}$, позволяет инициализировать при помощи метода расщепления блока любое необходимое для расчетов по методу Монте-Карло число независимых потоков случайных чисел. Длина каждого из потоков может быть выбрана приблизительно равной 10^{100}.

4.5. Интерфейс вызова для однонитевой генерации потока псевдослучайных чисел с использованием GPGPU

Интерфейс вызова однонитевых реализаций на языке программирования NVIDIA CUDA C представляется следующим образом:

__device__ void RNG_init (RNG_state* state);

__device__ void RNG_init_sequence (RNG_state* state, unsigned SequenceNumber);

__device__ void RNG_SkipAhead (RNG_state* state, unsigned long long offset);

__device__ unsigned int RNG_generate (RNG_state* state);

__device__ float RNG_generate_uniform_float (RNG_state* state).

Аббревиатура RNG в названии каждой функции заменяется на название одного из 11 упомянутых генераторов, а именно — GM19, GM31, GM61, GM29.1, GM55.4, GQ58.1, GQ58.3, GQ58.4, MRG32K3A, LFSR113 и MT19937 (см. разд. 2). Более подробная информация также изложена в работе [31], где подробно обсуждаются технические детали устройства всех 11 генераторов и их реализация в командах CPU. Здесь RNG_state — структура, в которой хранится состояние этого генератора. Функция RNG_init инициализирует последовательность псевдослучайных чисел. Начальные параметры автоматически подбираются функцией RNG_init одним из рекомендуемых для конкретного генератора способов. Функция RNG_init_sequence инициализирует последовательность псевдослучайных чисел. Таких последовательностей может быть очень много, и их инициализация проводится так, чтобы последовательности с разными значениями *SequenceNumber* не коррелировали между собой. Максимальное значение *SequenceNumber*, т. е. максимальное число независимых последовательностей, может достигать 10^{19}, например, для параллельных алгоритмов генераторов MRG32K3A или GM55.4 (см. подразд. 4.1 и 4.2). Функций RNG_init_sequence может быть несколько. Они могут отличаться друг от друга максимальным разрешенным значением *SequenceNumber* и максимальной длиной каждой последовательности. Функция RNG_SkipAhead проводит пропуск куска длины offset в реализуемом генераторе случайных чисел для произвольного заданного offset. Вызов этой функции меняет состояние генератора state таким же образом, как оно поменялось бы после генерации случайных чисел в количестве offset. Для некоторых генераторов параметры функции RNG_SkipAhead будут более сложными. Например, реализована функция со следующим интерфейсом, которая позволяет для генератора MRG32K3A проводить пропуск, в частности, очень длинных кусков вплоть до длины 2^{196}: __device__ void MRG32K3A_SkipAhead (MRG32K3A_state* state, unsigned long long offset128, unsigned long long offset64, unsigned long long offset0). Функция RNG_generate проводит генерацию случайного 32-битного целого числа средствами одной нити графического процессора. Функция RNG_generate_uniform_float осуществляет генерацию случайного вещественного числа, равномерно распределенного на интервале [0, 1), средствами одной нити графического процессора. Множество таких функций, генерирующих случайные числа, могут одновременно выполняться в разных нитях системы для параллельной генерации независимых потоков случайных чисел.

4.6. Интерфейс вызова для многонитевой генерации потока псевдослучайных чисел с использованием GPGPU

Интерфейс вызова многонитевых реализаций на языке NVIDIA CUDA C следующий:

void RNG_initialize (RNG_state* state);

void RNG_initialize_sequence (RNG_state* state, unsigned SequenceNumber);

void RNG_skip_ahead (RNG_State* state, unsigned long long offset);

void RNG_generate_array (RNG_state* state, unsigned int * out, unsigned int length);

void RNG_generate_uniform_float_array (RNG_state* state, float * out, unsigned int length).

Аббревиатура RNG в названии каждой функции заменяется на название одного из 11 упомянутых выше генераторов. Эти функции вызываются из CPU. Функции RNG_generate_array и RNG_generate_uniform_float_array задействуют производительные возможности графического процессора для быстрой генерации массива случайных чисел длины length. Процедуры RNG_initialize, RNG_initialize_sequence, RNG_skip_ahead выполняют те же функции, что и RNG_init, RNG_init_sequence, RNG_SkipAhead, которые были описаны в подразд. 4.5. Отличие в том, что они вызываются из CPU и могут использовать всю производительность графического процессора для простой инициализации, для быстрой и правильной инициализации независимой последовательности случайных чисел с номером *SequenceNumber*, а также для быстрого пропуска кусков последовательности, соответственно.

Заключение

В настоящей работе рассмотрены алгоритмы генерации случайных чисел с использованием возможностей графических процессоров для выбранных авторами современных и наиболее надежных генераторов

случайных чисел. Разработанный интерфейс для генерации псевдослучайных чисел позволяет эффективно использовать разработанные методы генерации случайных чисел в приложениях, использующих графические ускорители, в том числе в расчетах Монте-Карло на гибридных суперкомпьютерных системах. За счет использования массивного параллелизма может быть достигнута существенно бóльшая эффективность работы генерации случайных чисел, по сравнению как с однонитевыми реализациями, так и с эффективными алгоритмами генерации случайных чисел для CPU.

Настоящая работа выполнена по проекту, выполняемому в рамках ФЦП "Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007—2013 годы", государственный контракт 07.514.11.4032 с Министерством образования и науки.

Список литературы

1. **Binder K., Heermann D. W.** Monte Carlo Simulation in Statistical Physics. Berlin: Springer-Verlag, 1992. 358 p.

2. **Barash L. Yu., Shchur L. N.** RNGSSELIB: Program library for random number generation, SSE2 realization // Computer Physics Communications. 2011. Vol. 182. P. 1518—1527.

3. **Reidler I., Aviad Y., Rosenbluh M., Kanter I.** Ultrahigh-Speed Random Number Generation Based on a Chaotic Semiconductor Laser // Physical Review Letters. 2009. Vol. 103. 024102. P. 1—4.

4. **Кнут Д. Э.** Искусство программирования. 2-е изд. М.: Вильямс, 2007. 824 с.

5. **Shchur L.** On the quality of random number generators with taps // Computer Physics Communications. 2009. Vol. 121—122. P. 83—85.

6. **Coveyou R. R. and MacPherson R. D.** Fourier Analysis of Uniform Random Number Generators // Journal of the ACM. 1967. Vol. 14. P. 100—119.

7. **Marsaglia G.** Random numbers fall mainly in the planes // Proceedings of the National Academy of Sciences USA. 1968. Vol. 61. P. 25—28.

8. **Ferrenberg A. M., Landau D. P., Wong Y. J.** Monte Carlo simulations: Hidden errors from "good" random number generators // Physical Review Letters. 1992. Vol. 69. P. 3382—3384.

9. **Lemer D. H.** Mathematical methods in large-scale computing units // Proceedings of the 2nd Symposium on Large-Scale Digital Calculating Machinery, Cambridge, MA, 1951. P. 141—146.

10. **Golomb S. W.** Shift Register Sequences // San Francisco: Holden-Day, 1967. 172 p.

11. **Selke W., Talapov A. L. and Shchur L. N.** Cluster-flipping Monte-Carlo algorithm and correlations in "good" random number generators // Письма в ЖЭТФ. 1993. Т. 58. С. 684—686.

12. **Grassberger P.** On correlations in "good" random number generators // Physics Letters A. 1993. Vol. 181. P. 43—46.

13. **Vattulainen I., Ala-Nissila T., Kankaala K.** Physical Tests for Random Numbers in Simulations // Physical Review Letters. 1994. Vol. 73. P. 2513—2516.

14. **Schmid F., Wilding N. B.** Errors in Monte Carlo Simulations Using Shift Register Random Number Generators // International Journal of Modern Physics C. 1995. Vol. 6. P. 781—797.

15. **Shchur L. N., Heringa J. R., Bloete H. W. J.** Simulation of a directed random-walk model: the effect of pseudo-random-number correlations // Physica A. 1997. Vol. 241. P. 579—592.

16. **Shchur L. N., Bloete H. W. J.** Cluster Monte Carlo: Scaling of Systematic Errors in 2D Ising Model // Physical Review E. 1997. Vol. 55. P. R4905—R4908.

17. **Lusher M.** A portable high-quality random number generator for lattice field theory calculations // Computer Physics Communications. 1994. Vol. 79. P. 100—110.

18. **Matsumoto M. and Nishimura T.** Mersenne twister: A 623-dimensionally uniform pseudo-random number generator // ACM Trans. Mod. Comp. Sim. 1998. Vol. 8. P. 3—30.

19. **L'Ecuyer P.** Good parameters and implementations for combined multiple recursive random number generators // Operations Research. 1999. Vol. 47. P. 159—164.

20. **L'Ecuyer P.** Maximally equidistributed combined Tausworthe generators // Mathematics of Computation. 1996. Vol. 65. P. 203—213.

21. **L'Ecuyer P.** Tables of maximally equidistributed combined LFSR generators // Mathematics of Computation. 1999. Vol. 68. P. 261—269.

22. **L'Ecuyer P.** Uniform Random Number Generation // Annals of Operations Research. 1994. Vol. 53. P. 77—120.

23. **Barash L. Yu.** Geometric and statistical properties of pseudo-random number generators based on multiple recursive transformations // Springer Proceedings in Mathematics and Statistics. 2012. Vol. 23. P. 265—280.

24. **Couture R., L'Ecuyer P., Tezuka S.** On the distribution of k-dimensional vectors for simple and combined Tausworthe sequences // Mathematics of Computation 1993. Vol. 60. P. 749—761.

25. **Fushimi M., Tezuka S.** The k-distribution of generalized feedback shift register pseudorandom numbers // Communications of the ACM 1983. Vol. 26. P. 516—523.

26. **Tootil J. P. R., Robinson W. D., Eagle D. J.** An Asymptotically Random Tausworthe Sequence // Journal of the ACM. 1973. Vol. 20. P. 469—481.

27. **Shchur L. N., Butera P.** The RANLUX generator: Resonances in a random walk test // International Journal Modern Physics C. 1998. Vol. 9. P. 607—624.

28. **L'Ecuyer P., Simard R.** TestU01: A C library for empirical testing of random number generators // ACM TOMS. 2007. Vol. 33. Article 22.

29. **Barash L., Shchur L. N.** Periodic orbits of the ensemble of Sinai-Arnold cat maps and pseudorandom number generation // Physical Review E. 2006. Vol. 73. P. 036701/1—14.

30. **Barash L. Yu.** Applying dissipative dynamical systems to pseudorandom number generation: Equidistribution property and statistical independence of bits at distances up to logarithm of mesh size // Europhysics Letters. 2011. Vol. 95. P. 10003.

31. **Щур Л. Н., Бараш Л. Ю.** Генерация случайных чисел и параллельных потоков случайных чисел для расчетов Монте-Карло // Моделирование и анализ информационных систем. 2012. Т. 19. С. 145—162.

32. **Bauke H., Mertens S.** Random numbers for large-scale distributed Monte Carlo simulations // Physical Review E. 2007. Vol. 75. P. 066701/1-14.

33. **Matteis A. D. and Pagnutti S.** A class of parallel random number generators // Parallel Computing. 1990. Vol. 13. P. 193—198.

34. **Бараш Л. Ю.** Алгоритм AKS проверки чисел на простоту и поиск констант генераторов псевдослучайных чисел // Безопасность информационных технологий. 2005. № 2. С. 27—38.

35. **Haramoto H., Matsumoto M., Nishimura T., Panneton F., L'Ecuyer P.** Efficient jump ahead for $\mathbb{F}_2$-linear random number generators // INFORMS Journal on Computing. 2008. Vol. 20. P. 385—390.

УДК 004.057, 519.8

М. А. Бульонков[1,2], канд. физ.-мат. наук, зав. лаб., e-mail: mike@iis.nsk.su,
П. Г. Емельянов[1,2], канд. физ.-мат. наук, стар. науч. сотр., e-mail: emelianov@iis.nsk.su,
Е. В. Пак[1], вед. программист, e-mail: pev@iis.nsk.su,
А. А. Харенко[2], магистрант, e-mail: anna.kharenko@gmail.com,
[1]Институт систем информатики СО РАН, Лаборатория смешанных вычислений,
[2]Новосибирский государственный университет

Моделирование данных в задаче составления расписаний в высших учебных заведениях

В связи с развитием общества, информационных и образовательных технологий задача составления расписаний все более усложняется, причем не только в части ее алгоритмики, но и в части описания данных, которые при этом используются. Авторами предпринята попытка систематизировать накопленный опыт в этой области и предложить язык описания данных для задачи составления расписаний в высших учебных заведениях.

Ключевые слова: *задача составления расписаний, моделирование ограничений, описание учебного плана, форматы данных, язык разметки XML*

1. Глобальные тенденции в сфере высшего образования

Система высшего образования и организация научных исследований являются основой инновационного процесса развития экономики любого государства. По этой причине любой стране, претендующей на достойное место в современном мире, необходима национальная система высшего образования, которая не только бы адекватно отражала национальные приоритеты, но и учитывала бы глобальные мировые тренды интеграции, взаимопроникновения культур и достижений научно-технического прогресса. В связи с этим следует отметить, что различные системы подготовки квалифицированных кадров, будь то североамериканская, европейская или российская [3], имеют ряд общих тенденций, основными из которых являются перечисленные ниже.

- Рост числа студентов и высших учебных заведений различного профиля, а также увеличение возраста студентов. Первый факт связан с увеличением зависимости получения хорошо оплачиваемой работы от уровня профессиональной подготовки нанимаемого работника, а второй — с требованием обновления знаний в современном меняющемся мире, необходимостью получения знаний в смежных областях.
- Усложнение управления процессом обучения. В частности, это обусловлено как ростом числа работников, вовлеченных в организацию функционирования системы высшего образования, так и увеличением вариативности процесса обучения.
- Повсеместная компьютеризация образования. Распространение глобальных информационных сетей фактически ликвидировало границы в сфере потоков информации. Таким образом, образование оказалось в ситуации, когда источником получения обучающей информации стали не только обычные лекции и практические занятия, но и глобальные информационные ресурсы. В связи с этим обстоятельством все большее распространение получает развитие дистанционного обучения.
- Унификация системы высшего образования, выражающаяся, в частности, в широком внедрении многоуровневой системы подготовки "бакалавр-магистр" и использовании международных рекомендаций по формированию учебных программ (Curricula). Это влечет за собой необходимость выработки единых критериев сравнения и оценки качества обучения в высших учебных заведениях.
- Академическая мобильность, под которой понимается перемещение студентов в другие учебные заведения (как индивидуально, так и в составе кол-

лективов) для расширения обучения по определенным направлениям подготовки или проведения практических научных исследований. Эта тенденция вытекает из необходимости учитывать инновационность современного производства и связанной с ней диверсификацией подготовки специалистов.

Новые задачи выдвигают на первый план многоуровневые высшие учебные заведения, которые и должны обеспечить доступ к профессиональному образованию различного уровня максимально широкого круга населения, в частности, за счет реализации индивидуальных образовательных траекторий. Эти учебные заведения призваны обеспечивать возможность реализации принципа "обучение в течение всей жизни", а также способствовать развитию как горизонтальной, так и вертикальной мобильности студентов, предоставляя возможность переходить с одного образовательного уровня на другой или изменять образовательную траекторию в рамках одного уровня.

Для новых форм образовательных учреждений характерны размывание обязательных сроков обучения, использование таких форм как дистанционное обучение или обучение с преимущественным использованием электронных средств и сети Интернет. Кроме того, новые формы получения знаний должны обеспечить связи между различными профессиональными траекториями за счет модульного обучения и введения в практику смешанных курсов и блоков дисциплин. Из изложенного выше вытекает значительная вариативность организации самого процесса обучения.

2. Современные методы обучения и их влияние на процесс составления расписания

К традиционным методам обучения (лекции, практические занятия и семинары) в настоящее время добавляются и новые, вносящие существенный вклад в формирование компетенции. К их числу относятся исследовательские семинары, творческие семинары (так называемые мастерские), демонстрационные занятия, тренинги и деловые игры, дискуссии, стажировки, электронное обучение.

Важно отметить, что оценка полученных знаний и умений должна рассматриваться как основной элемент управления процессом обучения. Помимо традиционных зачетов и экзаменов появляются и новые методы оценки результатов обучения, например, проверка в виде теоретического и практического тестирования, устная презентация, реферат или эссе, различного рода отчеты, демонстрация полученных практических навыков на рабочем месте, письменные обзоры литературы и отзывы.

В последнее время [2] существует тенденция к видоизменению традиционных форм проведения занятий, например, лекций. Некоторые вузы для стимулирования более эффективных форм подачи лекционного материала студентам предоставляют в рамках одной и той же учебной дисциплины право выбора лектора самим студентам. Кроме того, все шире и шире преподаватели используют различные формы наглядного представления материала с использованием специального оборудования. Доступ к специализированному аудиторному фонду может иметь ограничения как по времени, так и по вместимости. Это обстоятельство приводит к более сложному разделению аудиторного фонда между различными факультетами, кафедрами, учебными курсами, что, естественно, усложняет составление расписания занятий.

Другая новая форма лекций — лекции вдвоем. Они могут происходить в форме диалога двух преподавателей, представляющих различные научные школы. Такая своеобразная форма проведения занятий, очень полезная для студентов, может также внести дополнительные трудности при составлении расписания ввиду усложнения ограничений. Расширением данного типа лекции может быть занятие в виде пресс-конференции с участием нескольких преподавателей. Такого рода занятия могут быть достаточно регулярными, но нечастыми. Они могут объединять лекции по одному предмету на разных потоках, "сливая" тем самым несколько регулярных занятий в одно, "искривляя" при этом основное расписание в плане занятости преподавателей и аудиторий.

В современном образовательном процессе появляются новые формы практических и семинарских занятий. Более того, в новых российских стандартах [3] они становятся одним из важных инструментов развития деятельностного (активностного) компонента компетенций. Они могут проходить в виде дискуссий или круглых столов, а также мини-конференций. Одним из основных направлений развития творческих способностей студентов является так называемое "проблемное" обучение — индивидуальный и/или коллективный процесс решения нестандартных учебных или исследовательских задач. В западных вузах существуют специальные аудитории (комнаты мозгового штурма), которые используются для этих целей. Соответственно, и формы проведения этих занятий могут быть такими же нестандартными. Наиболее известными формами активного обучения можно считать ролевые и деловые игры.

Самостоятельная работа студента, важность которой особенно подчеркивается в современных образовательных стандартах [3], не может быть эффективна без развития системы контроля результатов этой работы со стороны преподавателей. Это приводит к значительному числу консультационных и проверочных занятий, особенностью которых может быть неравномерное распределение этих занятий в течение семестра. Более того, процедуры контроля могут стать многоуровневыми и дробными. Самостоятельная работа студентов, требующая специальной инструментальной поддержки, приведет к "росту давления" на такие разделяемые образовательные ресурсы как терминальные классы, лаборатории, мастерские и т.д.

Особого внимания заслуживает проблема индивидуализации образования (личная образовательная траектория) в современных вузах. Эта форма образовательного процесса подразумевает в первую очередь составле-

ние индивидуальных планов обучения. Самостоятельная работа может быть включена в учебный план и расписание занятий с организацией соответствующих индивидуальных консультаций на кафедрах вуза.

Таким образом, усложнение технологии преподавания и формирования компетенций — системы форм, методов и средств обучения, обеспечивающих получение наиболее эффективного результата, — влечет за собой значительное усложнение процесса составления расписания учебных занятий в высших учебных заведениях.

3. Современные исследования в области формализации описания учебного процесса

Выработка единого формата описания учебного процесса имеет цель создания единой основы для решения описанных ниже технологических, административных и научных задач.

- Обмен данными между разными системами, в том числе в среде Semantic Web. В частности, наличие единого общепризнанного формата данных способствовало бы развитию конкуренции между системами составления расписаний, а значит совершенствованию их качества.
- Проверка в автоматическом или полуавтоматическом режиме соответствия государственным и международным образовательным стандартам, государственным санитарным нормам, а также количественного оценивания "качества" составленного расписания.
- Создание общезначимых тестовых наборов данных для систем автоматического составления расписания, которые бы также давали возможность сравнивать различные системы.

За последние годы за рубежом было сделано несколько попыток разработки форматов данных для описания учебного процесса. Следует отметить, что на начальном этапе они в основном были направлены на то, чтобы формат покрывал как можно больше типов ограничений, и были ориентированы на системы автоматического составления расписаний (тестирование производительности и оценка качества результатов). Краткий обзор состояния исследований к 2000 г., сложившиеся классификации, а также язык описания ограничений UniLang, конвертируемый в логический язык программирования в ограничениях ECLiPse, представлены в работе [12]. В работе [4] предложен основанный на теории множеств и математической логике объектно-ориентированный язык спецификаций ограничений, с помощью которого может быть определена любая вычислимая функция. Далее эти идеи получили развитие в объектно-ориентированном функциональном языке STTL [8], в котором возможно моделирование не только задачи составления учебного расписания, но и более общих сложных задач о назначениях. Кроме того, объектно-ориентированное моделирование исследовалось в работе [7]. В работе [9] предложено расширение языка разметки математических формул MathML средствами описания ограничений довольно общего вида — языка TTML. Следствием универсальности этих стандартов является сложность их структуры и "многословность" при описании реальных данных.

Так как главной задачей при разработке данного стандарта авторы считают адекватность моделирования учебного процесса и интероперабельность, то в качестве наиболее близких по духу можно указать работы [5, 6]. В них не предложен завершенный стандарт, но указана важность выработки единого понятийного аппарата и унификации требований для задачи составления расписаний. Авторы работ [10, 11], проанализировав образовательные системы Австралии, Англии, Греции, Нидерландов и Финляндии, выражают надежду, что их работа и предложенная XML-схема послужат базой для создания общепринятого стандарта для представления данных для составления расписаний. Заявленная цель — совершенствование взаимодействия различных программных систем — являлась одним из побудительных мотивов и нашей работы [1], развитие которой мы представляем в данной статье.

4. Базовые понятия и средства моделирования

При составлении расписания учебное заведение оперирует некоторыми понятиями, которые описываются и связываются друг с другом в данной модели при помощи вспомогательных объектов. Все используемые объекты можно разбить на две группы:

- объекты планирования, которые непосредственно участвуют в составлении расписания;
- объекты описания структуры учебного заведения.

Наиболее важными с точки зрения составления расписания являются объекты планирования, а именно учебный план и назначения, критические разделяемые ресурсы и время. Далее эти понятия описываются более подробно.

4.1. Учебный план и назначения

Учебный план (PlanItem) определяет состав *учебных дисциплин* (Subject), изучаемых в данном учебном заведении в рамках определенного направления подготовки, и их распределение по годам в течение всего срока обучения. Исходя из учебного плана строится расписание занятий, представленное в виде *назначений* (Assignment). Назначение закрепляет три вида ресурсов: студенты-слушатели, преподаватель (один или несколько), аудитория (одна или несколько) на определенный промежуток времени для проведения занятия, описанного в учебном плане.

В определенных случаях возникает потребность наложить некоторые *ограничения* (Constraint) непосредственно на конкретное назначение. Ограничения могут быть следующих видов:

- *временные ограничения,* которые используют для фиксации конкретного назначения в определенном временном интервале;

• *пространственные ограничения,* используемые для территориальной фиксации конкретного назначения; для этого указывается либо перечень желаемых (или строго определенных) мест (*зданий* или *аудиторий*), либо наоборот — недопустимых.

4.2. Критические разделяемые ресурсы

Критический разделяемый ресурс — это ресурс, который может быть назначен только одному событию в каждый конкретный момент времени. По некоторым причинам ресурс может быть недоступен. В таких случаях на ресурсы могут быть наложены как временные, так и пространственные, а также количественные ограничения.

Описываемая модель предусматривает три основных вида ресурсов, описанных ниже.

• *Преподаватели*. Данный вид ресурсов имеет, как правило, большое число ограничений. Более того, основная часть всех ограничений, задаваемых в системе, приходится на преподавателей. Это связано с тем, что профессорско-преподавательский состав в современных вузах включает большое число совместителей, которые заняты в различных учебных заведениях, а также с тем, что многие преподаватели работают на разных факультетах.

• *Аудитории*. Аудитории различают по вместимости и функциональной принадлежности (лекционные, семинарские, лаборатории), наличию специального оборудования (компьютеров, проекторов и пр.), закрепленности за кафедрами и факультетами, а также территориальной сгруппированности. В большинстве случаев в процессе планирования именно этот ресурс является наиболее востребованным.

• *Студенты*. Для удобства при планировании студенты объединяются в *студенческие группы* (см. подразд. 5.2). В свою очередь, студенческие группы могут быть объединены или разбиты на подгруппы для распределения студентов в *коллекции слушателей* (см. подразд. 6.2). Коллекция слушателей является инструментом описания аудиторного состава для каждого занятия. При формировании коллекции слушателей можно указывать не только любое множество групп и подгрупп, но и любое подмножество группы, с использованием *разбиений на подгруппы* (SubGrouping).

4.3. Время

Время события (Time), характеризующее временные свойства ограничений и назначений, может быть описано как в астрономических (абсолютных) единицах, так и в логических (относительных) единицах. Для этого используют три компонента: *начало события* (UniDate), *продолжительность* (UniTime) и *периодичность* (UniPeriod). Используя данные способы описания, можно указать любой промежуток времени с любой периодичностью (в том числе разовые назначения). В работе [9] для временных отметок предложен формат, который позволяет указать время вплоть до секунд.

Следует отметить, что наличие возможности гибкого указания продолжительности и периодичности занятий является ключевым моментом в современных системах составления расписания. Это связано с тем, что современный учебный процесс подразумевает значительное число занятий с относительно небольшим общим количеством часов (специальные лабораторные занятия, выступления приглашенных лекторов и т.д.), а также блочной организацией учебных курсов с довольно произвольными временными интервалами (неделями, триместрами и т.д.). Кроме того, заочная форма обучения требует назначения занятий строго по часам учебной нагрузки без периодичности, но в заданный календарный интервал, а, например, дистанционная форма обучения может подразумевать проведение аудиторных мероприятий или сеансов связи с преподавателем в конкретные дни или календарные даты без указания в расписании конкретного времени. Продуманная система представления времени назначения может существенным образом повлиять на удобство интерфейса, предоставляемого конечному пользователю, что, в свою очередь, определит полноту функциональности всей системы.

Для описания начала события указывается дата начала по календарному времени — месяц и число, или по логическому времени — номер недели и день недели. Время начала также может быть указано по календарному (астрономическому) или относительному времени (номер пары).

Для фиксации временного конца события вводится понятие продолжительности, которая может определяться как по календарному (в часах), так и по относительному (в парах) времени.

В расписании занятий присутствуют события, имеющие различную периодичность:

• стабильные занятия (регулярные занятия, проводимые в одно и то же время каждую неделю);

• периодические занятия, которые могут проводиться время от времени (например, через неделю или всего несколько раз в семестр);

• разовые события (например, экзамены, консультации);

• занятия, неравномерно распределенные в течение учебного времени (например, "недели начитки").

Для предоставления возможности описывать события различной периодичности вводится понятие периода, где указывается периодичность (например, раз в две недели) и число повторений в течение учебного времени (семестра, триместра).

4.4. Насколько общей является рассматриваемая задача

Следует отметить, что, с одной стороны, стоит задача построения как можно более универсального формата представления данных, а с другой стороны, некоторые используемые концепции на данный момент в него не включены и их включение неочевидно.

Примером первого является рассмотрение задачи составления расписания с максимальным числом степеней свободы планирования. Хотя в некоторых образовательных системах [10] сложность планирования понижают за счет связывания некоторых объектов планирования, например, аудитории и/или часы могут быть жестко закреплены за преподавателями или коллективами слушателей. Обычно это существует в системах с преобладанием штатных преподавателей с высокой учебной нагрузкой.

Примером второго может служить то, что на данный момент при составлении расписаний не рассматриваются разделяемые ресурсы типа мобильного оборудования (магнитофоны, ноутбуки, мобильные проекторы и т.д.). Вполне вероятно, что за счет своего удешевления такого рода ресурсы станут массовыми, и для них никогда не потребуется специального планирования.

5. Модель учебного заведения

При описании модели учебного заведения используются понятия, хорошо известные в объектно-ориентированном и реляционном моделировании, такие как класс, структурное отношение (агрегирование), неструктурное отношение (ассоциация) и наследование.

Каждое учебное заведение имеет свою специфику организации учебного процесса. Необходимо учитывать такие особенности, как число недель в семестре (триместре), количество дней в учебной неделе, количество пар в учебном дне. Не менее важной особенностью является дата начала учебного процесса, так как в зависимости от нее происходит пересчет времени из учебного в календарное, учет праздничных дней, которые выпадают на учебное время, и корректировка расписания занятий. По этой причине в данной модели учебное заведение включает в себя все перечисленные выше понятия (рис. 1). В учебно-административную структуру учебного заведения (University) входят:

- учебные корпуса (Building), в которых проводятся занятия, в том числе и отдельные аудитории (Room), входящие в состав корпуса;
- кафедры (Chair), которые объединяют научно-преподавательский состав (Teacher) и набор родственных учебных дисциплин (Subject);
- факультеты (Faculty), осуществляющие подготовку студентов и аспирантов по одной или нескольким родственным специальностям и руководство научно-исследовательской деятельностью кафедр, которые факультет объединяет;
- студенты одного факультета и одного курса, объединенные в группы (Group).

5.1. Учебные корпуса и аудитории

Основной задачей, связанной с учебными корпусами, является расчет времени перемещения между ними и учет этих данных в расписании. Эта задача может быть решена внесением в модель сущности, содержащей информацию о взаимной удаленности корпусов и времени перемещения между ними. Однако

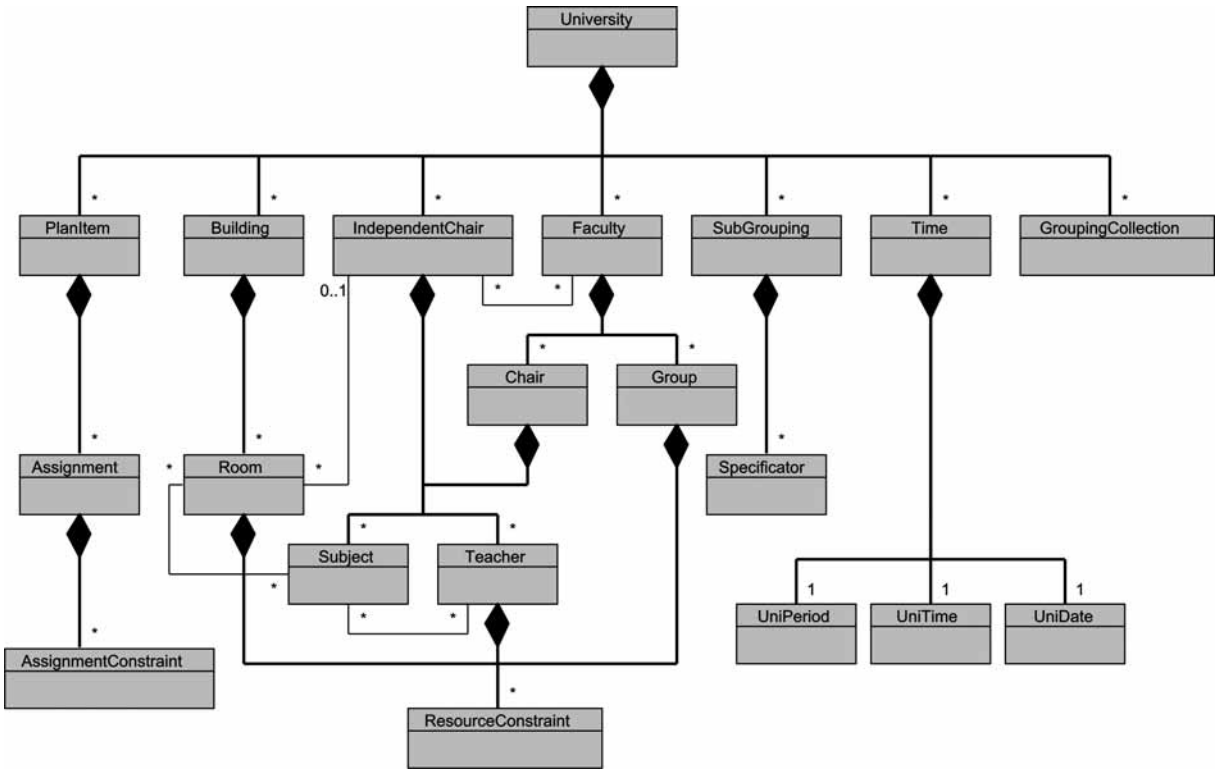

Рис. 1. Диаграмма "Модель учебного заведения"

окончательное решение, связанное с этой задачей, еще не принято.

Каждый учебный корпус имеет набор аудиторий, которые характеризуются вместимостью (максимальное число человек) и типом аудитории. Вместимость аудитории используется во время принятия решения о назначении занятия в данной аудитории. Достаточность мест для конкретной коллекции слушателей определяется с учетом численности студенческой группы/подгруппы или суммарной численности студентов с учетом заданных разбиений. Тип аудитории определяет возможные виды занятий, например, лекции или лабораторные, проводимые в ней, а также наличие специального оборудования. Отметим, что не все форматы описания данных поддерживают понятие специализации аудитории. В работе [12], например, явно указывается, что это довольно редкая ситуация для составления расписаний. Это утверждение является достаточно спорным, так как очевидно, что для вузов естественнонаучного и технического профиля проведение занятий в специализированных лабораториях со стационарным оборудованием является неотъемлемой частью учебного процесса.

5.2. Факультеты и студенческие группы

Учебное заведение подразделяется на факультеты, которые осуществляют подготовку студентов по одной или нескольким родственным специальностям или направлениям подготовки и руководство деятельностью выпускающих кафедр. Кафедры могут принадлежать конкретному факультету или быть независимыми (общеуниверситетские кафедры — IndependentChair). Поэтому в данной модели нет жесткой привязки кафедр к факультетам. Следует отметить, что с точки зрения составления расписаний понятие факультета лишнее, хотя, несомненно, он играет существенную административно-научную роль в процессе обучения.

В рамках факультета студенты объединяются в студенческие группы, которые характеризуются понятием численности. Понятие численности используется в дальнейшем при назначении занятия в аудиторию. Также группа может иметь набор ограничений, описывающих отсутствие возможности у студентов группы находиться в какое-либо время в каком-то месте по некоторым причинам, что учитывается при назначении занятий.

5.3. Кафедры, преподаватели и учебные дисциплины

Другой взгляд на учебное заведение — это совокупность кафедр, которые осуществляют подготовку студентов и аспирантов в рамках определенной специализации. Каждая кафедра имеет свой преподавательский состав и набор преподаваемых учебных дисциплин. Обычно преподаватель работает с одной кафедрой, однако встречаются случаи, когда преподаватель совмещает работу на нескольких кафедрах. Другой существующий вариант — дисциплину, закрепленную за конкретной кафедрой, ведет преподаватель с другой кафедры, не являясь при этом внутренним совместителем. Данная модель подразумевает возможность описания подобных случаев.

Информация о преподавателе в зависимости от потребности в конкретном учебном заведении может содержать достаточно большой объем сведений: полное имя, контактная информация, ученая степень, ученое звание, должность и т.д. Эти данные могут оказаться важными для составления различного рода отчетов. Преподаватель может иметь набор ограничений, описывающих отсутствие возможности (или пожелание) проводить занятия в какое-либо время или в каком-то месте. Такие ограничения естественно должны быть учтены при составлении расписания занятий.

У каждого преподавателя существует свой набор учебных дисциплин, которые он преподает в данном учебном заведении на данной кафедре (или нескольких кафедрах). Очень часто встречаются ситуации, когда несколько преподавателей ведут одну и ту же учебную дисциплину у разных групп, либо у всех групп, но различные виды занятий. Существуют дисциплины (или типы занятий), для которых необходимо какое-либо специальное оборудование. В таких случаях для каждой дисциплины (или типу занятий по данной дисциплине) можно указать либо типы аудиторий, где ее можно преподавать, либо набор конкретных аудиторий.

6. Модель учебного процесса

Детальность описания учебного процесса — важная характеристика формата данных, предназначенного для составления расписаний. Если с его помощью невозможно выразить существенные для планирования понятия, то в случае ручного составления расписания нагрузка перераспределяется на человека — составителя расписания (он вынужден задействовать "бизнес-логику", заложенную в его голове, тогда как хотелось бы перепоручить часть этого машине), а в случае автоматического составления полученное расписание может оказаться совершенно непригодным к использованию (возможно, оно и будет составлено очень быстро, но число коллизий превзойдет все разумные пределы; опыт авторов показывает, что ручная доработка автоматически составленного расписания является нетривиальной задачей). Излишняя детализация, во-первых, будет требовать поддержки на всех уровнях (хранение, визуализация, редактирование и т.д.) данных, которые, возможно, не являются общезначимыми ("артефакты" отдельно взятого вуза), а во-вторых, может превратить автоматическое составление качественного расписания в чрезвычайно трудную задачу.

6.1. Учебные планы и назначения

Учебный план по направлению подготовки или специальности (PlanItem) — это перечень дисциплин с указанием их объема в академических часах и распределением этих часов по неделям, семестрам, учебным годам, а также число часов, отводимых на лекции,

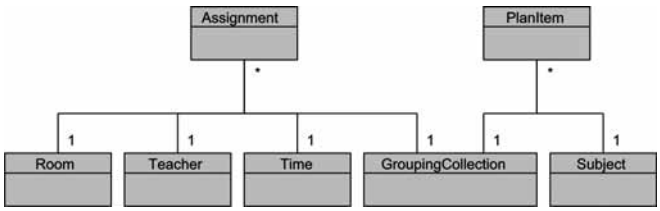

Рис. 2. Диаграмма "Учебные планы и назначения"

семинары, лабораторные работы и т.д. Для вузов РФ учебный план определяется государственными стандартами. Для распределения учебного плана по пространственно-временной шкале (составление расписания) необходимо предварительно распределить контингент студентов на группы (возможно с разделением на подгруппы), определить периодичность различного типа занятий в соответствии с объемом академических часов для конкретного учебного периода и назначить преподавателей для заданных дисциплин. Каждое назначение (Assignment) распределяет во времени и пространстве определенную часть учебного плана. Оно закрепляет три вида ресурсов, а именно студентов (коллекцию слушателей, см. подразд. 6.2), преподавателя (одного или нескольких) и аудиторию (одну или несколько), на определенный промежуток времени для проведения занятия, описанного в учебном плане. На рис. 2 дано описание структуры и связей для PlanItem и Assignment.

Следует отметить, что после того как расписание составлено, возникает несколько задач проверки его корректности, в частности, соответствие суммарного числа часов, фигурирующих в расписании занятий, учебному плану, а также проверка отсутствия различных накладок.

Эти основные задачи решаются в рамках данной модели.

6.2. Коллекции слушателей, фильтры и разбиения на подгруппы

Очень часто занятия по учебным дисциплинам назначаются не для конкретной группы, а для нескольких групп или целого потока, или же наоборот, только для одной подгруппы группы. Для таких случаев удобно ввести понятие *Коллекция слушателей* (GroupingCollection), с помощью которой можно описать такой набор студентов. Следует отметить, что все эти понятия и манипуляции с ними легко излагаются и доказываются в теоретико-множественных терминах путем введения дополнительных объектов (например, множества всех студентов вуза). Однако, с одной стороны, дополнительные объекты являются искусственными для исходной задачи, а с другой стороны, авторам хотелось бы избежать вопросов, которые обсуждались в работе [4]. Их суть в том, что чрезвычайно выразительный язык привел к появлению алгоритмических трудностей, которые, быть может, не встречаются в реальной практике составления расписаний, но, тем не менее, потенциально возможны.

Диаграмма, представленная на рис. 3, описывает правила, по которым может быть составлена коллекция слушателей. Таким образом, получим рекурсивную формулу построения коллекции. Она может представлять собой объединение других коллекций слушателей и групп или фильтр (Filter), накладываемый на другие коллекции слушателей и группы с использованием некоторого способа (Specificator) разбиения группы на подгруппы:

$$GroupingCollection = \\ = \begin{bmatrix} Group, \\ GroupingCollection.Specificator, \\ GroupingCollection + GroupingCollection. \end{bmatrix}$$

Таким образом, можно указать подгруппу группы (например, "932.1" — подгруппа "1" группы "932"), или студентов группы, обучающихся на кафедре (например, "9208.СИ" — студенты группы "9208", обучающиеся на кафедре Систем информатики), или студентов по любому другому признаку (например, "9208.Девушки" — девушки, обучающиеся в группе "9208"). Фильтр можно накладывать не только на группы, но и на другие коллекции слушателей (например, "(9208 + 9209).Юноши" — юноши, обучающиеся в группах "9208" и "9209"). В качестве коллекции слушателей также может выступать объединение групп (например, "9208 + 9209" — объединение групп "9208" и "9209") или коллекций слушателей (например, "(931 + 932 + 933).СИ + (9208 + + 9209).СИ" — объединение коллекций слушателей "(931 + 932 + 933).СИ" и "(9208 + 9209). СИ"). Отметим, что в теоретико-множественных терминах это соответствует операциям объединения и пересечения, при этом операция дополнения не используется. Разбиение группы на подгруппы (SubGrouping) представляет собой перечисление элементов разбиения (Specificator), имеющих название и некоторый комментарий (если требуется), которые используются в дальнейшем в фильтрах.

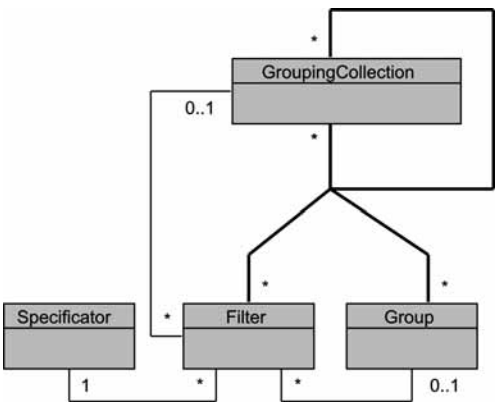

Рис. 3. Диаграмма "Коллекции слушателей"

7. Ограничения

Неотъемлемой частью моделирования учебного процесса является описание разного рода ограничений. Например, преподаватель может провести занятие только в первой половине дня или в конкретный день недели. Другой пример — требуется освободить аудиторию в конкретный интервал времени, или данную пару желательно провести в конкретном корпусе (или аудитории). Использование понятия *Ограничение* (Constraint) (рис. 4) дает возможность описать условия, при которых возможно провести назначение.

Общее понятие ограничения распадается на два класса: ограничения, накладываемые на ресурсы планирования (ResourseConstraint), и ограничения, накладываемые на назначения — результаты планирования (AssignmentConstraint).

Примеры ограничений, которые могут накладываться на группы, преподавателей, аудитории и назначения:

- необходимость (или отсутствие возможности) проведения занятий в какой-либо промежуток времени;
- необходимость (или отсутствие возможности) проведения занятий в каком-либо месте;
- максимальное число пар в день;
- максимальное число пар подряд (в том числе по одной дисциплине);
- максимальное число окон в день.

Примеры ограничений, которые могут накладываться на назначения:

- проведение лекционных занятий перед практическими занятиями;
- обязательное проведение консультаций перед экзаменами;
- чтение блока лекций перед блоком практики;
- ограничения, накладываемые на какое-то конкретное занятие (например, проведение в строго определенный день).

Примеры ограничений, накладываемые на аудитории:

- отсутствие возможности проведения занятий в данной аудитории в какой-то промежуток времени;
- невозможность (или необходимость) проведения в данной аудитории определенных видов занятий;
- проведение в данной аудитории занятий для определенных коллекций слушателей (например, студентов конкретного направления подготовки);
- санитарные часы для аудитории (подготовка оборудования или лабораторного материала);
- строгое ограничение числа учащихся (например, лингафонные или компьютерные классы).

Следовательно, ограничения можно разбить таким образом:

- ограничения, накладываемые на ресурсы: временные (TimeConstraintToResource), пространственные (PlaceConstraintToResource), количественные (QuantitativeConstraintToResource);
- ограничения, накладываемые на назначения: временные (TimeConstraintToAssignment), пространственные (PlaceConstraintToAssignment).

Каждое ограничение имеет приоритет, это позволит давать рекомендации диспетчеру, какие занятия желательно расставить в первую очередь.

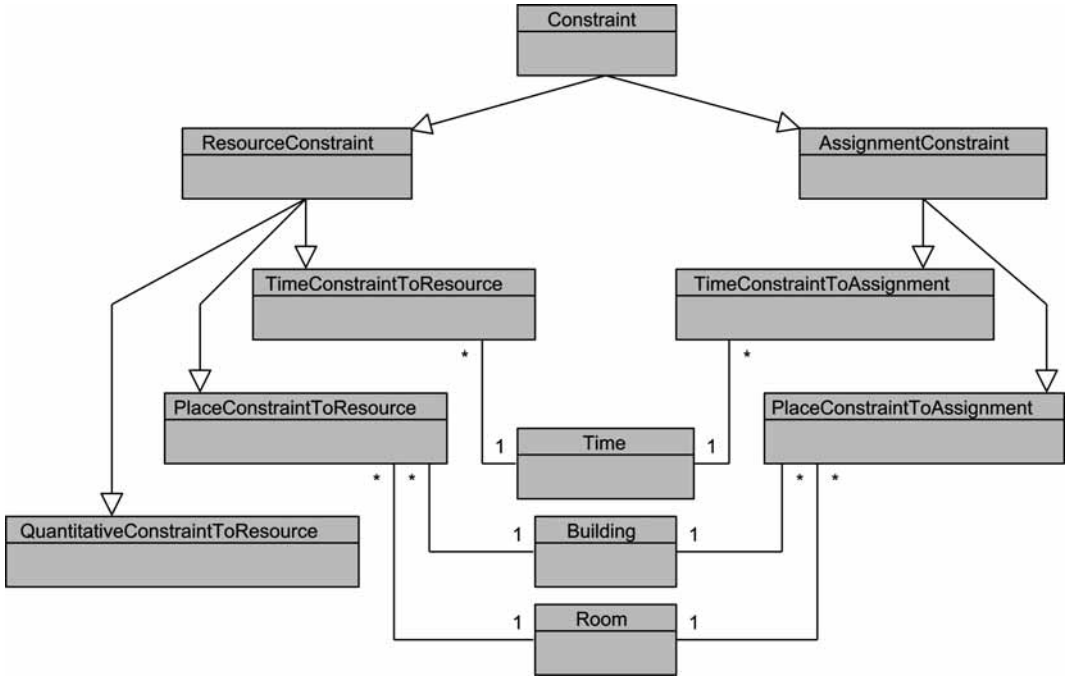

Рис. 4. Диаграмма "Ограничения"

* * *

Авторы полагают, что данный подход к описанию учебного процесса и расписания в вузах в достаточно полной мере отражает современное состояние образовательных технологий.

Описанная модель, на данный момент уже имеющая представление в формате XML, должна лечь в основу создаваемой системы составления расписания, которая будет разрабатываться на языке Java с использованием GWT (Google Web Toolkit). Формат XML обеспечивает удобство обмена данными между компонентами системы и предоставляет широкий набор средств манипулирования данными. Поэтому в разрабатываемой системе авторы планируют использовать данный формат в качестве способа хранения данных.

В процессе разработки (возможно уже на этапе прототипирования) в представлении модели могут возникнуть иные решения или видоизмениться предложенные. Попытка представить как можно более полно особенности реализации предписаний учебного плана на практике может оказаться все-таки недостаточно исчерпывающей. Вместе с тем она может, наоборот, затруднить реализацию или снизить ее эффективность, например, чрезмерно усложнить пользовательский интерфейс. Опыт в области использования автоматизированных систем подготовки расписаний показывает, что именно интерфейс, предоставляемый пользователю, является наиболее проблемной частью данного рода систем. Его удобство, простота и интуитивная понятность непосредственно связаны с заинтересованностью диспетчеров бюро расписаний в отказе от ручного способа составления расписания в пользу его автоматизированного аналога.

Список литературы

1. **Бульонков М. А., Емельянов П. Г., Пак Е. В.** К стандартизации описания учебного процесса в учебных заведениях // Открытое образование. 2010. № 3. С. 45—57.

2. **Педагогика** и психология высшей школы: учеб. пос. / под ред. М. В. Булановой-Топорковой. Ростов н/Д: Феникс, 2002. 544 с.

3. **Сайт** Федерального агентства по образованию. Государственные образовательные стандарты. URL: http://www.edu.ru/db/portal/spe/index.htm

4. **Burke E. K., Kingston J. H., Pepper P. A.** A standard data format for timetabling instances // Selected papers from the Second International Conference on Practice and Theory of Automated Timetabling (PATAT 1997, Toronto). Lecture Notes in Computer Science. Vol. 1408. Berlin: Springer-Verlag, 1998. P. 213—222.

5. **De Causmaecker P., Demeester P., Lu Y., Vander Berghe G.** Using Web standards for timetabling // Selected papers from the Fourth International Conference on Practice and Theory of Automated Timetabling (PATAT 2002, Gent). Lecture Notes in Computer Science. Vol. 2740. Berlin, Heidelberg: Springer-Verlag, 2003. P. 238—257.

6. **De Causmaecker P., Custers N., Demeester P., Vanden Berghe G.** Semantic components for timetabling // Selected papers from the Fifth International Conference on Practice and Theory of Automated Timetabling (PATAT 2004, Pittsburgh). Lecture Notes in Computer Science. Vol. 3616. Berlin, Heidelberg: Springer-Verlag, 2005. P. 17—33.

7. **Gr̈bner M., Wilke P., B̈ttcher S.** A standard framework for timetabling problems // Selected papers from the Fourth International Conference on Practice and Theory of Automated Timetabling (PATAT 2002, Gent). Lecture Notes in Computer Science. Vol. 2740. Berlin, Heidelberg: Springer-Verlag, 2003. P. 24—38.

8. **Kingston J. H.** Modelling timetabling problems with STTL // Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling (PATAT 2000, Konstanz). Lecture Notes in Computer Science. Vol. 2079. Berlin: Springer-Verlag, 2001. P. 309—321.

9. **Özcan E.** Towards an XML based standard for timetabling problems: TTML // In Multidisciplinary Scheduling: Theory and Applications / Eds. Kendall G. et al. Selected papers of the First International Conference MISTA 2003, Nottingham. New-York: Springer-Verlag, 2005. P. 163—187.

10. **Post G., Ahmadi S., Daskalaki S., Kingston H., Kyngas J., Ranson D., Ruizenaar H.** An XML format for benchmarks in high school timetabling // Annals of Operations Research. 2010. Vol. 179. № 1. P. 1—13.

11. **Ranson D., Ahmadi S.** An extensible modelling framework for the examination timetabling problem // In Proceedings of the Sixth International Conference of the Practice and Theory of Automated Timetabling (PATAT 2006, Brno). Lecture Notes in Computer Science. Vol. 3867. Berlin, Heidelberg: Springer-Verlag, 2007. P. 383—393.

12. **Reis L. P., Oliveira E.** A language for specifying complete timetabling problems // Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling (PATAT 2000, Konstanz). Lecture Notes in Computer Science. Vol. 2079. Berlin: Springer-Verlag, 2001. P. 322—341.

ИНФОРМАЦИЯ

УДК 519.681

М. Д. Шурлин, аспирант, Воронежский государственный университет,
e-mail: mshurlin@gmail.com

Редукция LP-структур для автоматизации рефакторинга в объектно-ориентированных системах

Рассматривается класс основанных на решетках алгебраических структур, описывающих семантику иерархии типов в объектно-ориентированной системе. Исследуются свойства таких структур, включая существование логической редукции. Методология предназначена для верификации и модернизации иерархий типов, важным направлением которой является устранение избыточности кода. Впервые приводятся доказательства полученных результатов.

Ключевые слова: *иерархия типов, рефакторинг, алгебраическая модель, совмещение атрибутов*

Введение

Алгебраические структуры предоставляют основу формального построения и исследования информационных систем, основанных на различных технологиях [1, 2]. Это относится и к распространенным на практике системам продукционного типа [3]. В ряде работ (см. библиографию в работе [4]) предложена методология алгебраизации продукционных систем на основе решеток и отношений. Получены результаты для обоснования верификации и оптимизации таких систем. Решетка с заданным на ней дополнительным продукционным отношением названа LP-структурой (*lattice production structure*).

Исследования показали применимость данной методологии в различных областях информатики. В работе [5] впервые установлено, что отношения обобщения и агрегации типов обладают свойствами продукционно-логического вывода. В результате построен класс LP-структур для моделирования иерархий типов в целях рефакторинга — модернизации кода. В этих LP-структурах из решеточных операций использовалось лишь объединение, что ограничивало возможности теории формализацией единственного метода рефакторинга — поднятия общих атрибутов (обзор известных методов содержится в работе [6]). Статья [7] развивает теорию LP-структур для иерархий типов. В результате "инвертирования" модели [5] получен новый метод рефакторинга. Его суть состоит в замене нескольких атрибутов класса общим потомком (совмещение атрибутов). Идеи статьи [7] получили продолжение в работе [8], однако доказательства сформулированных результатов опубликованы еще не были.

Настоящая работа завершает построение алгебраической модели, описанное в работах [7, 8]. Для данного вида LP-структур рассматривается ряд вопросов, основной из которых — логическая редукция и способ ее построения. Впервые представлены доказательства полученных результатов.

Решение родственных задач методами анализа формальных понятий (FCA) описано в работе [9]. Элементам множества классов предлагается в некотором смысле оптимально назначить наборы атрибутов — элементов другого независимого множества. В соответствии с выбранными назначениями формируется иерархия классов. В постановке настоящей работы, в отличие от работы [9], атрибуты сами относятся к исследуемой иерархии классов (типов), что усложняет задачу и не оставляет возможности непосредственного применения методов FCA.

1. Основные понятия и обозначения

Бинарное отношение R на множестве F называется:

- рефлексивным, если для всех $a \in F$ справедливо $(a, a) \in R$;
- транзитивным, если из $(a, b), (b, c) \in R$ следует $(a, c) \in R$.

Существует замыкание R^* произвольного отношения R относительно свойств рефлексивности и транзитивности — рефлексивно-транзитивное замыкание (РТЗ). Пара элементов $a, c \in F$ называется транзи-

тивной в R, если $(a, c) \in R_1^*$, где R_1^* — РТЗ отношения $R_1 = R\backslash\{(a, c)\}$.

Обратная задача — нахождение транзитивной редукции: по заданному R отыскивается минимальное отношение R' такое, что его РТЗ совпадает с РТЗ для R [10]. Как обычно, для частично упорядоченных множеств различаются понятия минимального элемента (для него нет меньшего элемента) и наименьшего элемента (он меньше всех). В работе [10] представлен алгоритм построения транзитивной редукции конечного отношения, показано, что эта задача вычислительно эквивалентна построению РТЗ.

Необходимые сведения о решетках содержатся в работе [11]. Решеткой называется частично упорядоченное множество $\mathbb{F}$, в котором наряду с отношением $\leqslant$ ("не больше", "содержится") определены две двуместные операции $\wedge$ (пересечение) и $\vee$ (объединение), вычисляющие соответственно точную нижнюю и верхнюю грани любой пары $a, b \in \mathbb{F}$. Решетка называется ограниченной, если она содержит общие верхнюю и нижнюю грани — такие два элемента O, I, что $O \leqslant a \leqslant I$ для любого $a \in \mathbb{F}$.

2. Обсуждение задач и приложений

Рассмотрим иерархию типов $\mathbb{F}$ в объектно-ориентированной программной системе. Между парами типов могут существовать как минимум два вида связей — наследование (тип наследует атрибуты типа-предка) и агрегация (тип содержит в качестве атрибута представителя другого типа) [6].

Отношение наследования порождает на $\mathbb{F}$ частичный порядок: если тип b является потомком a, то $b \leqslant a$. Требуется, чтобы для любых $a, b \in \mathbb{F}$ были определены две решеточные операции: пересечение $a \wedge b$ — наибольший общий потомок; объединение $a \vee b$ — наименьший общий предок a, b (первая операция актуальна в системах с множественным наследованием). Для ограниченности решетки добавим к $\mathbb{F}$ два специальных элемента: I — универсальный тип (общий предок, существует в ряде современных систем программирования) и O — фиктивный потомок всех типов.

На решетке $\mathbb{F}$ рассмотрим второе, соответствующее агрегации, отношение R: если экземпляр типа a в качестве атрибута содержится в определении типа b, то $(b, a) \in R$. Оба отношения ($\leqslant$ и R) имеют общую семантику: тип b получает возможности типа a в виде доступа к его атрибутам. Семантически понятно, что это общее отношение "обладания набором возможностей" (обозначим его $\xleftarrow{R}$) обязано быть рефлексивным и транзитивным. Обсудим другие свойства введенных отношений.

Пусть для элементов $a, b_1, b_2 \in \mathbb{F}$ справедливо $b_1 \leqslant a$, $b_2 \leqslant a$. Тогда по определению решетки имеем $b_1 \vee b_2 \leqslant a$. Это естественное для отношения $\leqslant$ свойство (согласно работе [4]) называется $\vee$-дистрибутивностью. Посмотрим, что будет означать обладание этим же свойством для отношения $\xleftarrow{R}$. Пусть $b_1 \xleftarrow{R} a$ и $b_2 \xleftarrow{R} a$, т. е. каждый тип b_1 и b_2 обладает возможностями типа a. Тогда в силу предполагаемой $\vee$-дистрибутивности имеем $b_1 \vee b_2 \xleftarrow{R} a$. Последнее означает, что тип $b_1 \vee b_2$ также обладает возможностями типа a. С точки зрения проектирования типов это не обязательно. Однако если более одного типа-наследника (в данном случае — b_1 и b_2) содержат одинаковые атрибуты, то согласно принципу рефакторинга [6] целесообразно "поднять" общие атрибуты, т. е. поместить один такой атрибут в общий тип-предок $b_1 \vee b_2$, после чего каждый b_1 и b_2 получит возможности a в порядке наследования. В рассмотренной ситуации $\vee$-дистрибутивность отношения $\xleftarrow{R}$ содержит решение важной задачи — устранение дублирования кода.

Последнее из отмеченных свойств исследовано в работе [5]. В частности, показано, что отношение $\xleftarrow{R}$, обладая свойством транзитивности вне зависимости от контекста, не может во всех ситуациях удовлетворять свойству $\vee$-дистрибутивности, так как это приведет к некорректным результатам. Формально описаны ситуации подобных коллизий и построена LP-структура с ограниченным свойством $\vee$-дистрибутивности. Принятая стратегия предполагает отказ от "поднятия" атрибутов при невыполнении $\vee$-дистрибутивности. Теоретически возможны и другие подходы, более тонко учитывающие особенности конкретных систем.

Как известно из алгебраической логики [12], "решеточные" операции $\vee$ и $\wedge$ порождают в алгебраических системах двойственные свойства (например, законы де Моргана). Подобная закономерность существует и в LP-структурах [4], где наряду с $\vee$-дистрибутивностью отношений присутствует симметричное свойство — $\wedge$-дистрибутивность. Применительно к модели иерархии типов такое свойство впервые рассмотрено в работе [7]. В настоящей статье уточняются и доказываются некоторые утверждения, анонсированные в работах [7—8]. Выясним вначале, что означает свойство $\wedge$-дистрибутивности применительно к указанному выше отношению $\xleftarrow{R}$.

Предположим, что для элементов $a_1, a_2, b \in \mathbb{F}$ выполнено $b \leqslant a_1$, $b \leqslant a_2$. Тогда по определению решетки справедливо $b \leqslant a_1 \wedge a_2$. Такое свойство частичного порядка $\leqslant$ на решетке называется $\wedge$-дистрибутивностью [4]. Распространим его на отношение $\xleftarrow{R}$. Пусть $b \xleftarrow{R} a_1$ и $b \xleftarrow{R} a_2$, т. е. тип b обладает возможностями типов a_1 и a_2. В силу предполагаемой $\wedge$-дистрибутивности получим $b \xleftarrow{R} a_1 \wedge a_2$. Таким образом, тип b также обладает возможностями типа $a_1 \wedge a_2$. Как и было сказано выше, с точки зрения проектирования типов последнее соотношение обязательным не является. Его семантика такова: если тип имеет два или более различных атрибута, то эти атрибуты можно заменить единственным атрибутом, относящимся к ближайшему общему потомку типов исходных атрибутов. Очевидно, что такая реорганизация сделает определение типа-контейнера более компактным с сохранением его функциональности. Этот новый метод рефакторинга (по край-

ней мере, он отсутствует в известном перечне [6]) назван "совмещением атрибутов". Данный метод актуален лишь для иерархий типов с множественным наследованием. Тем не менее его существование можно оправдать популярностью языка C++.

Выясним, насколько в данной алгебраической модели типов свойство $\wedge$-дистрибутивности отношения $\overset{R}{\leftarrow}$ универсально, и не окажутся ли его ограничения двойственными по отношению к описанным в работе [5] ограничениям $\vee$-дистрибутивности.

Пример 1. Пусть $b \overset{R}{\leftarrow} a$ и $b \overset{R}{\leftarrow} b$. При $\wedge$-дистрибутивности отношения $\overset{R}{\leftarrow}$ должно быть выполнено $b \overset{R}{\leftarrow} b \wedge a$. Согласно принципам объектно-ориентированного программирования, тип b не имеет права что-либо знать о своем типе-наследнике $b \wedge a$. По этой причине тип b, являясь предком типа $b \wedge a$, может обладать его возможностями лишь в случае $b \leqslant a$, иначе соотношение $b \overset{R}{\leftarrow} b \wedge a$ окажется некорректным.

Пример 2. Пусть при $b \overset{R}{\leftarrow} a_1$, $b \overset{R}{\leftarrow} a_2$ элементы b и $a_1 \wedge a_2$ имеют объединение $b \vee (a_1 \wedge a_2) = d \neq I$, причем $b < d$ и $a_1 \wedge a_2 < d$. Если в данном случае допустить $(b, a_1 \wedge a_2) \in R$, то окажется, что тип b обладает возможностями другого типа d одновременно по двум линиям, а именно — и как его потомок, и как контейнер типа $a_1 \wedge a_2$, также имеющего возможности d в порядке наследования. Недостатки такого кода заключаются в его избыточности — разрыв связи $a_1 \wedge a_2 < d$ не приведет к потере функциональности системы типов.

Пример 3. Пусть (b, a_1), (b, a_2), $(b, a_3) \in R$, причем элементы a_1, a_2, a_3, $a_1 \wedge a_2$, $a_2 \wedge a_3$ попарно различны и $(a_1 \wedge a_2) \vee (a_2 \wedge a_3) = a_2$. Тогда пары (b, a_1), (b, a_2) "конфликтуют" с парами (b, a_2), (b, a_3). Этот факт означает, что если в обоих случаях совместить атрибуты (применив свойство $\wedge$-дистрибутивности), то тип b получит возможности типа a_2 одновременно посредством двух атрибутов — $a_1 \wedge a_2$ и $a_2 \wedge a_3$, что также ухудшит код.

Анализируя примеры 1—3 в сравнении с примерами из работы [5], приходим к выводу о двойственном характере ограничений $\wedge$-дистрибутивности и $\vee$-дистрибутивности отношения $\overset{R}{\leftarrow}$, которые необходимы для адекватного моделирования иерархий типов. Отмеченный факт служит дополнительным подтверждением естественности разрабатываемых алгебраических моделей, и, в частности, вводимых ограничений дистрибутивности.

На основе представленных выше соображений в следующем разделе определяется понятие логического бинарного отношения на ограниченной решетке. Оно отражает свойство "обладания набором возможностей" в иерархии типов. Логическое замыкание произвольного отношения на решетке дает все такие пары (b, a), что в типе b доступны возможности типа a. Решив задачу построения логического замыкания, можно автоматизировать верификацию системы типов. Результатом логической редукции являются иерархия с минимальным эквивалентным набором связей в рамках рассматриваемой модели и снижение избыточности кода.

3. LP-структуры и их основные свойства

В разд. 2 изложены общие идеи формализованного представления иерархии типов на основе LP-структуры. Ниже дается формальное определение соответствующей алгебраической системы.

3.1. Основные понятия

Вначале сформулируем некоторые понятия и условия, связанные с ограничением свойства $\wedge$-дистрибутивности продукционно-логических отношений. Как отмечалось в разд. 2, это свойство описывает совмещение атрибутов в определении типа. В данном случае трудности формализации обусловлены попыткой ввести статическое формальное условие для описания динамического процесса. Более точно — условие $\wedge$-дистрибутивности отношения R на любых подходящих парах (b, a_1), $(b, a_2) \in R$ должно быть выполнено как до совмещения атрибутов, относящихся к типам a_1, a_2, так и после него. В дальнейшем при ссылках на примеры для краткости будут использоваться фразы "совмещение атрибутов a_1, a_2", или просто "совмещение a_1, a_2", если это не вызовет противоречий. Такое тем более возможно потому, что формальные определения данного раздела связаны не с типами, а с элементами абстрактной решетки.

Пара (b, a) элементов ограниченной решетки $\mathbb{F}$ называется простой, если $a \vee b = I$ — верхняя грань $\mathbb{F}$.

Определение 1. Пусть R — бинарное отношение на ограниченной решетке $\mathbb{F}$. Две пары вида (b, a_1), $(b, a_2) \in R$ называются $\wedge$-совместимыми в R, если существуют такие c_1, $c_2 \in \mathbb{F}$, что $c_1 \wedge c_2 \leqslant a_1 \wedge a_2$, причем пара $(b, c_1 \wedge c_2)$ простая, а пары (b, c_1), (b, c_2) нетранзитивны в $R \cup \leqslant$. При этом набор элементов $T = (b, c_1, c_2)$ будем называть $\wedge$-дистрибутивной тройкой, а $C = (b, a_1, a_2, c_1, c_2)$ — $\wedge$-дистрибутивным кортежем (в R).

Поясним определение 1 на примере. Предположим, что тип b непосредственно содержит атрибуты a_1 и a_2, причем пара $(b, a_1 \wedge a_2)$ является простой. Тогда, в соответствии с обсуждением разд. 2, атрибуты a_1 и a_2 могут быть совмещены, т. е. заменены одним атрибутом $a_1 \wedge a_2$. До этого действия имеем $c_1 = a_1$, $c_2 = a_2$. После совмещения атрибутов условие определения 1 остается выполненным, но с другими промежуточными элементами: $c_1 = c_2 = a_1 \wedge a_2$.

Сформулированное в определении 1 условие нетранзитивности пар (b, c_1), (b, c_2) обеспечивает совмещение атрибутов, относящихся непосредственно к типу b, а не косвенных, связанных с b цепочкой наследований и агрегаций. Конечно, атрибуты неявно совмещаются вместе со своими "возможностями", но именно непосредственные атрибуты должны удовлетворять определению 1.

Следующее понятие формально описывает возможные варианты конфликтов между тройками элементов, претендующими на свойство $\wedge$-дистрибутивности (см. также о конфликте пар в примере 3).

Определение 2. Рассмотрим $\wedge$-дистрибутивную тройку $T = (b, c_1, c_2)$, а также тройку $T' = (b', c_1', c_2')$, тестируемую на аналогичное свойство. Тройка T назы-

вается нейтрализующей для T' (обозначим $T' \prec T$), если выполнено одно из условий:

1. $b' = b$, $c_1 \wedge c_2 \neq c'_1 \wedge c'_2$ и справедливо хотя бы одно из неравенств $c_1 \wedge c_2 < c'_i$, $i = 1,2$;

2. $b' < b$, $c_1 \wedge c_2 \neq c'_1 \wedge c'_2$ и справедливо хотя бы одно из неравенств $c_1 \wedge c_2 \leqslant c'_i$, $i = 1, 2$;

3. $b' < b$, $c_1 \wedge c_2 = c'_1 \wedge c'_2$.

Говоря неформально, данное определение описывает ситуации, когда нейтрализующая тройка T "угрожает" свойству $\wedge$-дистрибутивности тройки T'. Рассмотрим с этой точки зрения условия 1—3 определения 2.

Пусть имеет место условие 1. Тогда после возможного совмещения атрибутов c_1 и c_2 (т. е. их замены атрибутом $c_1 \wedge c_2$), в силу соотношений $b' = b$, $(b, c_1 \wedge c_2) \in R$, $c_1 \wedge \wedge c_2 < c'_i$ пара (b', c'_i) окажется транзитивной в $R \cup \leqslant$, что сделает невозможной $\wedge$-дистрибутивность тройки T'. Условие 1, в частности, содержит конфликт, который проиллюстрирован в примере 3 разд. 2. Для него справедливы оба соотношения $T' \prec T$ и $T \prec T'$.

Если для T, T' выполнено условие 2, то после совмещения атрибутов c_1 и c_2 получим соотношения $b' < b$, $(b, c_1 \wedge c_2) \in R$, $c_1 \wedge c_2 < c'_i$, что вновь означает транзитивность в $R \cup \leqslant$ пары (b', c'_i) и, соответственно, аннулирует свойство $\wedge$-дистрибутивности тройки T'.

В случае выполнения условия 3, после совмещения атрибутов c_1 и c_2 придем к ситуации, когда обе пары (b', c'_i), $i = 1, 2$ окажутся транзитивными в $R \cup \leqslant$: $b' < b$, $(b, c_1 \wedge c_2) \in R$, $c_1 \wedge c_2 = c'_1 \wedge c'_2 < c'_i$.

Тройку $T' = (b', c'_1, c'_2)$ будем называть неконфликтной, если для нее не существует ни одной нейтрализующей $\wedge$-дистрибутивной тройки.

Понятия, введенные в определении 2 (и далее) для троек вида $T = (b, c_1, c_2)$ и $T' = (b', c'_1, c'_2)$, автоматически распространяются на соответствующие им кортежи $C = (b, a_1, a_2, c_1, c_2)$ и $C' = (b', a'_1, a'_2, c'_1, c'_2)$. Две $\wedge$-совместимые пары (b, a_1), (b, a_2) называются неконфликтно $\wedge$-совместимыми, если для них существует неконфликтный $\wedge$-дистрибутивный кортеж (b, a_1, a_2, c_1, c_2).

Отношение R на решетке $\mathbb{F}$ называется ограниченно $\wedge$-дистрибутивным, если для любых неконфликтно $\wedge$-совместимых в R пар (b, a_1), (b, a_2) справедливо $(b, a_1 \wedge a_2) \in R$.

Определение 3. Отношение называется логическим с ограничением пересечений (в данной работе — просто логическим), если оно содержит отношение $\leqslant$, транзитивно и ограниченно $\wedge$-дистрибутивно. Логическим замыканием отношения R называется наименьшее логическое отношение, содержащее R и его множество неконфликтно $\wedge$-совместимых пар.

Два отношения R_1 и R_2, определенные на общей решетке, называются эквивалентными ($R_1 \sim R_2$), если их логические замыкания совпадают. Логической редукцией отношения R называется эквивалентное ему минимальное отношение R_0. Заметим, что при этом не требуется вложения $R_0 \subseteq R$. В принципе можно говорить о некотором отношении R_0 как логической редукции вообще, не относя этот факт к какому-либо другому отношению R. Это означает, что при изъятии любой пары меньшее отношение не будет эквивалентно R_0.

Для выяснения вопроса о существовании логической редукции введем следующее понятие логической связи.

Определение 4. Пусть задано произвольное бинарное отношение R на решетке $\mathbb{F}$. Будем констатировать, что упорядоченная пара b, $a \in \mathbb{F}$ логически связана отношением R ($b \xleftarrow{R} a$), если выполнено одно из следующих условий:

1. $(b, a) \in R$;

2. $b \leqslant a$;

3. существуют такие $a_1, a_2 \in \mathbb{F}$, что $a = a_1 \wedge a_2$, причем $b \xleftarrow{R} a_1$, $b \xleftarrow{R} a_2$ и пары (b, a_1), (b, a_2) неконфликтно $\wedge$-совместимы;

4. существует элемент $c \in \mathbb{F}$ такой, что $b \xleftarrow{R} c$ и $c \xleftarrow{R} a$.

Условия 1—4 определения 4 будем также называть правилами (вывода). При получении некоторой логической связи на основе определения 4 шагом вывода будем называть применение ровно одного правила из этого определения к некоторому конечному множеству элементов решетки. Например, если $b_t \xleftarrow{R} c_t$, $c_t \xleftarrow{R} a_t$, $t \in T$, то $b_t \xleftarrow{R} a_t$, $t \in T$.

Уровнем рекурсии в соотношении $b \xleftarrow{R} a$ будем называть число шагов, необходимое для получения этой связи. При этом учитываются лишь применения правил 3—4. Для связи, основанной только на правилах 1—2, уровень рекурсии считается равным нулю. Поскольку в общем случае связь $b \xleftarrow{R} a$ может быть получена не единственным набором правил, будем лишь оценивать ее уровень рекурсии, не указывая его точного значения.

Применительно к шагам вывода соотношения $b \xleftarrow{R} a$ можно употреблять слова "начальный", "последний", "предыдущий", "следующий" и т.д. При этом имеется в виду продвижение в направлении прямого логического вывода, т. е. от пар исходного отношения ($R \cup \leqslant$) к рассматриваемой паре отношения $\xleftarrow{R}$. Заметим, что рекурсивное определение 4 сформулировано в соответствии с принципом обратного вывода.

3.2. Структура логических связей и редукция

Для дальнейших рассуждений необходим ряд свойств $\wedge$-дистрибутивных троек и $\wedge$-совместимых пар. Следующие лемма и две теоремы приводятся без доказательств, которые планируются к опубликованию в отдельной статье.

Лемма 1. Множество неконфликтно $\wedge$-совместимых пар отношения $\xleftarrow{R}$ совпадает с таковым множеством базового отношения R.

Теорема 1. Для произвольного отношения R на решетке $\mathbb{F}$ логическое замыкание существует и совпадает с множеством $\xleftarrow{R}$ всех упорядоченных пар, логически связанных отношением R.

Далее рассмотрим вопрос об эквивалентных преобразованиях логических структур на решетках типов. Пусть дано отношение R на решетке $\mathbb{F}$. Его эквивалентным преобразованием называется такая замена множества упорядоченных пар R, в результате кото-

рой полученное новое отношение P эквивалентно R. Справедлива следующая теорема.

Теорема 2. Пусть R — отношение на решетке $\mathbb{F}$. Тогда каждая из следующих операций над R приводит к эквивалентному отношению:

- добавление или исключение пары (b, a), если $b \leqslant a$;
- добавление пары $(b, c_1 \wedge c_2)$, если тройка $T = (b, c_1, c_2)$ $\wedge$-дистрибутивна и неконфликтна в R;
- добавление или исключение пары (c, a) при наличии пар $(c, b), (b, a) \in (R \cup \leqslant)$, $c \neq b$, $b \neq a$.

В работе [4] и ряде других работ для стандартных LP-структур показано, что логическое замыкание отношения R совпадает с РТЗ другого отношения $\tilde{R} \supseteq R$, построенного в виде некоторого "дистрибутивного многообразия" над R. Этот факт позволяет свести некоторые вопросы, касающиеся логических отношений, к соответствующим проблемам транзитивных отношений. В частности, построение логического замыкания или редукции можно осуществить с помощью быстрых алгоритмов (типа Уоршолла) [10]. В рамках модели, основанной на ограниченной $\wedge$-дистрибутивности, также удается разделить процесс построения логического замыкания на этапы дистрибутивного и рефлексивно-транзитивного замыканий.

Для отношения R на решетке $\mathbb{F}$ рассмотрим отношение $\tilde{R}$, построенное последовательным выполнением следующих двух шагов:

- для каждой неконфликтной $\wedge$-дистрибутивной тройки (b, c_1, c_2) $(c_1 \neq c_2)$ добавить к исходному отношению пару $(b, c_1 \wedge c_2)$;
- к полученному отношению добавить отношение $\leqslant$.

Заметим, что по теореме 2 отношение $\tilde{R}$ эквивалентно R.

Теорема 3. Логическое замыкание отношения R совпадает с РТЗ $\tilde{R}^*$ соответствующего отношения $\tilde{R}$.

Доказательство. Пусть $b \xleftarrow{R} a$. Покажем, что при этом $(b, a) \in \tilde{R}^*$, что в свою очередь означает вложение $\xleftarrow{R} \subseteq \tilde{R}^*$. Воспользуемся индукцией по m — уровню рекурсии в соотношении $b \xleftarrow{R} a$. При $m = 0$ имеет место одно из условий 1—2 определения 4. Очевидно, что $\tilde{R}^*$ содержит отношения R и $\leqslant$.

Предположим, что вложение верно для некоторого $m \geqslant 0$. Докажем его справедливость при уровне рекурсии $m + 1$. В этом случае новые для рассмотрения варианты могут появиться из правил 3—4 определения 4.

Рассмотрим ситуацию, когда соотношение $b \xleftarrow{R} a$ получено из условия 3. Тогда существуют неконфликтно $\wedge$-совместимые в R пары (b, a_1), (b, a_2) с кортежем (b, a_1, a_2, c_1, c_2), причем $a = a_1 \wedge a_2$. При построении $\tilde{R}$ на первом шаге к исходному отношению должна быть добавлена пара $(b, c_1 \wedge c_2)$, на втором — пара $(c_1 \wedge c_2, a_1 \wedge a_2)$. Наконец, при нахождении транзитивного замыкания $\tilde{R}^*$ появится также пара $(b, a_1 \wedge a_2)$. Таким образом, $(b, a) \in \tilde{R}^*$.

Предположим теперь, что $b \xleftarrow{R} a$ произошло из правила 4. В этом случае по предположению индукции базовые связи $b \xleftarrow{R} c$ и $c \xleftarrow{R} a$ имеют уровень рекурсии $\leqslant m$, т. е. $(b, c), (c, a) \in \tilde{R}^*$. Поскольку $\tilde{R}^*$ — транзитивное замыкание, то оно содержит все свои транзитивные пары, т. е. $(b, a) \in \tilde{R}^*$.

Таким образом, доказано вложение $\xleftarrow{R} \subseteq \tilde{R}^*$. Покажем обратное. Пусть $(b, a) \in \tilde{R}^*$. Тогда в силу свойств транзитивного замыкания существует такой набор элементов $d_0, d_1, ..., d_n$, что $(d_{j-1}, d_j) \in \tilde{R}$, $j = 1, ..., n$, причем $d_0 = b$, $d_n = a$. Если при этом $(d_{j-1}, d_j) \in R$ либо $d_{j-1} \leqslant d_j$, то по условиям 1—2 определения 4 имеем $d_{j-1} \xleftarrow{R} d_j$. Рассмотрим пару (d_{j-1}, d_j), для которой условия 1—2 не выполнены. Тогда возможен лишь случай, когда пара $(d_{j-1}, d_j) \in \tilde{R}$ имеет вид $(b, c_1 \wedge c_2)$, соответствующий некоторой неконфликтной тройке $T = (b, c_1, c_2)$ отношения R (см. построение $\tilde{R}$). Как следствие, по теореме 2 и в этом случае $d_{j-1} \xleftarrow{R} d_j$. Таким образом, установлено, что для пары $(b, a) \in \tilde{R}^*$ существует набор $d_0, d_1, ..., d_n$ $(d_0 = b, d_n = a)$, для которого справедливо $d_{j-1} \xleftarrow{R} d_j, j = 1, ..., n$. Применяя к его элементам последовательно (например, слева направо) n раз правило 4 определения 4, получим $b \xleftarrow{R} a$.

Далее докажем вспомогательные утверждения, которые необходимы для изучения вопросов о существовании и построении логической редукции отношений на ограниченной решетке.

Лемма 2. Пусть R — бинарное отношение на решетке $\mathbb{F}$ и $b_t \xleftarrow{R} a_t$ $\forall t \in T$. Тогда отношение $R' = R \cup \{(b_t, a_t) | t \in T\}$ эквивалентно R.

Доказательство. Заметим вначале, что по определению любое логическое отношение является собственным логическим замыканием. В силу теоремы 1 это относится и к отношению $\xleftarrow{R}$. Из определения 4 очевидным образом следует, что для любых $R_1 \subseteq R_2$ справедливо $\xleftarrow{R_1} \subseteq \xleftarrow{R_2}$. В рассматриваемом случае по построению отношения R' выполнены включения $R \subseteq R' \subseteq \xleftarrow{R}$. Переходя к логическим замыканиям и учитывая изложенное выше, получаем, что отношения R и R' имеют общее логическое замыкание $\xleftarrow{R}$.

Лемма 3. Пусть R — бинарное отношение. Для того чтобы R являлось логической редукцией, необходимо и достаточно, чтобы R не содержало ни одной такой пары (b, a), для которой выполнено соотношение $b \xleftarrow{R\setminus\{(b, a)\}} a$.

Доказательство. Пусть отношение R представляет собой логическую редукцию, т. е. является минимальным логически эквивалентным себе отношением. Предположим противное, а именно, что существует пара $(b, a) \in R$, логически связанная отношением $R\setminus\{(b, a)\}$. Если это так, то в силу леммы 2 пару (b, a) можно исключить из R, получив при этом меньшее эквивалентное отношение. Таким образом, при сделанном предположении отношение R не может быть логической редукцией.

Для доказательства обратного утверждения предположим, что не существует ни одной пары $(b, a) \in R$, для которой справедливо $b \xleftarrow{R\setminus\{(b, a)\}} a$. Необходимо

доказать, что в этом случае R есть логическая редукция. Вновь предположим противное — пусть существует отношение $R_0 \subset R$, эквивалентное R, и $(b, a) \in \in R \backslash R_0$. Тогда, поскольку $(b, a) \in R$, в силу эквивалентности рассматриваемых отношений справедливо $b \xleftarrow{R_0} a$. Так как отношение R_0 не содержит пару (b, a), то $R_0 \subseteq \subseteq R\backslash\{(b, a)\}$, и логическая связь $b \xleftarrow{R_0} a$ противоречит сделанному предположению — таких пар (b, a) в R нет. Полученное противоречие доказывает требуемое утверждение.

Выясним непосредственно вопрос о существовании и построении логической редукции. Справедлива следующая теорема.

Теорема 4. Пусть для отношения R построено соответствующее отношение $\tilde{R}$. Тогда если для $\tilde{R}$ существует транзитивная редукция R^0, то отношение $\underset{\sim}{R}^0$, полученное исключением из R^0 всех пар вида $b \leqslant a$, представляет собой логическую редукцию исходного отношения R.

Доказательство. Из леммы 2 следует, что указанное в теореме отношение $\underset{\sim}{R}^0$ логически эквивалентно R. Осталось показать, что $\underset{\sim}{R}^0$ является логической редукцией вообще. Для этого достаточно проверить выполнение для $\underset{\sim}{R}^0$условия леммы 3.

Пусть (b, a) — произвольная пара из $\underset{\sim}{R}^0$. Необходимо показать, что связь $b \xleftarrow{\underset{\sim}{R}^0\backslash\{(b, a)\}} a$ невозможна. Предположим противное, что эта связь существует. Тогда в силу леммы 2 отношение $\underset{\sim}{R}^0\backslash\{(b, a)\}$ эквивалентно $\underset{\sim}{R}^0$. Сразу заметим, что применение правила 1 определения 4 для вывода $b \xleftarrow{\underset{\sim}{R}^0\backslash\{(b, a)\}} a$ невозможно, поскольку пара (b, a) не содержится в множестве $\underset{\sim}{R}^0\backslash\{(b, a)\}$.

По лемме 1 множество неконфликтно $\wedge$-совместимых пар произвольного бинарного отношения инвариантно относительно применения правил вывода 2—4 определения 4. Следовательно, любая логическая связь может быть получена таким образом, что все необходимые для этого применения $\wedge$-дистрибутивного правила 3 будут проведены на начальных шагах вывода, а все применения транзитивного правила 4 — лишь в завершающей стадии этого вывода. Таким образом, для связи $b \xleftarrow{\underset{\sim}{R}^0\backslash\{(b, a)\}} a$ существует цепочка элементов $b = c_0, c_1, ..., c_n = a$ такая, что выполнены соотношения $c_i \xleftarrow{\underset{\sim}{R}^0\backslash\{(b, a)\}} c_{i-1}$, $i = 1, ..., n$, при выводе каждого из которых правило 4 не используется. Отсюда следует, что $(c_i, c_{i-1}) \in \tilde{R}$. Таким образом, получаем, что при $n > 1$ пара (b, a) оказывается транзитивной в $\tilde{R}$. Следовательно, она не может содержаться в $\underset{\sim}{R}^0$, являющемся подмножеством транзитивной редукции отношения $\tilde{R}$. Это противоречит исходному предположению $(b, a) \in \underset{\sim}{R}^0$.

Остается рассмотреть случай $n = 1$. В этой ситуации существующая логическая связь $b \xleftarrow{\underset{\sim}{R}^0\backslash\{(b, a)\}} a$ означает, что $(b, a) \in \tilde{R}$. В силу вышеуказанной инвариантности множества неконфликтно $\wedge$-совместимых пар для (b, a) остается лишь одна из двух возможностей: применено правило 2 или 3. Если это правило 2, то при получении $\underset{\sim}{R}^0$ такая пара (b, a) должна быть исключена.

Покажем, наконец, что в случае применения правила 3 пара $(b, a) \in \tilde{R}$ будет исключена на этапе вычисления транзитивной редукции $\tilde{R}$. Пусть существовали такие $a_1, a_2 \in \mathbb{F}$, что $a_1 \wedge a_2 = a$, причем $b \xleftarrow{R} a_1$, $b \xleftarrow{R} a_2$ и пары (b, a_1), (b, a_2) неконфликтно $\wedge$-совместимы. Тогда при построении $\tilde{R}$ к нему была добавлена соответствующая пара $(b, c_1 \wedge c_2)$. Поскольку (см. определение 1) $c_1 \wedge c_2 \leqslant a_1 \wedge a_2$, то пара (b, a) оказывается транзитивной в $\tilde{R}$. В результате снова приходим к противоречию исходному предположению $(b, a) \in \underset{\sim}{R}^0$.

Таким образом, рассмотрены возможные варианты предполагаемого логического вывода $b \xleftarrow{\underset{\sim}{R}^0\backslash\{(b, a)\}} a$. В результате установлено, что в каждом случае наличие связи $b \xleftarrow{\underset{\sim}{R}^0\backslash\{(b, a)\}} a$ противоречит факту $(b, a) \in \underset{\sim}{R}^0$. Следовательно, по лемме 3 отношение $\underset{\sim}{R}^0$ представляет собой логическую редукцию.

4. Заключение

Представленный формализм позволяет проводить автоматизированные исследования иерархий типов, включая эквивалентные преобразования, верификацию и оптимизацию. Он может служить основой для практической реализации (или модернизации) типов.

Список литературы

1. **Подловченко Р. И.** Иерархия моделей программ // Программирование. 1981. № 2. С. 3—14.
2. **Замулин А. В.** Алгебраическая семантика императивного языка программирования // Программирование. 2003. № 6. С. 51—64.
3. **Davis R., King J.** An overview of production systems // Machine Intell. 1977. Vol 8. P. 300—332.
4. **Махортов С. Д., Подвальный С. Л.** Алгебраический подход к исследованию и оптимизации баз знаний продукционного типа // Информационные технологии. 2008. № 8. С. 55—60.
5. **Махортов С. Д.** LP-структуры на решетках типов и некоторые задачи рефакторинга // Программирование. 2009. Т. 35. № 4. С. 5—14.
6. **Фаулер М.** Рефакторинг: улучшение существующего кода: пер. с англ. СПб.: Символ-Плюс, 2004. 432 с.
7. **Махортов С. Д.** LP-структуры для обоснования и автоматизации рефакторинга в объектно-ориентированном программировании // Программная инженерия. 2010. № 2. С. 15—21.
8. **Махортов С. Д., Шурлин М. Д.** Алгебраические модели иерархий типов для проектирования и рефакторинга // Онтология проектирования. 2012. № 1(3). С. 73—79.
9. **Godin R., Valtchev P.** Formal Concept Analysis-Based Class Hierarchy Design in Object-Oriented Software Development // Formal Concept Analysis / eds. B. Ganter, G. Stumme, R. Wille. Lecture Notes In Computer Science. Springer Berlin/Heidelberg. 2005. Vol. 3626. P. 304—323.
10. **Aho A. V., Garey M. R., Ulman J. D.** The transitive reduction of a directed graph. SIAM J. Computing 1:2. 1972. P. 131—137.
11. **Биркгоф Г.** Теория решеток: пер. с англ. М.: Наука, 1984. 568 с.
12. **Расёва Е., Сикорский Р.** Математика метаматематики: пер. с англ. М.: Наука, 1972. 591 с.

CONTENTS

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4

Дизайнер *Т. Н. Погорелова*. Технический редактор *Е. М. Патрушева*. Корректор *Е. В. Комиссарова*

Сдано в набор 02.11.2012 г. Подписано в печать 17.12.2012 г. Формат 60×88 1/8. Заказ PI113
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
105120, г. Москва, ул. Нижняя Сыромятническая, д. 5/7, стр. 2, офис 2.