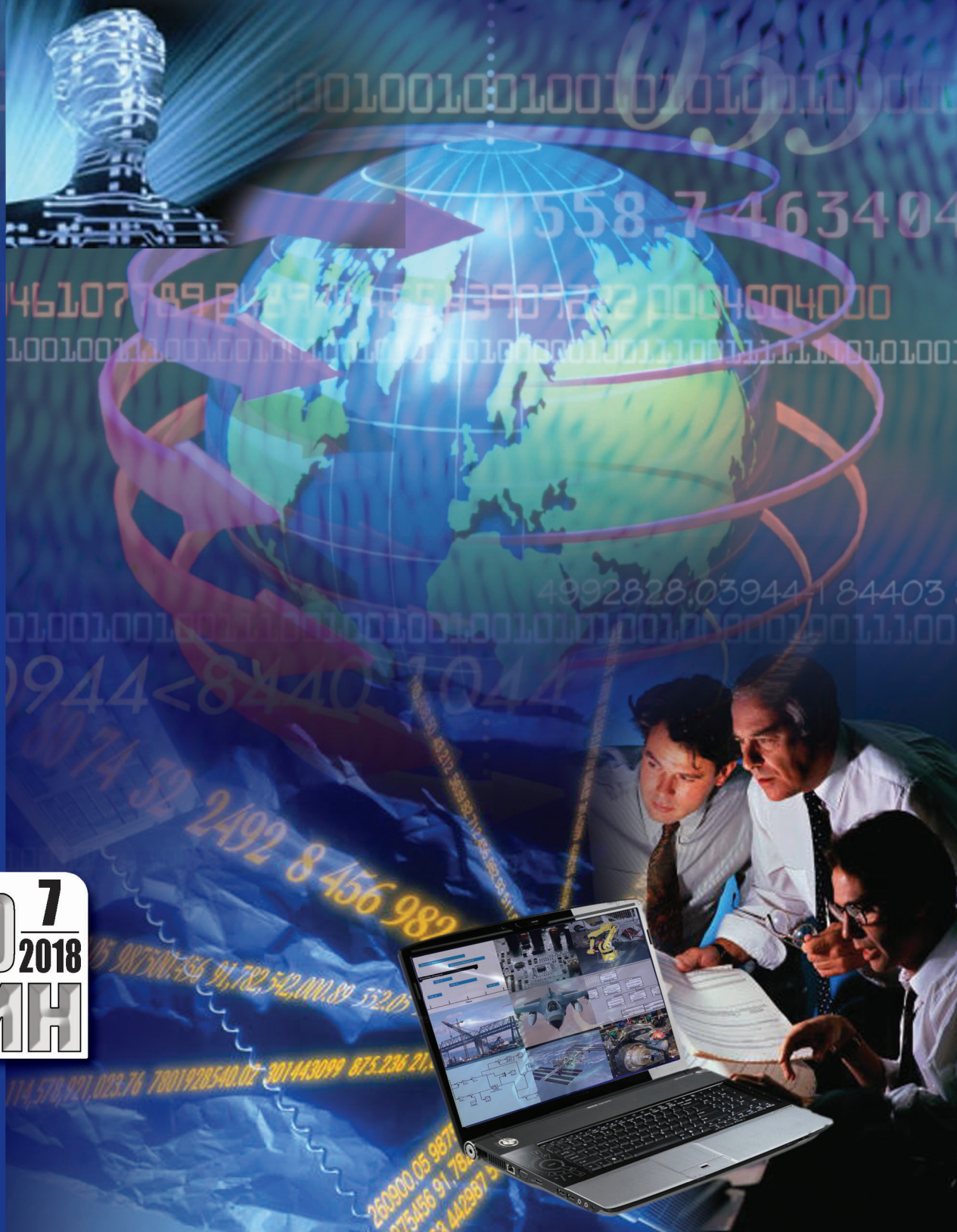
ISSN 2220-3397

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия



ПрИн
7
2018
Том 9

Рисунки к статье Д. И. Читалова, С. Т. Калашникова

«РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОДГОТОВКИ РАСЧЕТНЫХ СЕТОК С ПОМОЩЬЮ УТИЛИТЫ foamyQuadMesh ПЛАТФОРМЫ OpenFOAM»

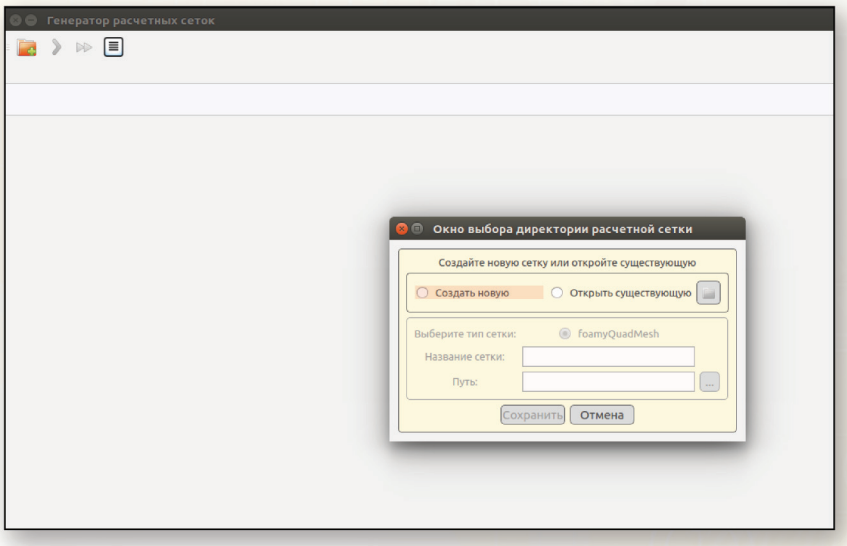


Рис. 3. Основное окно приложения foamyQuadMesh_generator с открытым диалоговым окном выбора режима работы с программой

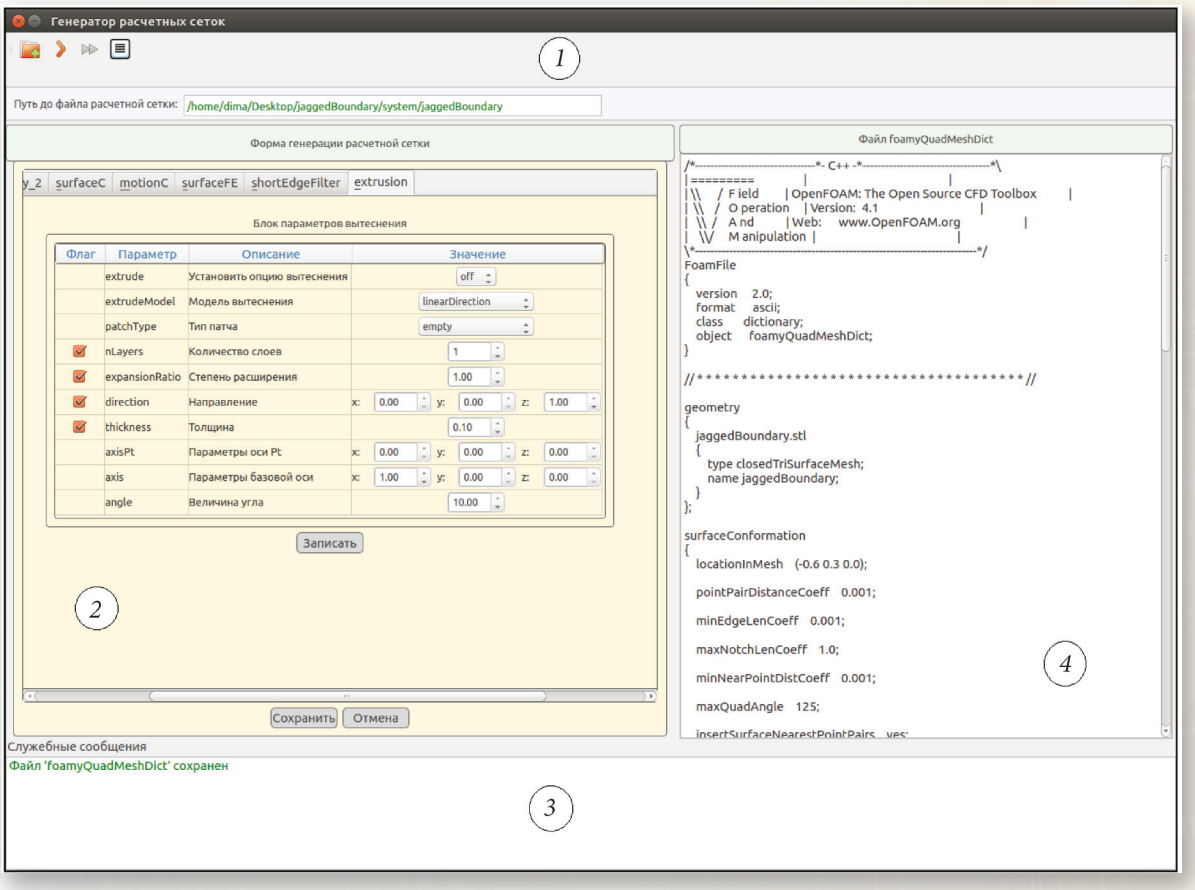


Рис. 4. Основное окно приложения foamyQuadMesh_generator после сохранения параметров РС:
1 – панель инструментов; *2* – окно указания исходных параметров сетки;
3 – окно вывода служебной информации; *4* – окно вывода результатов

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Редакционный совет
Садовничий В.А., акад. РАН (*председатель*)
Бетелин В.Б., акад. РАН
Васильев В.Н., чл.-корр. РАН
Жижченко А.Б., акад. РАН
Макаров В.Л., акад. РАН
Панченко В.Я., акад. РАН
Стемпковский А.Л., акад. РАН
Ухлинов Л.М., д.т.н.
Федоров И.Б., акад. РАН
Четверушкин Б.Н., акад. РАН

Главный редактор
Васенин В.А., д.ф.-м.н., проф.

Редколлегия
Антонов Б.И.
Афонин С.А., к.ф.-м.н.
Бурдонов И.Б., д.ф.-м.н., проф.
Борзовс Ю., проф. (Латвия)
Гаврилов А.В., к.т.н.
Галатенко А.В., к.ф.-м.н.
Корнеев В.В., д.т.н., проф.
Костюхин К.А., к.ф.-м.н.
Махортов С.Д., д.ф.-м.н., доц.
Манцивода А.В., д.ф.-м.н., доц.
Назиров Р.Р., д.т.н., проф.
Нечаев В.В., д.т.н., проф.
Новиков Б.А., д.ф.-м.н., проф.
Павлов В.Л. (США)
Пальчунов Д.Е., д.ф.-м.н., доц.
Петренко А.К., д.ф.-м.н., проф.
Позднеев Б.М., д.т.н., проф.
Позин Б.А., д.т.н., проф.
Серебряков В.А., д.ф.-м.н., проф.
Сорокин А.В., к.т.н., доц.
Терехов А.Н., д.ф.-м.н., проф.
Филимонов Н.Б., д.т.н., проф.
Шапченко К.А., к.ф.-м.н.
Шундеев А.С., к.ф.-м.н.
Щур Л.Н., д.ф.-м.н., проф.
Язов Ю.К., д.т.н., проф.
Якобсон И., проф. (Швейцария)

Редакция
Лысенко А.В., Чугунова А.В.

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

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/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования и базу данных RSCI на платформе Web of Science.
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.

© Издательство "Новые технологии", "Программная инженерия", 2018

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 9
N 7
2018

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

Editorial Council:

SADOVNICHY V. A., Dr. Sci. (Phys.-Math.), Acad. RAS (*Head*)
BETELIN V. B., Dr. Sci. (Phys.-Math.), Acad. RAS
VASIL'EV V. N., Dr. Sci. (Tech.), Cor.-Mem. RAS
ZHIZHCHENKO A. B., Dr. Sci. (Phys.-Math.), Acad. RAS
MAKAROV V. L., Dr. Sci. (Phys.-Math.), Acad. RAS
PANCHENKO V. YA., Dr. Sci. (Phys.-Math.), Acad. RAS
STEMPKOVSKY A. L., Dr. Sci. (Tech.), Acad. RAS
UKHLINOV L. M., Dr. Sci. (Tech.)
FEDOROV I. B., Dr. Sci. (Tech.), Acad. RAS
CHETVERTUSHKIN B. N., Dr. Sci. (Phys.-Math.), Acad. RAS

Editor-in-Chief:

VASENIN V. A., Dr. Sci. (Phys.-Math.)

Editorial Board:

ANTONOV B.I.
AFONIN S.A., Cand. Sci. (Phys.-Math)
BURDONOV I.B., Dr. Sci. (Phys.-Math)
BORZOVS JURIS, Dr. Sci. (Comp. Sci), Latvia
GALATENKO A.V., Cand. Sci. (Phys.-Math)
GAVRILOV A.V., Cand. Sci. (Tech)
JACOBSON IVAR, Dr. Sci. (Philos., Comp. Sci.), Switzerland
KORNEEV V.V., Dr. Sci. (Tech)
KOSTYUKHIN K.A., Cand. Sci. (Phys.-Math)
MAKHORTOV S.D., Dr. Sci. (Phys.-Math)
MANCIVODA A.V., Dr. Sci. (Phys.-Math)
NAZIROV R.R. , Dr. Sci. (Tech)
NECHAEV V.V., Cand. Sci. (Tech)
NOVIKOV B.A., Dr. Sci. (Phys.-Math)
PAVLOV V.L., USA
PAL'CHUNOV D.E., Dr. Sci. (Phys.-Math)
PETRENKO A.K., Dr. Sci. (Phys.-Math)
POZDNEEV B.M., Dr. Sci. (Tech)
POZIN B.A., Dr. Sci. (Tech)
SEREBRJAKOV V.A., Dr. Sci. (Phys.-Math)
SOROKIN A.V., Cand. Sci. (Tech)
TEREKHOV A.N., Dr. Sci. (Phys.-Math)
FILIMONOV N.B., Dr. Sci. (Tech)
SHAPCHENKO K.A., Cand. Sci. (Phys.-Math)
SHUNDEEV A.S., Cand. Sci. (Phys.-Math)
SHCHUR L.N., Dr. Sci. (Phys.-Math)
YAZOV Yu. K., Dr. Sci. (Tech)

Editors: LYSENKO A.V., CHUGUNOVA A.V.

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 004.3 DOI: 10.17587/prin.9.291-304

Н. А. Авдеев, канд. техн. наук., ст. науч. сотр., e-mail: avdeev_n@newman.bas-net.by,
П. Н. Бибило, д-р техн. наук, проф., зав. лаб., e-mail: bibilo@newman.bas-net.by,
Объединенный институт проблем информатики Национальной академии наук Беларуси, г. Минск,
В. В. Коробкин, канд. техн. наук, IT-директор, e-mail: vvk@niimvs.ru,
Научно-исследовательский институт многопроцессорных вычислительных и управляющих систем Южного федерального университета, г. Таганрог,
А. Е. Колоденкова, д-р техн. наук, e-mail: anna82_42@mail.ru, Самарский государственный технический университет

Верификация VHDL-описаний сетей синхронных конечных автоматов

Предложена методика верификации VHDL-описания сети синхронных конечных автоматов. Под верификацией понимается проверка соответствия VHDL-описания сети автоматов спецификациям на ее проектирование. Методика использует возможности системы Questa Sim, которая по результатам моделирования VHDL-описания сети автоматов позволяет выделить ориентированные графы переходов компонентных автоматов и подсчитать в графах число прохождений дуг. Однако система Questa Sim не распознает сети конечных автоматов и не имеет средств, обеспечивающих построение тестов по результатам моделирования. Поэтому для решения данных задач предлагается сохранять результаты моделирования — последовательности входных наборов (стимулов) и кортежей состояний компонентных автоматов, а по полученным последовательностям проверять выполнение переходов в графе состояний сети автоматов и тем самым проводить верификацию.

Ключевые слова: *сеть конечных автоматов, функциональная верификация, моделирование, VHDL, функциональные тесты*

Введение

Автоматный стиль нашел широкое применение в программировании, так как позволяет успешно решать задачи верификации компьютерных программ, в том числе с использованием методов формальной верификации [1—3]. При разработке проектов цифровых систем, которые затем реализуются аппаратно в виде логических схем, модели конечных автоматов также имеют широкое применение, особенно для описания блоков управляющей логики, микропроцессоров, интерфейсных схем и др. Для описания проектов цифровых систем в настоящее время два языка занимают лидирующее положение — Verilog [4] и VHDL (*Very high speed integrated circuits Hardware Description Language* — язык описания аппаратуры сверхскоростных интегральных схем) [5]. Далее будут рассматриваться примеры описаний конечных автоматов и сетей автоматов на языке VHDL, однако предлагаемая в настоящей работе практическая методика верификации сетей конечных автоматов применима и для проектов, представленных на языке Verilog.

По VHDL-описаниям сетей конечных автоматов синтезируются синхронные логические схемы в том или ином базисе логических элементов, называемом технологическим (целевым) базисом либо целевой библиотекой логических элементов. В настоящее время процесс синтеза автоматизирован и важнейшей проблемой при создании проектов СБИС и систем-на-кристалле является проблема верификации [6] исходных VHDL-моделей, используемых для алгоритмического описания проектируемых цифровых устройств и систем. В отличие от формальной верификации, когда на эквивалентность поведения проверяют два VHDL-описания цифровой системы [7], в нашей работе под верификацией будем понимать проверку правильности исходного VHDL-описания, т. е. проверку соответствия составленного синтезируемого VHDL-описания проектируемой цифровой системы спецификациям на проектирование [8].

Большим достоинством модели конечного автомата (FSM — *Finite State Machine*) является то, что данная модель может быть верифицирована. Система *Questa Sim* [6, 7] моделирования HDL-описаний цифровых устройств имеет в своем составе средства для функциональной верификации FSM. Такие средства (опции) позволяют при моделировании распозна-

вать конечный автомат, входящий в состав проекта цифрового устройства, определять все пройденные состояния конечного автомата и подсчитывать число прохождений дуг в графе переходов автомата. Эти средства являются весьма полезными, однако конечные автоматы, как правило, часто образуют сети и входят в состав более сложных проектов. Для проведения верификации всего проекта в целом [8] требуется построение компактных функциональных тестов для сетей конечных автоматов. Однако система *Questa Sim* не распознает сети автоматов и не имеет средств, обеспечивающих построение функциональных тестов по результатам моделирования.

Описание предложенных авторами методики и средств автоматизированного построения таких тестов по результатам моделирования VHDL-описаний сетей конечных автоматов и являются целью настоящей работы. Для проведения моделирования требуется написание тестирующих VHDL-программ, использующих средства генерации псевдослучайных тестовых наборов, а также средства функционального покрытия.

1. Постановка задачи

Исходные спецификации на проектирование отдельных (компонентных) конечных автоматов и сетей взаимодействующих автоматов чаще всего задаются в виде таблиц либо ориентированных графов переходов между состояниями компонентных автоматов, вершинам которых соответствуют состояния, а ориентированным дугам — переходы между состояниями. Далее под состоянием компонентного автомата будем всегда понимать его внутреннее состояние, а под состоянием сети синхронных автоматов — кортеж внутренних состояний компонентных автоматов. По исходным неформальным спецификациям затем составляются VHDL-описания, которые являются формальными и моделируемыми и для которых требуется провести верификацию. Основным подходом для проведения такой верификации является моделирование, которое требует

— написания тестирующих программ;

— подготовки соответствующих тестов, либо VHDL-программ, генерирующих тесты;

— выполнения моделирования и сравнения полученных реакций VHDL-модели с ожидаемыми реакциями.

Написание тестирующих программ и проведение моделирования в системе *Questa Sim* подробно описано в работе [6]. Для моделирования VHDL-описаний была использована система *Questa Sim*, далее будут обсуждены некоторые ее возможности. Верификация VHDL-описания отдельного компонентного автомата рассмотрена в работе [9]. Она заключается в проверке достижимости всех состояний автомата и выполнении всех требуемых переходов между состояниями автомата. Для сети автоматов задача существенно усложняется, так как требуется проверить, чтобы компонентные автоматы синхронно попадали

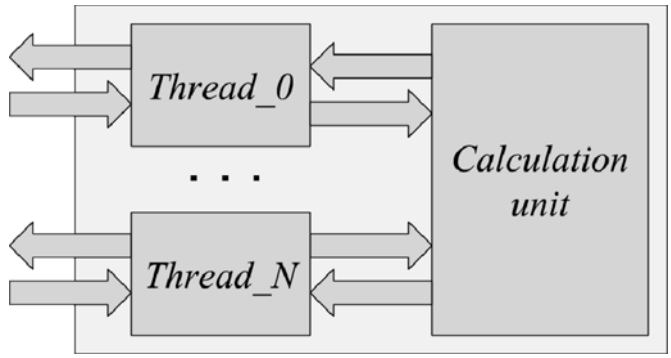


Рис. 1. Сеть автоматов *fsm_multi*

в требуемые состояния. Важно также верифицировать сеть автоматов на то, чтобы компонентные автоматы одновременно (синхронно) не были в запрещенных состояниях. Приведем несколько примеров.

Пусть имеется *fsm_multi* — синхронное цифровое устройство (рис. 1), состоящее из N автоматов (запросов) *Thread_0*, ..., *Thread_N*, которые связаны, т. е. требуют получить доступ к одному и тому же автомату (ресурсу) *Calculation unit*.

Такая модель возникает при доступе к арифметико-логическому устройству (АЛУ), в роли которого выступает *Calculation unit*, от нескольких устройств — автоматов-запросов *Thread_J* ($J = 0, ..., N$), которые не связаны друг с другом. Каждый из автоматов-запросов *Thread_J* связан только с *Calculation unit*. Каждый автомат-запрос имеет входные данные — операнды арифметического блока и выходные данные — результаты выполнения операций в *Calculation unit*. В этом примере вычислительный блок в течение нескольких тактов обрабатывает операнды, поступающие только от одного автомата-запроса. При этом должно быть обеспечено требование — во время работы вычислительного блока запрещается подача в него следующих операндов (от любого из автоматов-запросов). Примеры взаимодействующих автоматных моделей представлены, например, в статьях специального выпуска "Автоматное программирование" журнала [1]. При управлении технологическим оборудованием и робототехническими комплексами [10], при проектировании встраиваемых систем [11] также используют взаимодействующие автоматные модели.

Введем обозначения и сформулируем постановку задачи для *параллельной* сети конечных автоматов, которая легко обобщается для произвольной синхронной сети взаимодействующих конечных автоматов.

Пусть A^0, A^1, ..., A^{p-1} — компонентные конечные автоматы с входными сигналами — подмножествами из общего множества X входных сигналов, общим сигналом *rst* сброса и общим синхросигналом *clk*. Через $Q^0 = \{q_0^0, ..., q_{k_0}^0\}$, $Q^1 = \{q_0^1, ..., q_{k_1}^1\}$, ..., $Q^{p-1} = \{q_0^{p-1}, ..., q_{k_{p-1}}^{p-1}\}$ обозначим внутренние состояния компонентных автоматов; через G^0, G^1, ..., G^{p-1} — графы переходов между состояниями компонент-

ных автоматов; p — число компонентных автоматов. Математические модели конечных автоматов широко известны в литературе [12], VHDL-описания синхронных конечных автоматов можно найти в работах [4—6]. Компонентные автоматы образуют *параллельную сеть H* (рис. 2), т. е. переходы между состояниями всех компонентных автоматов осуществляются синхронно, например, по переднему фронту синхросигнала *clk*.

Назовем *обобщенным графом* переходов граф G^H, вершинами которого являются упорядоченные векторы (кортежи). В каждом кортеже i-я компонента — это элемент из множества Q^i, $i = 0, 1, ..., p - 1$. Легко видеть, что множество Q вершин графа G^H образует декартово произведение множеств Q^i: $Q = Q^0 \times Q^1 \times \times ... \times Q^{p-1}$. Обозначим через Z подмножество вершин графа G^H, которое является *запрещенным*. Сеть H автоматов никогда не должна попадать в состояние, входящее в Z. Подмножество $R = Q \backslash Z$ состояний сети H назовем множество *разрешенных* состояний сети. Заметим, что среди разрешенных состояний сети могут быть *недостижимые* состояния, т. е. такие, в которые автомат никогда не попадает. Чаще всего проектировщик в исходных спецификациях не перечисляет их в явном виде.

Переход (дугу графа G^H) назовем запрещенным, если он ведет в одно из запрещенных состояний. Модель компонентного автомата и его VHDL-описание назовем *корректными*, если из любого внутреннего состояния автомата имеется путь в начальное состояние автомата. Совокупность начальных состояний компонентных автоматов будет образовывать начальное состояние сети H.

Задача 1. Задано VHDL-описание параллельной сети H конечных автоматов и задано подмножество Z запрещенных состояний. Требуется провести верификацию VHDL-описания, т. е. проверить, будут ли выполняться в графе G^H переходы в разрешенные состояния R и не выполняться переходы в запрещенные состояния Z.

Проблеме верификации исходных высокоуровневых VHDL-описаний (либо Verilog-описаний) спецификациям на проектирование посвящено большое число работ. Речь идет о соответствии HDL-модели цифрового устройства спецификациям на проектирование. Для такой верификации могут быть применены два основных подхода. Первый подход [13, 14] является формальным и заключается в построении по исходному HDL-описанию соответствующей модели (*High-Level Decision Diagram*, HLLDD), которая сравнивается со спецификациями, в роли которых выступают условия срабатывания переходов расширенных конечных автоматов (*Extended Finite State Machine*, EFSM), также построенных по исходному HDL-описанию. Генерация функционального теста сводится к построению контрпримера в виде последовательности входных тестовых наборов, использование которых при моделировании приводит к поведению, противоречащему спецификации. Это конкретизация подхода, известного в литературе как

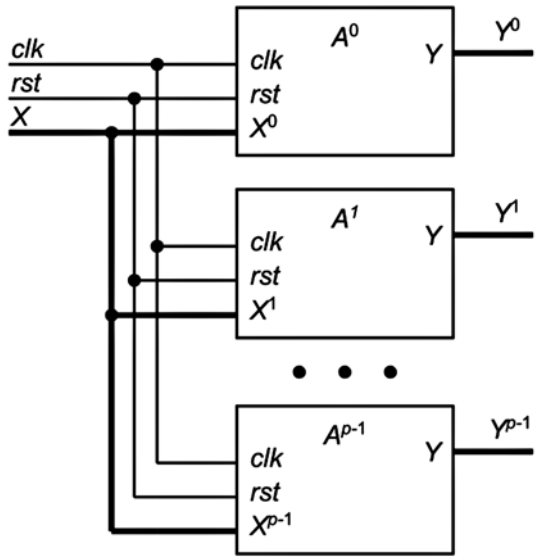


Рис. 2. Общий вид параллельной сети конечных автоматов:

rst — сигнал сбора (установки в начальное состояние); *clk* — синхросигнал; X^0, ..., X^p — входы компонентных автоматов; Y^0, Y^p — выходы компонентных автоматов

проверка моделей (*model checking*) [15]. Данный подход ограничен стилями HDL-описаний, из которых извлекаются соответствующие модели.

Другой подход реализуется на основе моделирования и комплексного тестирования, целью которых являются соответствующие проверки для подтверждения того, что функциональность HDL-описания будет правильной [8]. Этот подход не ограничен стилями исходных описаний, основные проблемы заключаются в генерации направленных функциональных тестов и организации тестирования. В рамках данного подхода, являющегося наиболее распространенным подходом к верификации и названного в работе [16] имитационным тестированием, предлагается решать поставленную задачу верификации параллельной сети конечных автоматов. Обзор существующих моделей параллельно функционирующих распределенных систем, методов и инструментальных средств их верификации представлен в работе [17], где указывается, что одной из основных проблем, возникающих при верификации взаимодействующих конечных автоматов, является комбинаторный взрыв числа возможных параллельных состояний компонентных автоматов. Это касается как формальной верификации, так и верификации на основе моделирования [16, 17].

Так как предполагается, что решение поставленной задачи верификации будет осуществляться на основе моделирования в системе *Questa Sim*, то речь не идет о строгом решении задачи, т. е. о строгой (формальной) верификации VHDL-модели. Если выяснится, что имеются переходы в запрещенные состояния, то это будет свидетельствовать о неправильности VHDL-описания сети автоматов, если же в результате всех видов моделирования (множество

тестов всегда ограничено) выяснится, что все разрешенные переходы выполнены (покрыты) при моделировании и нет ни одного перехода в запрещенное состояние, то будет ***условно*** считаться, что VHDL-описание сети H является правильным.

2. Верификация сети автоматов на основе моделирования

При верификации сети H автоматов возникают задачи верификации каждого из автоматов A^0, A^1, ... A^{p-1}, составляющих сеть, и сети H автоматов в целом. Большим достоинством системы моделирования *Questa Sim* является то, что граф переходов компонентного конечного автомата может быть визуализирован, если модель *FSM* конечного автомата написана по определенному шаблону (стилю). Модель *FSM* должна иметь конечное число внутренних состояний, должны быть переменные текущего и следующего состояний, смена состояний должна проходить по синхросигналу, следующее состояние должно зависеть от текущего состояния. Средства покрытия VHDL-кода позволяют при компиляции и моделировании распознать в составе модели цифровой системы конечный автомат *FSM*, входящий в состав проекта, отследить (учесть) все пройденные (в конкретном сеансе моделирования) состояния конечного автомата и подсчитать число прохождений ориентированных дуг в графе переходов автомата *FSM* и визуализировать граф. Данная методика подробно описана в работе [7]. Однако конечный автомат может быть записан и в другой форме (стиле), которую также надо верифицировать. Выделение из VHDL-описания автомата A^i внутренних состояний автомата и построение графа G^i переходов является нетривиальной задачей, так как сводится к анализу VHDL-кода, синтаксис которого является сложным, а автомат может быть задан в другой форме, отличающейся от требуемой формы описания, и тогда система моделирования *Questa Sim* не сможет его распознать. По сути надо автоматизировать процесс построения математической модели графа переходов по VHDL-программе, задающей автомат, например, построить матрицу смежности ориентированного графа переходов. Задача заметно усложняется для сети автоматов, так как требуется построить граф G^H по заданному VHDL-описанию сети H.

Чтобы избежать анализа VHDL-кода, для решения задачи предлагается подход, основанный на моделировании VHDL-описания сети автоматов. Для того чтобы реализовать данный подход, надо правильным образом организовать моделирование, а именно для каждого тестового набора (каждого такта моделирования) выдать внутреннее состояние автомата, в которое переходит автомат при подаче тестового набора на вход VHDL-модели автомата и в котором автомат будет находиться в следующем такте моделирования. Для компонентного автомата такой подход описан в работе [9]. Предложенный в этой работе подход легко обобщается на случай параллельной сети автоматов — при моделировании надо в каждом такте выдавать состояние каждого компонентного автомата, тогда множество вершин графа G^H будет образовано всеми различными кортежами внутренних состояний компонентных автоматов. Покрытие всех переходов в таком графе сведется к поиску соседних пар в последовательности состояний сети H. Этот подход прост в реализации, однако очевидны и его недостатки, связанные с подачей на вход модели псевдослучайных входных воздействий и отсутствием гарантии покрытия каждого перехода в графе G^H. Псевдослучайные тесты должны быть длинными и при этом все равно трудно обеспечить покрытие всех переходов, особенно это касается покрытия всех переходов в начальное состояние, вызываемых сбросом. Поэтому здесь важную роль играет правильное написание тестирующих программ, которые обеспечивают генерацию входных воздействий и проверяют достижение определенных целей верификации. Для написания тестирующих VHDL-программ предлагается использовать средства VHDL-пакетов, реализующих методологию, называемую OS-VVM (*Open Source VHDL Verification Methodology*) [6].

Предлагаемая *методика* верификации VHDL-описаний сетей конечных автоматов с помощью системы *Questa Sim* включает следующие этапы.

Этап 1. Неформальная проверка требуемого стиля описания компонентных автоматов, а именно стиля, который позволит системе *Questa Sim* выделить каждый компонентный автомат. Формальная проверка правильности использованного стиля будет осуществлена на этапе 3.

Этап 2. Моделирование VHDL-описания сети автоматов с помощью специально написанных тестирующих программ, позволяющих генерировать псевдослучайные входные воздействия, подавать их на вход VHDL-модели и получать состояния сети автоматов на каждом такте. Таким образом, результатами потактового моделирования являются входные воздействия для сети автоматов и соответствующие им кортежи состояний компонентных автоматов.

Этап 3. Визуализация графов переходов компонентных автоматов и проверка выполнения всех требуемых переходов компонентных автоматов согласно исходным спецификациям. Если модель конечного автомата не извлекается из VHDL-описания, т. е. если граф компонентного автомата не визуализируется, то надо вернуться на этап 1 и привести VHDL-описание компонентного автомата в требуемую форму.

Этап 4. Построение графа G^H по результатам моделирования.

Этап 5. Анализ графа G^H, т. е.

- получение списков достижимых и недостижимых состояний сети;
- проверка попаданий сети в запрещенные состояния.

Этап 6. Построение компактного теста для функциональной верификации сети, т. е. для проверки выполнения всех переходов в графе G^H.

Этап 7. Корректировка VHDL-описаний, если не выполняются спецификации на проектирование.

3. Задача покрытия дуг ориентированного графа

Основной математической (комбинаторной) проблемой является проблема построения теста на этапе 6. Данная проблема связана с решением задачи обхода ориентированного графа G^H в целях покрытия всех дуг. Данная задача формулируется следующим образом.

Задача 2. Для заданного ориентированного графа G^H найти минимальный по длине цикл, содержащий все дуги графа.

Данная задача является известным случаем задачи о китайском почтальоне для ориентированных графов [18, 19]. На этапе 2 данная задача возникает для графов переходов компонентных автоматов G^0, G^1, ..., G^{p-1}, однако число вершин графа G^H значительно больше числа вершин каждого из компонентных автоматов и может достигать произведения чисел вершин графов компонентных автоматов.

4. Пример верификации параллельной сети автоматов

Компонентный конечный автомат (узел) может пребывать в одном из следующих трех состояний: *I* (*Invalid*); *S* (*Shared*); *M* (*Modified*). Вся параллельная сеть *H* автоматов, состоящая из *p* компонентных автоматов, принимает входные сигналы вида <*op*, *j*>, где *op* ∈ {*R* (*read*), *W* (*write*), *E* (*evict*)} — код операции; *j* ∈ {0, 1, ..., *p* − 1} — номер компонентного автомата (узла). Если при подаче входного воздействия <*R*, *j*> узел *j* находится в состоянии *I*, его состояние меняется на *S*; узлы, отличные от *j* и находящиеся в состоянии *M*, также переходят в состояние *S*. При подаче входного воздействия <*W*, *j*> узел *j* переходит в состояние *M*, а все остальные — в состояние *I*. При подаче входного воздействия <*E*, *j*> узел *j* переходит в состояние *I*, а состояния других узлов не изменяются. Требуется проверить, что два узла не могут одновременно находиться в состоянии *M* [20].

Как отмечено в работе [20], рассматриваемая модель сети *H* автоматов — это обобщенное описание протокола MSI [20], обеспечивающего когерентность распределенной памяти, а задача проверки того, что два узла не могут одновременно находиться в состоянии *M*, является задачей верификации этого протокола. Для написания VHDL-кода, описывающего поведение узла, представим его поведение в табл. 1. Это и есть, по сути, исходные спецификации на проектирование и аппаратную реализацию сети автоматов.

Таблица 1

Табличное описание поведения сети автоматов

Узел *i* (*i* = 0, 1, 2, ..., *p* − 1)	Входные воздействия		
	<*R*, *j*>	<*W*, *j*>	<*E*, *j*>
i = *j*	*I* → *S*	*M*, *S*, *I* → *M*	*M*, *S*, *I* → *I*
	S → *S*		
	M → *M*		
i ≠ *j*	*M* → *S*	*M*, *S*, *I* → *I*	*S* → *S*
	S → *S*		*M* → *M*
	I → *I*		*I* → *I*

Рассмотрим VHDL-описание (см. листинг 1 в приложении) параллельной сети (рис. 3) из двух (*p* = 2) компонентных автоматов, множество *Z* запрещенных состояний будет включать единственную вершину графа G^H, которая помечена <*M*, *M*>.

Этап 1. VHDL-описание компонентного автомата *msi_gate* (представлено на листинге 1 в приложении) удовлетворяет требованиям системы *Questa Sim*, которая может извлечь конечный автомат, найти его внутренние состояния и провести анализ выполненных переходов на графе внутренних состояний, что будет показано на этапе 3.

Этап 2. Проведем с помощью тестирующей программы (представлено на листинге 2 в приложении) моделирование сети автоматов на 10 000 псевдослучайных входных наборах вида <*op*, *j*>

Тестирующая программа обеспечивает генерацию псевдослучайных тестовых наборов вида <*op*, 0>, <*op*, 1> и функциональное покрытие, она написана с использованием средств VHDL-пакетов RandomPkg, CoveragePkg, находящихся в VHDL-библиотеке (Library OS-VVM).

Для генерации случайных значений входных сигналов *op* и *node* используются переменные *RndOp*, *RndNode* типа *RandomPType*. Метод *RandInt*(*min*, *max*) возвращает случайное значение (тип *integer*) из диапазона [*min*, *max*]. Для преобразования значения типа *integer в* тип *operation_type* используется выражение

```
op < = operation _ type'val(integer(RndOp.
RandInt(0, 2)));
```

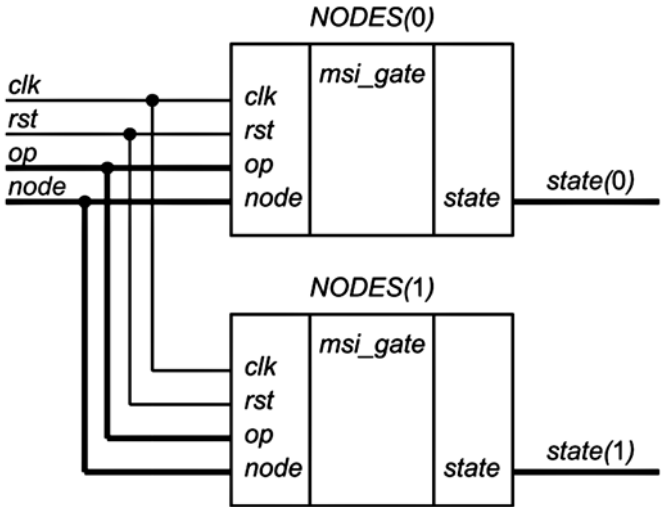


Рис. 3. Параллельная сеть из двух узлов *NODES*(0), *NODES*(1)

В тестирующей программе собирается покрытие состояний отдельных узлов 0 и 1, а также перекрестные состояния двух узлов. Для этого используются переменные *CovStateNode0*, *CovStateNode1* и *CovStateAllNodes* соответственно. Для задания моделей покрытия используются методы *AddBins* и *AddCross*. Сбор покрытия осуществляется с помощью метода *icover*. На каждом такте моделирования с помощью функции *write* сохраняются в текстовом файле *all_vectors.tst* входные воздействия (сигналы *op*, *node*) и состояния компонентных автоматов (сигналы *state(0)*, *state(1)*).

Текстовый файл *all_vectors.tst* для начальных 22 тактов моделирования имеет следующий вид:

```
e 1 i i
e 0 i i
r 0 s i
e 0 i i
e 1 i i
w 1 i m
e 0 i m
w 0 m i
e 1 m i
e 1 m i
w 0 m i
e 1 m i
w 0 m i
w 1 i m
r 1 i m
e 1 i i
w 1 i m
r 0 s s
e 0 i s
e 1 i i
```

Перед завершением работы тестирующей программы результаты покрытия печатаются в консоль (представлено на листинге 3 в приложении) с помощью метода *WriteBin*.

Как видно из листинга 3, все состояния отдельных узлов были многократно покрыты (значения *Count* больше нуля). Для перекрестного покрытия двух узлов видны состояния, которые не были покрыты (*Count = 0*). Это состояния 1-2 (<*S*, *M*>), 2-1 (<*M*, *S*>) и 2-2 (<*M*, *M*>), которые являются недостижимыми в данном тесте.

Этап 3. Выполнив моделирование сети автоматов с помощью тестирующей программы (см. листинг 2 в приложении), можно убедиться в том, что VHDL-модели компонентных автоматов написаны в соответствии с требованиями системы моделирования *Questa Sim*, по этим описаниям извлекаются графы переходов, визуализируются, что и позволяет сравнить их с исходными графами, являющимися спецификациями на проектирование VHDL-кода. Результаты верификации в системе моделирования *Questa Sim* узла 0 и узла 1 представлены на рис. 4 и 5 соответственно. Числа, помечающие дуги и вершины графов, изображенных на рис. 4 и 5, указывают число прохождений при моделировании соответствующей вершины либо дуги.

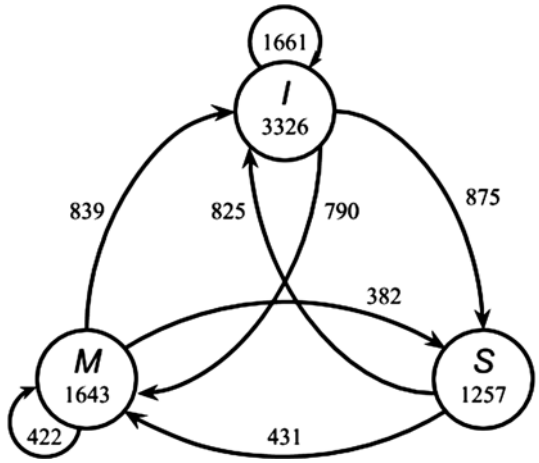


Рис. 4. Результат покрытия дуг в графе G^0 компонентного автомата A^0 (узла 0)

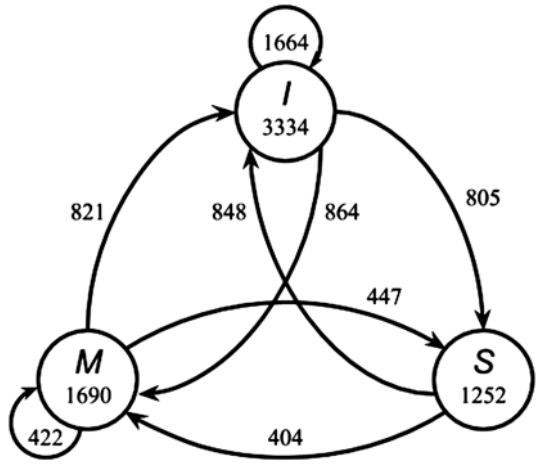


Рис. 5. Результат покрытия дуг в графе G^1 компонентного автомата A^1 (узла 1)

Этап 4. Результаты моделирования на начальных тактах даны в табл. 2, где во втором столбце даны входные воздействия, в третьем столбце — кортежи состояний компонентных автоматов. Наиболее часто сеть находилась в состоянии <*M, I*>. В правой части табл. 2 показаны проходимые циклы на подграфе ориентированного графа G^H. Рассмотрим в табл. 2 такты 7 и 8: в состоянии <*I*, *M*> сети на вход подается <*W*, 0>, тогда сеть переходит в состояние <*M*, *I*>. Из данных двух строк соответствующих текстовых файлов, полученных в результате моделирования, "извлекается" дуга графа G^H, исходящая из вершины <*I*, *M*> и заходящая в вершину <*M*, *I*>.

Полученный на начальном 21 такте подграф графа G^H показан на рис. 6.

По результатам моделирования на двух десятках тактов делать выводы о правильности либо неправильности VHDL-описания преждевременно. Поэтому было проведено моделирование сети на 10 000 псевдослучайных входных воздействиях, на рис. 7 показан граф G^H, полученный по результатам такого моделирования.

Таблица 2

Результат моделирования сети автоматов

Такт	Входные воздействия (тест)	Состояния компонентных автоматов	Циклы в графе G^H					
			1	**2**	**3**	**4**	**5**	**6**
0		I I	I I					
1	E 1	I I	I I	I I				
2	E 0	I I		I I	I I			
3	R 0	S I			S I			
4	E 0	I I			I I	I I		
5	E 1	I I				I I	I I	
6	W 1	I M					I M	
7	E 0	I M					I M	
8	W 0	M I					M I	
9	E 1	M I					M I	
10	E 1	M I					M I	
11	W 0	M I					M I	
12	E 1	M I					M I	
13	W 0	M I					M I	
14	W 1	I M					I M	
15	R 1	I M					I M	
16	E 1	I I					I I	I I
17	W 1	I M						I M
18	R 0	S S						S S
19	E 0	I S						I S
20	E 1	I I						I I

Этап 5. Анализ графа G^H. По графу G^H легко получить список выполненных переходов и списки достижимых и недостижимых состояний. Список <*I, I*>, <*S, I*>, <*I, M*>, <*M, I*>, <*S, S*>, <*I, S*> достижимых состояний сети составляют неизолированные вершины G^H. Список недостижимых состояний сети <*M, S*>, <*S, M*>, <*M, M*> составляют отсутствующие в третьем столбце табл. 2 кортежи состояний компонентных автоматов. Основное требование к состояниям сети автоматов выполнено — сеть никогда в данном сеансе моделирования не попадала в запрещенное состояние <*M, M*>.

Выделенные полужирным шрифтом строки в листинге 3 (см. приложение), по сути, информируют проектировщика о вершинах <*M, S*>, <*S, M*>, <*M, M*>, которые не попадают в граф G^H.

Этап 6. Построение компактных тестов для верификации сети автоматов. Результат обработки файлов входных воздействий и соответствующих кортежей состояний компонентных автоматов позволяет построить граф G^H.

Предлагаемая методика обобщена и применяется для произвольной (не обязательно параллельной) синхронной сети взаимодействующих автоматов. В кортеж состояний сети для каждого такта выводятся состояния компонентных автоматов, как и в случае параллельной сети.

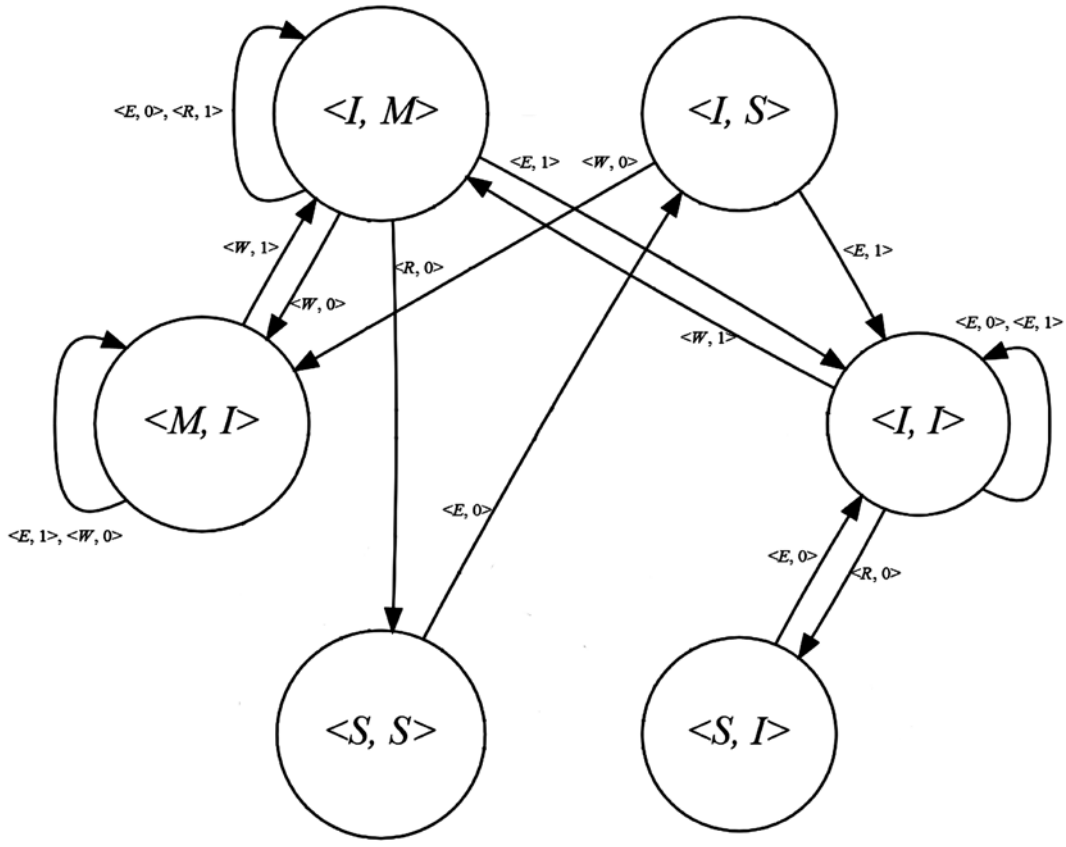


Рис. 6. Подграф графа G^H, полученный по результатам моделирования сети автоматов на 20 входных воздействиях из табл. 2

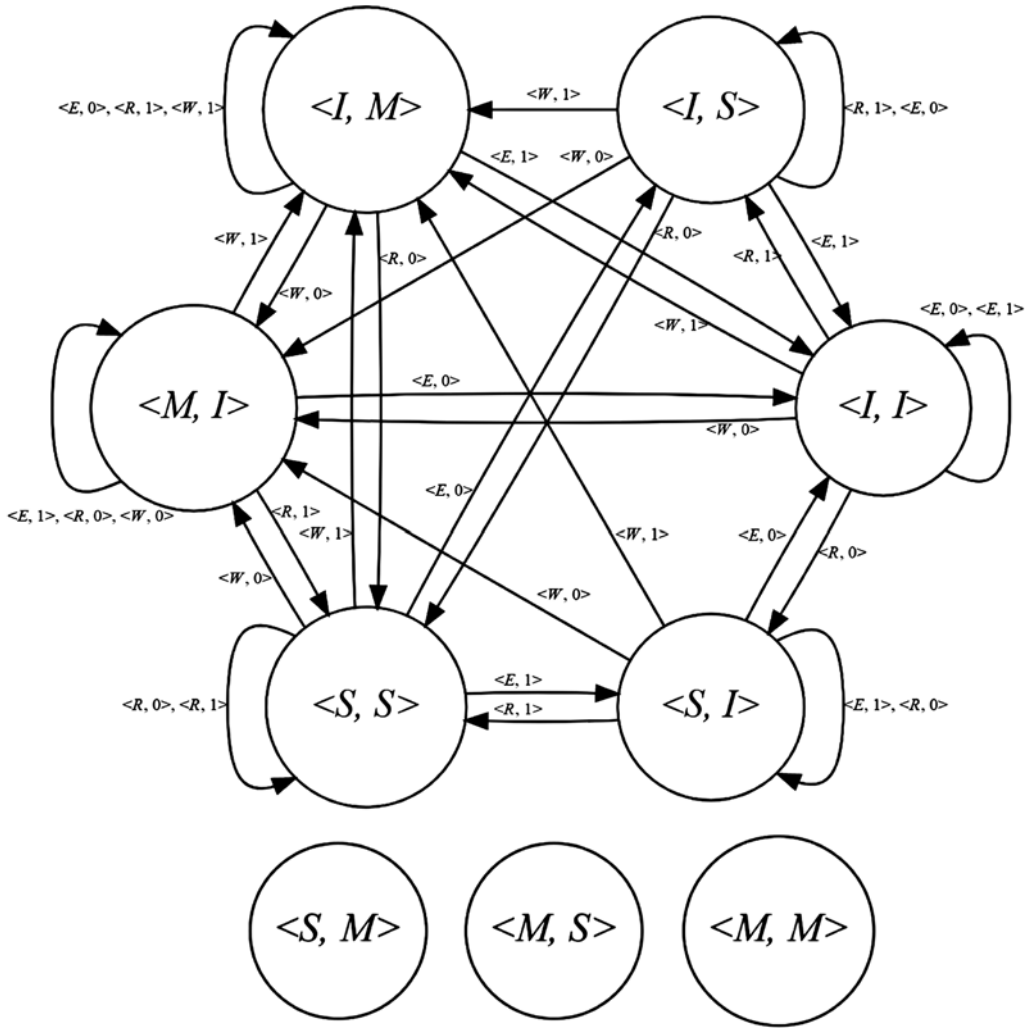


Рис. 7. Граф G^H для сети из двух узлов

Для автоматизированного построения графа G^H состояний сети компонентных автоматов и получения компактного функционального теста, обеспечивающего покрытие всех дуг данного графа, были разработаны соответствующие программы [9, 22], позволяющие обрабатывать тестовые последовательности с миллионами тестовых наборов. Входные тестовые наборы, соответствующие дугам, вошедшим в покрытие графа, образуют тест для функциональной верификации сети автоматов.

В рассматриваемом примере программа *Cover Graph* [9] по результатам моделирования сети автоматов строит компактный тест из 50 входных воздействий для покрытия 28 дуг графа G^H, приведенного на рис 7. Заметим, что в данном случае речь идет о покрытии шести (связанных между собой дугами) вершин данного графа. В результате моделирования ни одна из трех изолированных вершин <*S*, *M*>, <*M*, *S*>, <*M*, *M*> не была получена. Данный граф (рис. 7) соответствует случаю NUMBER_OF_NODES = 2 сети автоматов из двух узлов ($p = 2$). Результаты экспериментов для сети автоматов с большим числом (3, 4) компонентных автоматов приведены в табл. 3. Моделирование всех сетей осуществлялось на 10 000, 100 000, либо 5000 000 псевдослучайных входных наборов вида <*op*, *j*>. В работе [22] описан ряд программ, позволяющих получать лучшие решения по сравнению с решениями, получаемыми программой *CoverGraph* [9]. Например, применение лучшей из программ [22] для случая $p = 8$ (восемь компонентных автоматов) по результатам тестирования на 500 000 псевдослучайных наборов позволяет получить граф G^H, состоящий из 264 вершин и содержащий 4339 дуг, построенный компактный тест включает 11 661 набор. Таким образом, вместо 500 000 входных наборов можно использовать 11 661 набор компактного теста.

Этап 7. В данном примере предложенная методика верификации VHDL-описания сети автоматов ошибок не выявила, корректировка исходного описания не требуется.

Эксперименты показали: чтобы обеспечить максимальное покрытие дуг графа G^H, следует выполнять моделирование на возможно большем числе случайных входных воздействий. Построив компактные тесты для проверки переходов между состояниями, можно провести моделирование для проверки соответствия выходных сигналов сети автоматов тре-

Таблица 3

Построение тестов для параллельной сети автоматов

Число узлов p	Число псевдослучайных тестирующих наборов	Граф G^H		Число наборов компактного теста
		Число вершин	Число дуг	
2	10 000	6	28	50
3	100 000	11	74	199
4	100 000	20	176	550
8	500 000	264	4339	11 661

буемым значениям, и тем самым выполнить на основе моделирования еще один аспект функциональной верификаций — проверку не только требуемых переходов, но и требуемых реакций компонентных автоматов. Серьезной вычислительной проблемой является построение компактных тестов, нахождение которых связано с решением задач обхода ориентированных графов большой размерности.

Предложенный подход к верификации может быть применен, когда длины тестовых последовательностей достигают миллионов и десятков миллионов наборов, число параллельных состояний компонентных автоматов достигает сотен тысяч, что позволяет использовать его для верификации цифровых устройств достаточно большой сложности, когда "ручная" обработка результатов моделирования невозможна.

Заключение

Соблюдение несложных правил описания конечных автоматов позволяет провести в системе *Questa Sim* функциональное покрытие и визуализировать графы переходов компонентных автоматов. Предложенная методика верификации VHDL-описаний параллельных сетей конечных автоматов требует в каждом такте моделирования сохранять входные воздействия, состояния и выходные реакции как сети в целом, так и компонентных автоматов. Анализом результатов моделирования можно быстро провести верификацию сети автоматов, выявить предполагаемые запрещенные и недостижимые параллельные состояния и непокрытые переходы между состояниями сети автоматов.

Для формальной верификации VHDL-описаний сетей автоматов и проверки свойств проекта (например, проверки утверждения о том, что система попала в запрещенное состояние) требуются другие системы моделирования, например, система *Questa PropCheck*, позволяющая выполнять формальную верификацию свойств проекта, представленного на языке VHDL. Представленный в настоящей работе подход к верификации может быть использован при применении простых систем моделирования и может заменить дорогостоящие "фирменные" системы верификации в тех случаях, когда сложность каждого компонентного автомата ограничена несколькими десятками состояний.

Список литературы

1. **Шалыто А. А.** Парадигма автоматного программирования // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. Вып. 53. Автоматное программирование. 2008. С. 137—144.
2. **Кузьмин Е. В., Соколов В. А.** Моделирование, спецификация и верификация "автоматных" программ // Программирование. 2008. № 1. С. 38—60.
3. **Вельдер С. Э., Лукин М. А., Шалыто А. А., Яминов Б. Р.** Верификация автоматных программ. СПб.: Наука, 2011. 244 с.
4. **Поляков А. К.** Языки VHDL и VERILOG в проектировании цифровой аппаратуры. М.: СОЛОН-Пресс, 2003. 320 с.
5. **Ashenden P. J., Lewis J.** VHDL-2008. Just the New Stuff. Burlington, MA, USA. Morgan Kaufman Publishers, 2008. 909 p.
6. **Бибило П. Н., Авдеев Н. А.** Моделирование и верификация цифровых систем на языке VHDL. М.: ЛЕНАНД, 2017. 344 с.
7. **Лохов А., Рабоволюк А.** Комплексная функциональная верификация СБИС. Система Questa компании Mentor Graphics // Электроника: наука, технология, бизнес. 2007. № 3. С. 102—109.
8. **Чэнь М., Цинь К., Ку Х.-М., Мишра П.** Валидация на системном уровне. Высокоуровневое моделирование и управление тестированием. М.: Техносфера, 2014. 296 с.
9. **Бибило П. Н., Романов В. И.** Построение компактных тестов для функциональной верификации VHDL-описаний конечных автоматов // Управляющие системы и машины. 2017. № 1. С. 35—45.
10. **Закревский А. Д.** Параллельные алгоритмы логического управления. Минск: Ин-т техн. кибернетики НАН Беларуси, 1999. 202 с.
11. **Skliarova I., Sklyarov V., Sudnison A.** Design of FPGA-based Circuits using Hierarchical Finite State Machines. Tallinn: TUT Press, 2012. 242 p.
12. **Закревский А. Д., Поттосин Ю. В., Черемисинова Л. Д.** Логические основы проектирования дискретных устройств. М.: Физматлит, 2007. 589 с.
13. **Смолов С. А.** Обзор методов извлечения моделей из HDL-описаний Труды ИСП РАН. 2015. Т. 27, № 1. С. 97—123.
14. **Лебедев М. С., Смолов С. А.** Метод генерации функциональных тестов для HDL-описаний на основе проверки HLDD-моделей. // Проблемы разработки перспективных микро- и наноэлектронных систем. 2016. Сб. трудов / под общ. ред. акад. РАН А. Л. Стемпковского. М.: ИППМ РАН, 2016. Часть 2. С. 24—31.
15. **Карпов Ю. Г.** MODEL CHECKING. Верификация параллельных и распределенных программных систем. СПб.: БХВ-Петербург, 2010. 560 с.
16. **Слинкин Д. И.** Анализ современных методов тестирования и верификации проектов сверхбольших интегральных схем // Программные продукты и системы. 2017. Т. 30, № 3. С. 401—408.
17. **Бурдонов И. Б., Косачев А. С., Пономаренко В. Н., Шнитман В. З.** Обзор подходов к верификации распределенных систем. М.: Институт системного программирования РАН, 2003. Препринт № 16. 51 с. URL: http://www.ispras.ru/preprints/docs/prep_16_2006.pdf
18. **Thimbleby H.** The directed Chinese Postman Problem / Software Practice and Experience. 2003. Vol. 33 (11). P. 1081—1096.
19. **Бурдонов И. Б., Косачев А. С., Кулямин В. В.** Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай // Программирование. 2003. № 5. С. 11—30.
20. **Камкин А. С.** Проецирование систем переходов: преодоление комбинаторного взрыва при верификации параллельных систем // Программирование. 2015. № 6. С. 53—71.
21. **Sorin D. J., Hill M. D., Wood D. A.** A Primer on Memory Consistency and Cache Coherence. Morgan and Claypool, 2011. 195 p.
22. **Витязь К. А., Романов В. И.** Алгоритмы построения функциональных тестов для цифровой схемы на основе автоматной модели ее поведения. Танаевские чтения // Доклады Восьмой Международной научной конференции (27—30 марта 2018 г., Минск). Минск: ОИПИ НАН Беларуси, 2018. С. 52—56.

Приложение

Листинг 1. VHDL-описание сети автоматов

```
-- VHDL-пакет с объявлением вспомогательных типов
package msi_pkg is
type operation_type is (R, W, E);
type state_type is (I, S, M);
type state_vector_type is array (natural range <>) of state_type;
end msi_pkg;
```

```
-- VHDL-модель узла
library ieee;
use ieee.std_logic_1164.all;
use work.msi_pkg.all;

entity msi_gate is
  generic (
    NUMBER_OF_NODES: natural: = 2;
    NODE_NUMBER    : natural: = 0); -- Current node number
  port (
    clk  : in std_logic;
    rst  : in std_logic;
    op   : in operation_type;
    node : in natural range 0 to NUMBER_OF_NODES-1;
    state: out state_type);
end entity msi_gate;

architecture beh of msi_gate is
  signal state_s: state_type;
begin -- architecture beh
  p1: process (clk, rst) is
  begin -- process p1
    if rst = '1' then
      state_s <= I;
    elsif clk'event and clk = '1' then -- rising clock edge
      case op is
        when R =>                       -- read
          if (state_s = I and node = NODE_NUMBER)
            or (state_s = M and node / = NODE_NUMBER) then
           state_s <= S;                -- S
          end if;
        when W =>                       -- write
           if node = NODE_NUMBER then
             state_s <= M;              -- M
           else
             state_s <= I;              -- I
             end if;
        when E => -- evict
          if node = NODE_NUMBER then
            state_s <= I;               -- I
      end if;
     end case;
   end if;
  end process p1;
  state < = state_s;
end architecture beh;
```

```
-- Структурное VHDL-описание сети узлов
library ieee;
use ieee.std_logic_1164.all;
use work.msi_pkg.all;
entity msi_system is
  generic (
   NUMBER_OF_NODES: natural: = 2); -- Number of nodes in system
   port (
    clk  : in std_logic;
    rst  : in std_logic;
    op   : in operation_type;
    node : in natural range 0 to NUMBER_OF_NODES-1;
    state: out state_vector_type(NUMBER_OF_NODES — 1 downto 0));
end msi_system;
```

```
architecture beh of msi_system is
  component msi_gate
    generic (
      NUMBER_OF_NODES: natural;
      NODE_NUMBER    : natural);
    port (
      clk  : in std_logic;
      rst  : in std_logic;
      op   : in operation_type;
      node : in natural range 0 to NUMBER_OF_NODES-1;
      state: out state_type);
  end component;
begin -- beh
  l1: for i in 0 to NUMBER_OF_NODES — 1 generate
    NODES: msi_gate
      generic map (
        NUMBER_OF_NODES => NUMBER_OF_NODES,
        NODE_NUMBER = > i)
      port map (clk = > clk, rst => rst,
        op => op, node => node, state => state(i));
  end generate l1;
end beh;
```

Листинг 2. Тестирующая программа

```
library ieee;
use ieee.std_logic_1164.all;
use work.msi_pkg.all;
library osvvm;
use osvvm.RandomPkg.all;
use osvvm.CoveragePkg.all;
use std.textio.all;
------------------------------------------------------------------------------
entity msi_system_tb is
  generic (
    all_vectors_fname: string: = "all_vectors.tst";
    NUMBER_OF_NODES: natural: = 2);
end msi_system_tb;
------------------------------------------------------------------------------
architecture tb of msi_system_tb is
  component msi_system
    generic (
      NUMBER_OF_NODES: natural);
    port (
      clk  : in std_logic;
      rst  : in std_logic;
      op   : in operation_type;
      node : in natural range 0 to NUMBER_OF_NODES-1;
      state: out state_vector_type(NUMBER_OF_NODES — 1 downto 0));
end component;
  signal rst   : std_logic;
  signal op    : operation_type;
  signal node  : natural range 0 to NUMBER_OF_NODES-1;
  signal state : state_vector_type(NUMBER_OF_NODES — 1 downto 0);
  signal clk   : std_logic: = '1';
  signal start_sim: boolean: = false;

  shared variable RndOp  : RandomPType;
  shared variable RndNode: RandomPType;
  shared variable CovStateAllNodes: CovPType;
  shared variable CovStateNode0: CovPType;
  shared variable CovStateNode1: CovPType;
  file OUTFILE: text;
```

```
begin -- tb
  DUT: msi_system
    generic map (
      NUMBER_OF_NODES => NUMBER_OF_NODES)
    port map (
    clk => clk, rst => rst,
    op => op, node => node, state => state);
  -- clock generation
  clk <= '0' when start_sim = false else
         not clk after 10 ns;
  -- waveform generation
  WaveGen_Proc: process
  begin
    file_open(OUTFILE, all_vectors_fname, write_mode);
    -- CoverGroup initialisation
    CovStateNode0.SetName("Node 0 FSM states");
    CovStateNode1.SetName("Node 1 FSM states");
    CovStateAllNodes.SetName("All Nodes FSM states");
    CovStateNode0.AddBins(GenBin(0,2,3));
    CovStateNode1.AddBins(GenBin(0,2,3));
    CovStateAllNodes.AddCross(GenBin(0,2,3), GenBin(0,2,3));
    CovStateAllNodes.AddCross(ALL_ILLEGAL, ALL_ILLEGAL);

    start_sim < = false;
    RndOp.InitSeed(RndOp'instance_name);
    RndNode.InitSeed(RndNode'instance_name);
    -- insert signal assignments here
    rst <= '1';
    op <= operation_type'val(integer(RndOp.RandInt(0, 2)));
    node <= RndNode.RandInt(0, 1);
    start_sim < = true;
    wait for 11 ns;
    rst <= '0';
    for i in 0 to 10000 loop
      wait until clk'event and clk = '0';
      op < = operation_type'val(RndOp.RandInt(0, 2));
      node < = RndNode.RandInt(0, 1);
      CovStateAllNodes.icover(
        (state_type'pos(state(0)), state_type'pos(state(1))));
      CovStateNode0.icover(state_type'pos(state(0)));
      CovStateNode1.icover(state_type'pos(state(1)));
      write(OUTFILE, operation_type'image(op) & " " &
           to_string(node) & " " &
           state_type'image(state(0)) & " " &
           state_type'image(state(1)) & LF);
    end loop; -- i
    CovStateNode0.WriteBin;
    CovStateNode1.WriteBin;
    CovStateAllNodes.WriteBin;
    start_sim < = false;
    file_close(OUTFILE);
    wait;
  end process WaveGen_Proc;

end tb;
```

Листинг 3. Отчет о покрытии состояний узлов

```
# %%WriteBin: Node 0 FSM states
# %% Bin:(0) Count = 5076 AtLeast = 1 Weight = 1
# %% Bin:(1) Count = 2473 AtLeast = 1 Weight = 1
# %% Bin:(2) Count = 2452 AtLeast = 1 Weight = 1
#
# %%WriteBin: Node 1 FSM states
# %% Bin:(0) Count = 4998 AtLeast = 1 Weight = 1
# %% Bin:(1) Count = 2470 AtLeast = 1 Weight = 1
# %% Bin:(2) Count = 2533 AtLeast = 1 Weight = 1
#
# %%WriteBin: All Nodes FSM states
# %% Bin:(0) (0) Count = 1692 AtLeast = 1 Weight = 1
# %% Bin:(0) (1) Count = 851 AtLeast = 1 Weight = 1
# %% Bin:(0) (2) Count = 2533 AtLeast = 1 Weight = 1
# %% Bin:(1) (0) Count = 854 AtLeast = 1 Weight = 1
# %% Bin:(1) (1) Count = 1619 AtLeast = 1 Weight = 1
# %% Bin:(1) (2) Count = 0 AtLeast = 1 Weight = 1
# %% Bin:(2) (0) Count = 2452 AtLeast = 1 Weight = 1
# %% Bin:(2) (1) Count = 0 AtLeast = 1 Weight = 1
# %% Bin:(2) (2) Count = 0 AtLeast = 1 Weight = 1
```

Verification of VHDL Descriptions Networks of Synchronous Finite State Machines

N. A. Avdeev, avdeev_n@newman.bas-net.by, **P. N. Bibilo,** bibilo@newman.bas-net.by, The United Institute of Informatics Problems of the National Academy of Sciences of Belarus, Minsk, 220012, Belarus, **V. V. Korobkin,** vvk@niimvs.ru, Acad. Kalyaev Scientific Research Institute of Multiprocessor Computer Systems, Taganrog, 347928, Russian Federation, **A. E. Kolodenkova,** anna82_42@mail.ru, Samara State Technical University, Samara, 443100, Russian Federation

Corresponding author:

Bibilo Petr N., Head of Laboratory, The United Institute of Informatics Problems of the National Academy of Sciences of Belarus, 220012, Minsk, Belarus,
E-mail: bibilo@newman.bas-net.by

Received on April 24, 2018
Accepted on June 07, 2018

Finite state machines are widely applied in the development of digital systems for description of control logic nodes, microprocessors, interface circuits and so on. This work proposes verification procedure of VHDL description of parallel arrays of finite state machines in Questa Sim simulation system. The main advantage of Questa Sim is that the model of finite state machine (FSM) can be verified if it is written according to certain template. Verification is comprised of validating for compliance of VHDL description of finite state machine array with design specifications. The method utilizes the capabilities of the Questa Sim system, which makes it possible to identify the oriented graphs of the transitions of the component machines and to calculate the number of the arc passings in the graphs based on the results of simulation. However, the Questa Sim system does not recognize the FSM network and does not have the means to construct the tests based on the simulation results. Therefore, to solve these problems, it is suggested to store the simulation results — the sequence of input sets (stimuli) and the state tuples of the component machines, and to check the execution of transitions in the state graph of the machine network based on the sequences obtained,

and, thus, to conduct the verification. In addition, this article discusses an example of description of FSM's and FSM arrays using VHDL.

Keywords: *digital systems, VHDL descriptions, verification, simulation, finite state machines*

For citation:

Avdeev N. A., Bibilo P. N., Korobkin V. V., Kolodenkova A. E. Verification of VHDL Descriptions Networks of Synchronous Finite State Machines, *Programmnaya Ingeneria,* 2018, vol. 9, no. 7, pp. 291—304.

DOI: 10.17587/prin.9.291-304

References

1. **Shalyto A. A.** Paradigma avtomatnogo programmirovaniya (Automata-based programming paradigm), *Nauchno-tekhnicheskij vestnik Sankt-Peterburgskogo gosudarstvennogo universiteta informacionnyh tekhnologij, mekhaniki i optiki. Vypusk 53. Avtomatnoe programmirovanie*, 2008, pp. 137—144 (in Russian).

2. **Kuz'min E. V., Sokolov V. A.** Modelirovanie, spetsifikatsiya i verifikatsiya "avtomatnykh" programm (Simulation, specification and verification of automata-based programs), *Programmirovanie,* 2008, no. 1, pp. 38—60 (in Russian).

3. **Vel'der S. E., Lukin M. A., Shalyto A. A., Yaminov B. R.** *Verifikatsiya avtomatnykh programm* (Verification of automata-based programs), Nauka, St. Petersburg, 2011, 244 p. (in Russian).

4. **Polyakov A. K.** *Yazyki VHDL i VERILOG v proektirovanii tsifrovoi apparatury* (The VHDL and VERILOG languages in designing of digital hardware), SOLON-Press, Moscow, 2003, 320 p. (in Russian).

5. **Ashenden P. J., Lewis J.** *VHDL-2008. Just the New Stuff.* Burlington, MA, USA. Morgan Kaufman Publishers, 2008 909 p.

6. **Bibilo P. N., Avdeev N. A.** *Modelirovanie i verifikatsiya tsifrovykh sistem na yazyke VHDL* (Simulation and verification of digital systems using VHDL), LENAND, Moscow, 2017, 344 p. (in Russian).

7. **Lohov A., Rabovoljuk A.** Kompleksnaja funkcional'naja verifikacija SBIS. Sistema Questa kompanii Mentor Graphics (Complex functional verification of VLSI. Mentor Graphics Questa system), *Jelektronika: nauka, tehnologija, biznes,* 2007, no. 3, pp. 102—109 (in Russian).

8. **Chen M., Qin K., Ku H.-M., Mishra P.** *Validation at the system level. High-level simulation and testing management,* Moscow, Tekhnosfera, 2014, 296 p. (in Russian).

9. **Bibilo P. N., Romanov V. I.** Postroenie kompaktnykh testov dlya unktsional'noi verifikatsii VHDL-opisanii konechnykh avtomatov (Development of compact tests for functional verification of VHDL descriptions of final state machines), *Upravlyayushchie sistemy i mashiny.* 2017, no. 1, pp. 35—45 (in Russian).

10. **Zakrevskii A. D.** *Parallel'nye algoritmy logicheskogo upravleniya* (Parallel algorithms of logic control), Minsk, In-t tekhn. kibernetiki NAN Belarusi, 1999, 202 p. (in Russian).

11. **Skliarova I.,Sklyarov V., Sudnison A.** *Design of FPGA-based Circuits using Hierarchical Finite State Machines.* Tallinn,TUT Press, 2012, 242 p.

12. **Zakrevskii A. D., Pottosin Yu. V., Cheremisinov L. D.** *Logicheskie osnovy proektirovaniya diskretnykh ustroistv* (Logical foundations of designing of discrete hardware), Moscow, Fizmatlit, 2007, 589 p. (in Russian).

13. **Smolov S. A.** Obzor metodov izvlecheniya modeley iz HDL-opisaniy (Overview of methods for extracting models from HDL-descriptions), *Trudy ISP RAN,* 2015, vol. 27, no. 1, pp. 97—123 (in Russian).

14. **Lebedev M. S., Smolov S. A.** Metod generatsii funktsional'nykh testov dlya HDL-opisaniy na osnove proverki HLDD-modeley (Method for generating functional tests for HDL-descriptions based on testing of HLDD models), *Problemy razrabotki perspektivnykh mikro- i nanoelektronnykh sistem:* sb. trudov pod obshch. red. akad. RAN A. L. Stempkovskogo, Moscow, IPPM RAN, 2016, part 2, pp. 24—31 (in Russian).

15. **Karpov Yu. G.** *MODEL CHECKING. Verifikatsiya parallel'nykh i raspredelennykh programmnykh system* (Verification of parallel and distributed software systems), Saint Petersburg, BKHV-Peterburg, 2010, 560 p. (in Russian).

16. **Slinkin D. I.** Analiz sovremennyh metodov testirovanija i verifikacii proektov sverhbol'shih integral'nyh shem (Analysis of modern methods of testing and verification of projects of very-large-scale integrated circuits), *Programmnye produkty i sistemy,* 2017, vol. 30, no. 3, pp. 401—408 (in Russian).

17. **Burdonov I. B., Kosachev A. S., Ponomarenko V. N., Shnitman V. Z.** *Obzor podhodov k verifikacii raspredelennyh* system (Overview of approaches to verification of distributed systems), Moscow, Institut sistemnogo programmirovanija RAN. 2003. Preprint no 16, 51 p., available at: http://www.ispras.ru/preprints/docs/prep_16_2006.pdf (in Russian).

18. **Thimbleby H.** The directed Chinese Postman Problem, *Software Practice and Experience*, 2003, vol. 33 (11), pp. 1081—1096.

19. **Burdonov I. B., Kosachev A. S., Kulyamin V. V.** Neizbytochnyye algoritmy obkhoda oriyentirovannykh grafov. Determinirovannyy sluchay. (Inexhaustible algorithms for traversing oriented graphs. Deterministic case), *Programmirovaniye*, 2003, no. 5. pp. 11—30 (in Russian).

20. **Kamkin A. S.** Proetsirovanie sistem perehodov: preodolenie kombinatornogo vzryva pri verifikatsii parallel`nykh system (Designing of transition systems: overcoming of combinatorial explosion upon verification of parallel arrays), *Programmirovanie*, 2015, no. 6, pp. 53—71 (in Russian).

21. **Sorin D. J., Hill M. D., Wood D. A.** *A Primer on Memory Consistency and Cache Coherence.* Morgan and Claypool, 2011, 195 p.

22. **Vitjaz' K. A., Romanov V. I.** Algoritmy postroenija funkcional'nyh testov dlja cifrovoj shemy na osnove avtomatnoj modeli ee povedenija (Algorithms for constructing functional tests for a digital circuit based on an automatic model of its behavior**)**, *Tanaevskie chtenija, Doklady Vos'moj Mezhdunarodnoj nauchnoj konferencii* (27—30 March 2017, Minsk), Minsk, OIPI NAN Belarusi, 2018, pp. 52—56 (in Russian).

УДК 004.921 DOI: 10.17587/prin.9.305-310

А. В. Мальцев, канд. физ.-мат. наук, вед. науч. сотр., e-mail: avmaltcev@mail.ru,
ФГУ "ФНЦ Научно-исследовательский институт системных исследований РАН", Москва

Определение коллизий мелкоразмерных частиц с виртуальными объектами на основе буфера глубины

Предложены распределенные методы определения в реальном времени точек столкновения мелкоразмерных частиц, движущихся прямолинейно по параллельным траекториям (например, капли воды в ливневом дожде или крупные снежинки при снегопаде в безветренную погоду), с трехмерными объектами виртуальной среды. Рассматриваемые решения основаны на использовании буфера глубины видеокарты и архитектуры параллельных вычислений CUDA на современных графических процессорах.

Ключевые слова: *виртуальный объект, частица, графический процессор, визуализация, реальное время, шейдер, CUDA*

Введение

Существенная составляющая моделирования виртуальной среды на компьютере заключается в реализации взаимодействия содержащихся в данной среде трехмерных объектов. Одним из видов таких взаимодействий являются столкновения, или как их еще называют — коллизии виртуальных объектов друг с другом. Прежде чем моделировать и визуализировать результат столкновения, необходимо установить факт наличия коллизии и вычислить точки соприкосновения взаимодействующих элементов. Трудоемкость, а также методы решения этих задач во многом зависят от сложности геометрии трехмерных объектов и возможности описания их различными аппроксимирующими примитивами: параллелепипедами, сферами и т. д. [1, 2].

С вычислительной точки зрения особо сложным случаем является ситуация, когда виртуальная сцена включает в себя объекты, представляющие совокупность большого количества мелкоразмерных элементов. Это происходит, например, при моделировании атмосферных осадков, таких как дождь и снег, облаков пыли, струй жидкости и пены [3—5]. Подобные объекты без четкой геометрической границы обычно моделируют с помощью систем частиц (СЧ) [6], число которых достигает сотен тысяч и даже миллионов. Для каждой из частиц системы необходимо проверять наличие коллизии с другими объектами сцены и, если таковая имеет место, проводить расчет координат точки столкновения с поверхностью взаимодействующего объекта. При этом обеспечение моделирования динамики и визуализации объектов виртуальной среды в масштабе реального времени, необходимом для корректной работы систем виртуального окружения и имитационно-тренажерных комплексов управления сложными техническими системами, является весьма затруднительным.

Одним из эффективных способов решения задачи выявления коллизий между виртуальными объектами являются так называемые *image-based*-подходы. Они основаны на анализе двумерных изображений виртуальных сцен, полученных путем проецирования трехмерной сцены на некоторую плоскость. К таким изображениям относятся и карты глубины (*depth map*). Например, подход, совмещающий использование карт глубины и буфера трафарета (*stencil buffer*), представлен в работе [7]. Однако предложенные в ней решения не адаптированы под системы частиц и реализацию на многоядерных графических процессорах (GPU). В работе [8] описаны методы моделирования систем частиц с помощью шейдеров и применение карт глубины для поиска коллизий этих СЧ с виртуальными объектами. Негативным фактором использования шейдеров для осуществления всех этапов моделирования СЧ является необходимость организации структур данных, хранящих параметры частиц (положение, скорость, время жизни и т. д.), в виде некоторого массива текстур. Каждую из текстур, как правило, надо перезаписывать с помощью отдельного прохода рендеринга, как это и описано в работе [8].

В настоящей работе предложены новые распределенные методы и алгоритмы поиска коллизий СЧ с другими объектами трехмерной виртуальной среды, основанные на картах глубины. Преимуществом перед ранее известными методами является применение для моделирования СЧ и решения задачи поиска столкновений возможностей архитектуры параллельных вычислений CUDA на современных многоядерных GPU. Такой подход позволяет не привязывать параллельные алгоритмы расчета параметров частиц системы к структуре графического конвейера, как в случае со шейдерной реализацией, а также дает возможность хранить параметры частиц в виде клас-

сических массивов в видеопамяти, не прибегая к записи их в дополнительные текстуры. Расчет параметров СЧ в предлагаемом решении базируется на методах и алгоритмах, изложенных в работе [9]. Синтез карты глубины и непосредственно визуализация частиц выполняются с использованием шейдеров. Новизной предлагаемого подхода поиска коллизий СЧ с объектами также является использование для рендеринга карты глубины фиктивной виртуальной камеры, размещаемой в центре эмиттера СЧ. Предлагаемые решения обеспечивают в масштабе реального времени поддержку визуализации высокополигональных виртуальных сцен, включающих несколько миллионов частиц. Рассмотрим эти решения подробнее.

1. Получение карты глубины сцены относительно эмиттера

В настоящей работе будем рассматривать такой класс СЧ, элементы которых после испускания из эмиттера имеют прямолинейные и параллельные траектории движения. В качестве примера объектов виртуальной среды, моделируемых системами такого типа, могут выступать ливневый дождь и снегопад в безветренную погоду. Заметим, что условие параллельности траекторий движения частиц на практике может быть ослаблено допущением небольших погрешностей, т. е. траектории могут быть близкие к параллельным.

Предлагаемый подход к определению коллизий частиц с объектами виртуальной среды заключается в использовании возможностей буфера глубины видеокарты. Его суть состоит в размещении фиктивной виртуальной камеры C в центральной точке P_e эмиттера системы частиц (рис. 1). Пусть эмиттер имеет вид прямоугольника размером $W \times H$, а направление движения частиц определяется ветром **D**, перпендикулярным плоскости самого эмиттера. Тогда направление

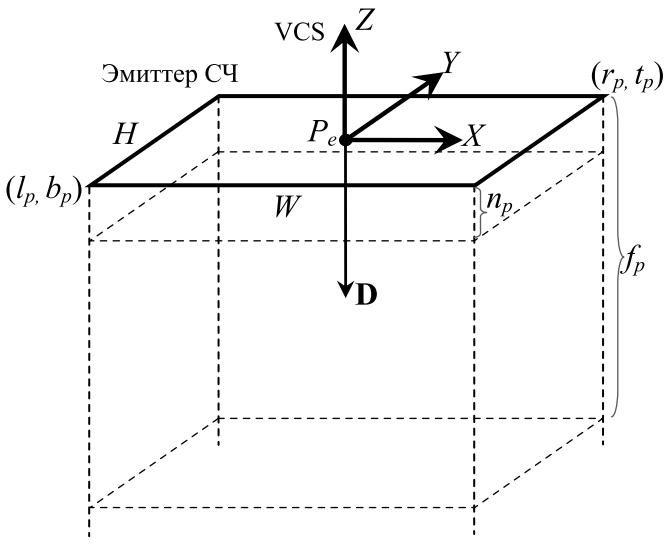


Рис. 1. Виртуальная камера

взгляда камеры C (отрицательное направление оси Z видовой системы координат VCS-камеры) установим совпадающим с **D**, ее верх (ось Y системы VCS) направим параллельно меньшей стороне с размером H. Также необходимо выбрать для камеры ортографическое проецирование. Чтобы область видимости созданной камеры охватывала область распространения частиц, определим левую l_p, правую r_p, нижнюю b_p и верхнюю t_p отсекающие плоскости по границам эмиттера:

$$l_p = -W/2,\ r_p = W/2,\ b_p = -H/2,\ t_p = H/2.$$

Ближнюю плоскость отсечения при этом расположим на небольшом расстоянии n_p от эмиттера. В нашем исследовании $n_p = 0{,}1$ (в единицах измерения, задаваемых при создании виртуальной сцены). Расстояние f_p до дальней отсекающей плоскости вычисляется, исходя из максимального пути, который может преодолеть произвольная частица рассматриваемой системы за время t своей жизни. Отметим, что описанная виртуальная камера жестко связана с эмиттером, т. е. должна повторять все его перемещения и повороты.

Каждый раз перед расчетом состояния системы частиц будем выполнять рендеринг трехмерной сцены из камеры C. Поскольку для решаемой задачи RGB-изображение объектов, получаемое камерой, не имеет значения, а важно только их расположение и геометрия, то расчет освещенности и применение каких-либо материалов и текстур не требуется. Поэтому на данном этапе предполагается использование стандартного механизма визуализации OpenGL с отключенной записью в цветовой буфер (отключение выполняется функцией *glColorMask*). В данном случае такой подход более эффективен по скорости, чем применение собственных шейдеров. Важным результатом визуализации будет являться информация, записанная в z-буфер (буфер глубины). Поскольку для камеры C установлено ортографическое проецирование, а ее поле видимости точно охватывает эмиттер СЧ, то каждый пиксел z-буфера будет хранить расстояние d от соответствующей ему точки эмиттера до ближайшего к этому эмиттеру объекта по лучу, который параллелен взгляду камеры и вектору **D** движения частиц (если луч не пересекает ни одного объекта вплоть до дальней плоскости отсечения камеры, то d будет соответствовать расстоянию до этой плоскости). Фактически z-буфер содержит карту глубины виртуальной сцены относительно эмиттера СЧ. Для дальнейшей работы нам необходимо сохранить буфер глубины в виде текстуры. Чтобы избежать трудоемких операций копирования данных, описанный рендеринг следует проводить не в стандартный z-буфер текущего контекста, а напрямую в заранее созданную текстуру T_{depth}, имеющую тип данных *GL_DEPTH_COMPONENT32F*. Такой подход можно реализовать с помощью технологии FBO (*framebuffer objects* [10]).

Отметим, что важным параметром является разрешение (в пикселах) генерируемой текстуры глубины T_{depth}. Если использовать текстуру с низким

разрешением для эмиттера с большой площадью, то в результате визуализации из камеры C объекты малого размера могут занимать в этой текстуре всего несколько пикселов или не отображаться вовсе, а остальные объекты будут иметь резкие ломаные границы. При этом точность дальнейшего определения коллизий частиц с объектами, особенно на их границах, может значительно снизиться. Для обеспечения приемлемого результата следует выбирать размер W_T большей стороны текстуры T_{depth} не менее 1024 пикселов. Размер меньшей стороны при этом будет равен $H_T = [W_T(H/W)]$ пикселов, где квадратные скобки обозначают взятие целой части числа.

2. Поиск столкновений частиц с объектами на GPU

Расчет всех требуемых параметров системы частиц будем выполнять с использованием графического процессора, поддерживающего архитектуру параллельных вычислений CUDA [11]. Распределенные методы и алгоритмы решения данной задачи в масштабе реального времени подробно изложены в работе [9]. Каждая частица при этом обрабатывается отдельным потоком на GPU. В нем же будем проводить поиск возможных столкновений частицы с объектами виртуальной сцены, а также моделирование поведения частицы после столкновения.

Для поиска коллизий частиц рассматриваемой системы с другими объектами виртуальной среды необходимо обеспечить в CUDA доступ к данным карты глубины T_{depth} (см. разд. 1). К сожалению, формат текстур *GL_DEPTH_COMPONENT32F*, ассоциированный с z-буфером, в настоящее время не поддерживается в архитектуре CUDA. Чтобы преодолеть эту трудность будем использовать дополнительный фрагментный GLSL-шейдер, преобразующий карту глубины в CUDA-совместимый формат данных *GL_R32F* (одноканальная текстура, каждый пиксел которой хранит 32-битное вещественное число). Данный шейдер принимает на вход текстуру T_{depth} и проводит рендеринг с использованием технологии FBO в новую текстуру T_r такого же размера в пикселах, как и T_{depth}, но имеющую требуемый формат.

Помимо изменения формата текстуры тот же шейдер целесообразно задействовать для еще одного преобразования данных. Как уже было упомянуто в разд. 1, каждый пиксел карты глубины хранит расстояние d от эмиттера СЧ до некоторой точки B ближайшего к нему объекта виртуальной сцены по лучу, перпендикулярному эмиттеру и проходящему через данный пиксел, или до дальней плоскости отсечения. Однако расстояние d выражено в единицах экранной системы координат (СК), которая связана с фиктивной виртуальной камерой C, поэтому $d \in [0{,}0,\ 1{,}0]$. В рассматриваемом подходе определения коллизий частиц и объектов виртуальной среды удобнее работать с расстоянием r_b от эмиттера до B по оси Z видовой системы координат VCS камеры C. Оно равно z-координате $B_{vsc,z}$ (в СК VCS) точки B, взятой с отрицательным знаком (рис. 2). Принимая во внимание, что при визуализации карты глубины выполнялось ортографическое проецирование, расчет r_b по значению глубины d выполняется по формуле

$$r_b = -B_{vcs,z} = n_p + d\left(f_p - n_p\right),$$

где n_p и f_p — расстояния соответственно до ближней и дальней плоскостей отсечения камеры, которые были заданы ранее (см. разд. 1). Вычисленные для всех пикселов T_{depth} значения r_b записываются шейдером в соответствующие пикселы результирующей текстуры T_r, передаваемой в CUDA-программу (ядро) расчета параметров частиц.

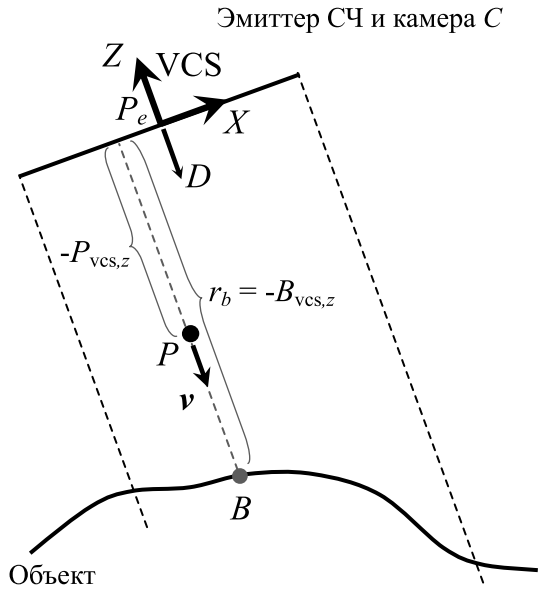


Рис. 2. Анализ глубины частиц относительно эмиттера

Отметим, что описанную процедуру конвертации данных можно было бы совместить с предыдущим этапом — визуализацией сцены из виртуальной камеры C, заменив стандартный механизм визуализации OpenGL на собственные шейдеры. Однако указанные преобразования в таком случае применялись бы ко всем обрабатываемым во фрагментном шейдере фрагментам, включая отбрасываемые в момент проведения теста глубины. Такой подход существенно повышает вычислительную нагрузку на GPU при рендеринге высокополигональных сцен, поскольку число отбрасываемых фрагментов в них довольно велико. В предлагаемой же реализации конвертация данных применяется только к уже прошедшим тест глубины фрагментам, определяющим глубину сцены относительно эмиттера.

Доступ к GL-текстуре T_r в CUDA-ядре осуществляется с помощью механизма CUDA OpenGL interoperability [11]. Он позволяет избежать копирования данных из одной области видеопамяти в другую с использованием центрального процессора и шины PCI Express. Рассчитав в текущий момент времени

новое положение P_{wcs} и скорость $\boldsymbol{v}$ некоторой частицы P в мировой СК WCS, ядро определяет ее координаты P_{vcs} (рис. 2) в видовой системе VCS путем умножения P_{wcs} на видовую матрицу M_v камеры C (матрица перехода от WCS к VCS):

$$P_{vcs} = M_v P_{wcs}.$$

Далее вычисляются экранные координаты P_s этой же частицы:

$$P_s = M_b M_{pr} P_{\text{vcs}},$$

где M_{pr} — проекционная матрица для камеры C, осуществляющая преобразование из СК VCS в систему координат нормализованного объема видимости (NDCS). Матрица

$$M_b = \begin{pmatrix} 0,5 & 0 & 0 & 0,5 \\ 0 & 0,5 & 0 & 0,5 \\ 0 & 0 & 0,5 & 0,5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

выполняет преобразование СК NDCS, при котором ее начало перемещается в вершину (−1, −1, −1) куба видимости, а оси растягиваются в 2 раза (рис. 3).

По паре координат ($P_{s,x}$, $P_{s,y}$) ядро проводит выборку из текстуры T_r значения расстояния r_b от эмиттера до ближайшего объекта виртуальной сцены по линии движения частицы P и сравнивает его с расстоянием от эмиттера до самой частицы, равным $-P_{\text{vcs},z}$ (см. рис. 2). Коллизия частицы и объекта имеет место, если $r_b \leqslant -P_{vcs,z}$. Координаты точки B столкновения в мировой СК WCS можно найти по формуле:

$$B_{wcs} = M_v^{-1}\left(P_{vcs,x}, P_{vcs,y}, -r_b\right).$$

При выявлении коллизии необходимо провести ее обработку, которая зависит от вида столкнувшихся объектов. Например, частицы падающего снега, столкнувшиеся с чем-либо, можно просто удалять из СЧ, а для капель дождя моделировать разбрызгивание при ударе о поверхность.

3. Результаты

На основе описанных методов и подходов были разработаны программные модули для систем визуализации трехмерных виртуальных сцен, применяющихся в имитационно-тренажерных комплексах и системах виртуального окружения. Они обеспечивают распределенный поиск и обработку коллизий виртуальных объектов, образованных множеством мелкоразмерных элементов, с другими объектами виртуальной среды в масштабе реального времени. Создание модулей выполнялось с применением графической библиотеки OpenGL, шейдерного языка GLSL 4.0 и аппаратно-программной архитектуры параллельных вычислений CUDA версии 7.0. Апробацию предложенных программных решений проводили в составе системы визуализации GLView [12], разработанной в ФГУ "ФНЦ Научно-исследовательский институт системных исследований РАН".

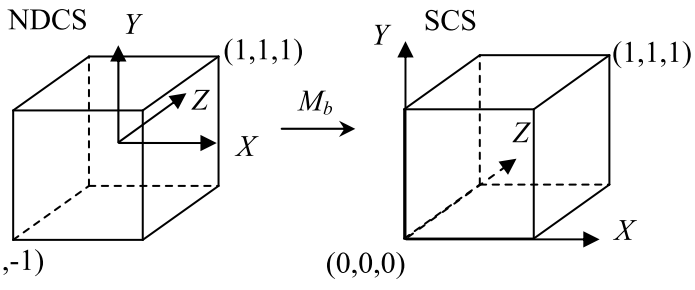


Рис. 3. Преобразование в экранные координаты

Рис. 4. Тестовая сцена трехмерного виртуального полигона с моделированием дождя

Оценку производительности разработанных методов и алгоритмов проводили путем измерения частоты генерации кадров (FPS) при визуализации сцены трехмерного виртуального полигона, включающей 650 тысяч треугольников (рис. 4). С помощью системы частиц в данной сцене моделировались атмосферные осадки в виде дождя различной интенсивности, частицы-капли которого уничтожались при столкновении с объектами. Эксперименты выполняли на персональном компьютере с процессором Intel Core i7 (3 ГГц) и видеоадаптером NVIDIA GeForce GTX 1080 Ti (3584 ядер CUDA). В ходе тестовых испытаний изменяли число моделируемых частиц (1, 2 и 4 млн), итоговое разрешение получаемого изображения (форматы Full HD 1920×1080 и UHD 4K 3840×2160), а также размер буферов и текстур под данные карты глубины эмиттера СЧ (2048×2048 и 4096×4096 пикселов). Поскольку использованная для каждой частицы трехмерная модель представляет собой тетраэдр и содержит четыре треугольника, то при рассматриваемых количествах частиц общее число треугольных полигонов в сцене составляло соответственно 4,65, 8,65 и 16,65 млн. Результаты проведенных экспериментов сведены в табл. 1 и 2.

На рис. 5 представлен пример моделирования снегопада в упомянутой ранее трехмер-

Таблица 1

Значение FPS, кадров/с, при визуализации сцены виртуального полигона в формате UHD 4K

Размер карты глубины для эмиттера СЧ	Количество частиц, млн/общее число полигонов в сцене, млн		
	1/4,65	2/8,65	4/16,65
Не используется (без поиска коллизий)	55	41	28
2048 × 2048 пикселов	51	38	27
4096 × 4096 пикселов	48	37	25

Таблица 2

Значение FPS, кадров/с, при визуализации сцены виртуального полигона в формате Full HD

Размер карты глубины для эмиттера СЧ	Количество частиц, млн/общее число полигонов в сцене, млн		
	1/4,65	2/8,65	4/16,65
Не используется (без поиска коллизий)	62	48	32
2048 × 2048 пикселов	59	45	30
4096 × 4096 пикселов	58	43	29

ной сцене виртуального полигона в условиях штиля. Наблюдатель находится в укрытии под бетонными плитами, которые препятствуют проникновению туда падающего снега, реализуемого с помощью системы частиц.

Рис. 5. Снег не проникает в укрытие под бетонными плитами

Заключение

Рассмотрена задача поиска коллизий объектов, образованных множеством мелкоразмерных частиц, с другими трехмерными объектами виртуальной среды. Предложены методы и алгоритмы решения данной задачи для систем частиц, движущихся прямолинейно по траекториям, близким к параллельным. Разработанные подходы основаны на широком использовании распределенных вычислений на современных многоядерных графических процессорах, что обеспечивает выполнение поиска описанных коллизий в масштабе реального времени. Апробация разработанных методов показала их применимость в области систем визуализации для имитационно-тренажерных комплексов и систем виртуального окружения.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 16-07-00796.

Список литературы

1. **Ming C. Lin, Gottschalk S.** Collision detection between geometric models: a survey // Proc. of IMA conference on mathematics of surfaces. 1998. Vol. 1. P. 602—608.

2. **Трушин А. М.** Определение коллизий аппроксимирующих сфер и прямоугольных параллелепипедов в системах трехмерного моделирования // Программные продукты и системы. 2015. № 4. С. 105—109.

3. **Мальцев А. В.** Моделирование атмосферных осадков в трехмерных сценах с использованием CUDA // Информационные технологии и вычислительные системы. 2015. № 2. С. 31—39.

4. **Tan J., Fan X.** Particle system based snow simulating in real time // Procedia Environmental Sciences 10. 2011. Vol. 10. P. 1244—1249.

5. **Мальцев А. В.** Реализация на GPU струй воды и пены в трехмерных виртуальных сценах на основе систем частиц // Информационные технологии и вычислительные системы. 2017. № 1. С. 40—45.

6. **Reeves W. T.** Particle systems — a technique for modeling a class of fuzzy objects // In Computer Graphics (Proc. SIGGRAPH). 1983. P. 359—376.

7. **Baciu G., Wong S.-K.** Image-based techniques in a hybrid collision detector // In IEEE Trans. on Visualization and Computer Graphics. 2003. Vol. 9. P. 254—271.

8. **Kolb A., Latta L., Rezk-Salama C.** Hardware-based Simulation and Collision Detection for Large Particle Systems // In Proc. of the ACM SIGGRAPH/ EUROGRAPHICS conference on Graphics hardware. 2004. P. 123—131.

9. **Мальцев А. В.** Реализация системы частиц в реальном времени на GPU // Программные продукты и системы. 2014. № 4. С. 57—62.

10. **Framebuffer** Object. URL: https://www.khronos.org/opengl/wiki/Framebuffer_Object (дата обращения 27.03.2018).

11. **CUDA** C Programming Guide. NVIDIA Corporation, 2018. URL: http://docs.nvidia.com/cuda/cuda-c-programming-guide (дата обращения 27.03.2018).

12. **Михайлюк М. В., Торгашев М. А.** Система визуализации "GLView" для имитационно-тренажерных комплексов и систем виртуального окружения // Труды 25-й Международной научной конференции "Графикон". 2015. С. 96—101.

Collision Detection of Small-sized Particles with Virtual Objects Based on Depth Buffer

A. V. Maltsev, avmaltcev@mail.ru, Scientific Research Institute for System Analysis of the Russian Academy of Sciences, Moscow, 117218, Russian Federation

Corresponding author:

Maltsev Andrey V., Ph.D., Leading Researcher, Scientific Research Institute for System Analysis of the Russian Academy of Sciences, Moscow, 117218, Russian Federation
E-mail: avmaltcev@mail.ru

Received on April 19, 2018
Accepted on May 15, 2018

This paper presents distributed methods to calculate in real-time collision points of small-sized particles, rectilinearly moving along parallel trajectories (for example, water droplets in heavy rain or large snowflakes in windless weather), with three-dimensional objects of virtual environment. Considered solutions are based using the CUDA parallel computing architecture on modern graphics processors and virtual scene depth map generated by means of hardware depth buffer. To create depth map, a fictitious virtual camera is placed to central point of particle system's emitter and scene is rendered from it. The eye vector of the camera (negative Z-axis of view coordinate system) is aligned with particle motion direction. RGB-channels are blocked for writing data, and depth values from z-buffer are stored in single channel floating point texture that is depth map. This texture is converted to format that supported by CUDA, and passed to kernel which calculates current states and positions of particles and also finds their collisions with virtual objects. Based on proposed methods and algorithms, software modules were created and tested in the visualization system developed in Scientific Research Institute for System Analysis of the Russian Academy of Sciences. Complex virtual scenes with different number of particles (from several thousand to several million) were used to evaluate efficiency of the modules and algorithms. It was shown that proposed solutions meet high requirements of real-time visualization. The modules can be used in virtual environment systems, training complexes, virtual laboratories, etc.

Keywords: *virtual object, particle, graphics processor, visualization, real time, shader, CUDA*

Acknowledgements: *This research was carried out with the financial support of the Russian Foundation for Basic Research in the framework of the scientific project no. 16-07-00796*

For citation:

Maltsev A. V. Collision Detection of Small-sized Particles with Virtual Objects Based on Depth Buffer, *Programmnaya Ingeneria*, 2018, vol. 9, no. 7, pp. 305—310.

DOI: 10.17587/prin.9.305-310

References

1. **Ming C. L., Gottschalk S.** Collision detection between geometric models: a survey, *Proc. of IMA conference on mathematics of surfaces*, 1998, vol. 1, pp. 602—608.

2. **Trushin A. M.** Opredelenie kollizii approksimiruiushchikh sfer i priamougolnykh parallelepipedov v sistemakh trekhmernogo modelirovaniia (Collision detection for bounding spheres and rectangular parallelepipeds in 3D modeling systems), *Programmnye produkty i sistemy*, 2015, no. 4, pp. 105—109 (in Russian).

3. **Maltsev A. V.** Modelirovanie atmosfernykh osadkov v trekhmernykh stsenakh s ispol'zovaniem CUDA (Simulation of atmospheric condensation at 3D scenes by using CUDA), *Informatsionnye tekhnologii i vychislitel'nye sistemy*, 2015, no. 2, pp. 31—39 (in Russian).

4. **Tan J., Fan X.** Particle system based snow simulating in real time, *Procedia Environmental Sciences 10*, 2011, vol. 10, pp. 1244—1249.

5. **Maltsev A. V.** Realizatsiia na GPU strui vody i peny v trekhmernykh virtualnykh stsenakh na osnove sistem chastits (GPU implementation of water and foam jets in 3D virtual scenes by using particle systems), *Informatsionnye tekhnologii i vychislitel'nye sistemy*, 2017, no. 1, pp. 40—45 (in Russian).

6. **Reeves W. T.** Particle systems — a technique for modeling a class of fuzzy objects, *In Computer Graphics (Proc. SIGGRAPH)*, 1983, pp. 359—376.

7. **Baciu G., Wong S.-K**. Image-based techniques in a hybrid collision detector, *In IEEE Trans. on Visualization and Computer Graphics*, 2003, vol. 9, pp. 254—271.

8. **Kolb A., Latta L., Rezk-Salama C.** Hardware-based Simulation and Collision Detection for Large Particle Systems, *In Proc. of the ACM SIGGRAPH/ EUROGRAPHICS conference on Graphics hardware*, 2004, pp. 123—131.

9. **Maltsev A. V.** Realizatsiya sistemy chastits v real'nom vremeni na GPU (Real-time particle system realization on GPU), *Programmnye produkty i sistemy*, 2014, no. 4, pp. 57—62 (in Russian).

10. **Framebuffer** Object, OpenGL Overview, available at: https://www.khronos.org/opengl/wiki/Framebuffer_Object

11. **CUDA** C Programming Guide, available at: http://docs.nvidia.com/cuda/cuda-c-programming-guide

12. **Mihayluk M. V., Torgashev M. A.** Sistema vizualizacii "GLView" dlja imitacionno-trenazhernyh kompleksov i sistem virtual'nogo okruzhenija (The system of visualization "GLView" for simulators and virtual environment systems), *Trudy 25-i Mezhdunarodnoj nauchnoj konferencii Grafikon-2015*, 2015, pp. 96—101 (in Russian).

УДК 004.5 DOI: 10.17587/prin.9.311-317

Д. И. Читалов, мл. науч. сотр., e-mail: cdi9@yandex.ru,
С. Т. Калашников, канд. техн. наук, зав. отделом,
Федеральное государственное бюджетное научное учреждение "Южно-Уральский научный центр", Челябинская обл., г. Миасс, Ильменский заповедник

Разработка приложения для подготовки расчетных сеток с помощью утилиты foamyQuadMesh платформы OpenFOAM

Описано приложение с графическим интерфейсом для подготовки параметрических сеток с помощью утилиты foamyQuadMesh, входящей в стандартный дистрибутив программной среды OpenFOAM. Представлены диаграммы структуры приложения и алгоритма работы пользователя с приложением. Приведены средства для создания приложения и его краткое описание. Представлены результаты тестирования приложения на одном из учебных примеров среды OpenFOAM.

Ключевые слова: *графический интерфейс пользователя, OpenFOAM, язык программирования Python, открытое программное обеспечение, foamyQuadMesh, библиотека PyQt, расчетные сетки, модуль Pickle*

Введение

В работе [1] авторами рассмотрены особенности разработки графической оболочки пользователя для взаимодействия с программной средой (ПС) OpenFOAM [2]. Эта консольная ПС предназначена для постановки численных экспериментов в различных областях механики сплошных сред (МСС), прежде всего в гидродинамике, газовой динамике и механике деформируемого твердого тела. Программная среда OpenFOAM является открытым программным решением и эффективной заменой коммерческим продуктам.

Инженеры и исследователи, работающие с ПС OpenFOAM, отмечают один существенный недостаток этого программного комплекса, который затрудняет его использование на отечественных предприятиях. Речь идет об отсутствии встроенной графической оболочки для централизованного управления этапами выполнения численного эксперимента. Над решением вопроса с графической оболочкой для ПС OpenFOAM работают программисты различных зарубежных компаний, представивших собственные проекты графических пользовательских интерфейсов, в частности, Salome [3], Helyx-OS [4], Visual-CFD [5].

Представленные программные решения на практике доказали свою эффективность при решении задач МСС на базе ПС OpenFOAM. Однако для отечественного пользователя их применение связано с определенными сложностями, обусловленными отсутствием русскоязычной сопроводительной документации для технической поддержки, либо ее предоставлением на платной основе. Таким образом, освоение представленных программных средств на российских предприятиях требует существенных временных, а также финансовых издержек.

В работе [1] описан процесс создания оригинальной графической оболочки, предусматривающей централизованное выполнение этапов предобработки (импорт готовой расчетной сетки (РС), определение исходных параметров), решения и постобработки (визуализация результатов решения) применительно к задаче МСС. В представленном интерфейсе предусмотрена загрузка только готовых РС, созданных с помощью стороннего программного обеспечения. Вместе с тем в ПС OpenFOAM предусмотрена возможность работы со встроенными утилитами для генерации РС. В настоящей работе рассмотрены особенности подготовки гексагональных доминирующих РС на основе триангулированных и аналитических поверхностей в 2D-формате и программное приложение с графическим интерфейсом, обеспечивающее централизованную подготовку РС данного типа.

Описываемое приложение, как и графическая оболочка, представленная в работе [1], разработано для специалистов предприятия АО "ГРЦ Макеева", но

может применяться и специалистами других предприятий, использующих ПС OpenFOAM для проведения численных исследования в аэрогидродинамике и механике деформируемого твердого тела. Предлагаемое графическое приложение позволяет упростить подготовку рассматриваемого типа РС и снизить затраты рабочего времени на выполнение этой работы. Благодаря русифицированному интерфейсу и руководству пользователя предоставляется возможность отказаться от дополнительных услуг по техническому сопровождению ПО.

1. Постановка цели и задач исследования

Объектом исследования, результаты которого представлены в настоящей работе, является процесс подготовки в ПС OpenFOAM гексагональных доминирующих параметрических сеток в 2D-формате, состоящих из триангулированных и аналитических поверхностей. В рамках проводимого исследования на основе официальной документации [6] авторами изучены особенности работы соответствующей утилиты ПС OpenFOAM — foamyQuadMesh, а также необходимый набор и структура соответствующих служебных файлов (параметры РС и допустимые типы данных для них).

Целью проводимой исследовательской работы является проектирование программного приложения foamyQuadMesh_generator, имеющего графический пользовательский интерфейс и позволяющего осуществлять подготовку параметрических сеток для ПС OpenFOAM, состоящих из трангулированных и аналитических поверхностей. При этом запуск всех необходимых консольных команд должен осуществляться программными средствами без необходимости запоминания команд пользователем. В соответствии с поставленной целью перед авторами сформулирован следующий комплекс задач.

1. Разработать для приложения основное окно, являющееся центральным звеном программы.

2. Реализовать две языковые версии приложения (русскоязычную и англоязычную), создать диалоговое окно для выбора нужной языковой версии.

3. В структуре основного окна реализовать несколько интерфейсных блоков, а именно — панель инструментов, окно указания исходных параметров сетки, окно вывода результатов, окно вывода служебной информации.

4. В панели инструментов реализовать кнопку открытия диалогового окна выбора директории сохранения РС, кнопку открытия диалогового окна выбора языковой версии интерфейса, кнопки запуска процессов генерации РС и визуализации результатов.

5. Для окна указания исходных параметров сетки реализовать элементы графического интерфейса (текстовые поля, выпадающие списки, флажки, поля ввода цифровых значений и т. д.).

6. Для текстовых полей ввода установить валидаторы, ограничивающие типы вводимых данных, реализовать всплывающие подсказки.

7. В окне вывода служебной информации реализовать механизм генерации служебных сообщений о статусе выполняемых пользователем действий.

8. Обеспечить возможность генерации любого числа РС для отдельного расчетного случая.

9. Обеспечить возможность визуализации результатов численных экспериментов на базе пакета ParaView[7].

2. Анализ объекта исследования

Утилита foamyQuadMesh реализована в ПС OpenFOAM, начиная с версии 2.3.0. Она представляет собой вспомогательное программное средство для формирования гексагональных доминирующих сеточных моделей на основе триангулированных поверхностей. Такие модели определяют моделируемый объект, относящийся к определенному разделу МСС (аэрогидродинамике или механике деформируемого твердого тела), и аналитические поверхности (сфер, цилиндров, плоскостей), задействованные при подготовке РС. Генерируемая сеточная модель при этом определяется в 2D-формате, она обладает высоким уровнем соответствия геометрии рассматриваемой поверхности и высоким уровнем точности фиксирует моделируемые в эксперименте объекты.

Для запуска утилиты foamyQuadMesh пользователю необходимо подготовить следующие служебные файлы.

1. Набор файлов, соответствующих триангулированным (цифровым) поверхностям и описывающих моделируемый объект. Для подготовки таких файлов применяют специализированные редакторы трехмерных моделей.

2. Файл исходных параметров определенной сеточной модели (foamyQuadMeshDict), определяющий свойства расчетной области.

3. Служебный файл surfaceFeatureExtractDict, содержащий параметры извлечения краевой сетки из файлов триангулированных поверхностей и необходимый для формирования основной сетки.

4. Служебный файл extrude2DMeshDict с параметрами экструзии (вытягивания поверхностей) для РС.

Для указания исходных параметров сеточной модели авторами предложен подход создания набора форм для блоков параметров сетки из файла foamyQuadMeshDict, а также включение в этот набор дополнительных форм для файлов surfaceFeatureExtractDict и extrude2DMeshDict. Кроме того, предложено создание формы определения начальных параметров сеточной модели, на базе которой формируется структура служебных файлов с параметрами сетки.

3. Средства разработки

Рассматриваемое в настоящей работе программное приложение, как графическая оболочка, представленная в работе [1], реализовано на базе средств для создания настольных приложений Python [8] и

PyQt [9]. Язык программирования Python 3.4 описывает логику работы программы, а библиотека PyQt4 реализует элементы графического интерфейса.

Связка средств Python и PyQt позволяет создавать полноценные настольные программные продукты для решения достаточно широкого спектра задач в различных предметных областях. Освоение рассматриваемых технологий по сравнению с освоением других средств разработки занимает меньше времени и не требует приобретения лицензии. Благодаря данным средствам разработки приложений повышается производительность программиста и упрощается последующая оптимизация программного кода, а также доработка функциональных возможностей программ.

В рамках представленной в настоящей статье разработки используется среда IDLE. Созданное приложение предназначено для использования на вычислительных устройствах, работающих на ОС Linux. Кроме того, необходимо наличие перечня установленного дополнительного программного обеспечения, включая OpenFOAM, ParaView, интерпретатор Python 3.4, библиотеку PyQt4.

4. Структура программы и алгоритм ее работы

На рис. 1 приведена диаграмма, демонстрирующая архитектуру приложения, разработка которого описывается в настоящей работе. Программа содержит исполняемый файл run.py и набор служебных директорий, содержащих файлы с исходным кодом.

Директория ../windows/ содержит три служебных файла: prj_window.py, lng_window.py и fQMD_window.py. Программный код первого из перечисленных файлов описывает внешний вид и логику работы диалогового окна для выбора директории сохранения РС, код второго файла — внешний вид и логику работы диалогового окна для выбора языковой версии интерфейса, код третьего файла — внешний вид и логику работы окна для указания исходных параметров РС, содержащего набор экранных форм, каждая из которых соответствует определенному блоку параметров РС.

Директория ../functions/ включает служебный файл foamyQuadMeshDict_generation.py, который содержит программный код генерации файла параметров сетки foamyQuadMeshDict.py. В файле msh_functions.py хранится программный код дополнительных служебных функций.

Директория ../threads/ содержит файлы msh_generation.py и msh_visualisation.py, отвечающие за запуск процессов генерации и визуализации РС.

Директория ../matches/ содержит набор файлов-шаблонов, которые вместе со своим содержимым копируются в директорию сохранения РС. К их числу относятся: файл с основными параметрами сеточной модели foamyQuadMeshDict; файл surfaceFeatureExtractDict, содержащий параметры извлечения краевой сетки; файл extrude2DMeshDict с параметрами экструзии для РС.

Директория ../forms/ содержит директорию ../fQMD_forms/ с набором служебных файлов, содержащих программные инструкции, описывающие структуру каждой формы набора для указания параметров РС, а также устанавливающие валидаторы и подсказки для элементов интерфейса.

Директория ../styles/ содержит служебный файл properties_form_style.qss с параметрами стилевого оформления программы.

На рис. 2 приведена диаграмма, описывающая последовательность действий пользователя при работе с программой.

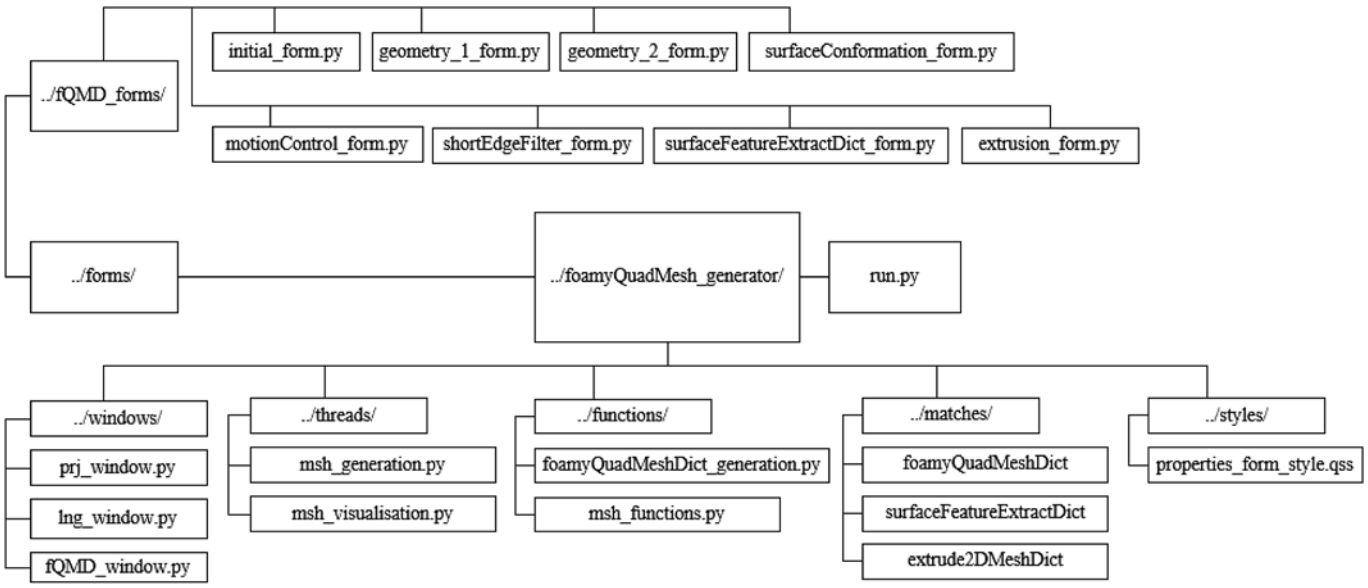


Рис. 1. Диаграмма архитектуры программы foamyQuadMesh_generator

При запуске открывается основное окно программы, в котором пользователь с помощью диалоговых окон может выбрать языковую версию интерфейса и определить режим работы с приложением, а именно — создание новой РС или изменение параметров существующей. Далее для сеточной модели определяются исходные параметры и осуществляется генерация основного (foamyQuadMeshDict) и дополнительных служебных файлов (surfaceFeatureExtractDict и extrude2DMeshDict) с параметрами РС.

Если сохранение исходных параметров РС выполнено без ошибок, то пользователь может перейти к генерации РС, иначе необходимо устранить ошибки. Процесс генерации РС рассматриваемого типа с помощью утилиты foamyQuadMesh может также завершиться с ошибками, что требует изменения исходных данных. Успешная генерация РС позволяет перейти к следующему шагу работы с приложением — к визуализации результатов, на основе которых определяется корректность и точность подготовленной РС.

В процессе указания исходных параметров РС действия пользователя ограничиваются с помощью валидаторов, что позволяет сохранять в служебных файлах только данные допустимых типов. Для упрощения работы пользователя с приложением каждый из элементов управления интерфейса связывается с соответствующей всплывающей подсказкой.

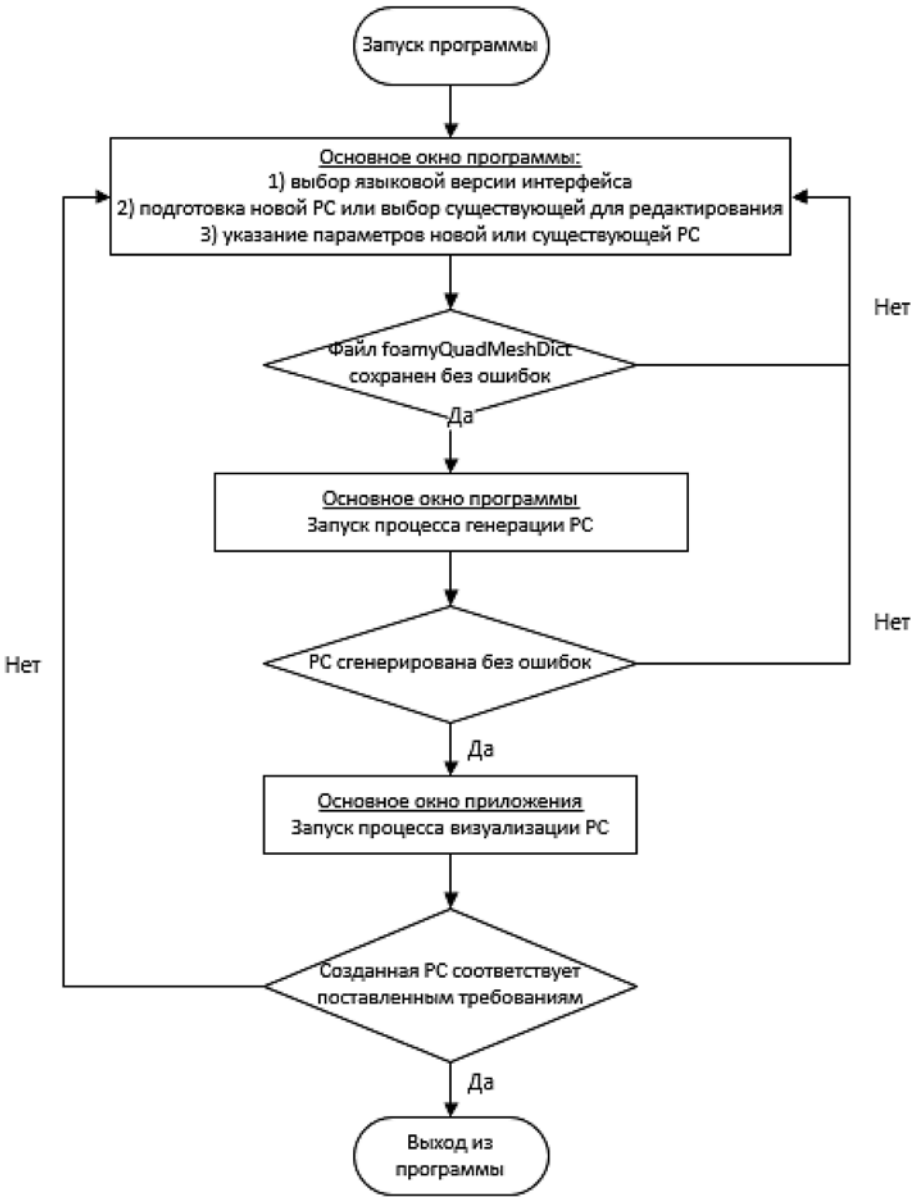


Рис. 2. Действия пользователя при работе с программой foamyQuadMesh_generator

5. Реализация логики работы программы

В процессе реализации логики работы приложения перед авторами возникла необходимость в решении двух задач. Первая из них — обеспечить возможность создания любого числа РС для отдельного расчетного случая, вторая — обеспечить программное выполнение утилит генерации РС.

В рамках решения первой задачи авторами предложен подход использования модуля сериализации данных Pickle [10] языка программирования Python 3.4 для упаковки данных в набор объектов, на основе которого создается содержимое служебных файлов с параметрами РС. К таким файлам относятся: foamyQuadMeshDict, surfaceFeatureExtractDict, extrude2DMeshDict. Для отдельного расчетного случая можно подготовить и сохранить любое количество наборов объектов, каждый набор при этом соответствует определенной РС. При необходимости редактирования существующей РС ее параметры после распаковки выводятся в соответствующие элементы управления окна указания исходных данных сеточной модели.

Каждому блоку данных файла foamyQuadMeshDict соответствует форма. На основе введенных в нее данных создается и сериализуется (консервируется) объект, который помещается в соответствующий набор. Для файлов surfaceFeatureExtractDict и extrude2DMeshDict предусмотрены аналогичные формы и сериализация указанных в них данных.

В рамках решения второй задачи предложен механизм про-

граммного выполнения bash-скриптов [11] с утилитами генерации РС, реализованными в ПС OpenFOAM. Для рассматриваемого типа сеточных моделей осуществляется автоматическая генерация соответствующего файла-сценария (листинг 1) и последующее его исполнение как внешнего Python-процесса (листинг 2).

```
#!/bin/sh
./opt/openfoam4/etc/bashrc
surfaceFeatureExtract
foamyQuadMesh
extrude2DMeshMeshedSurface
exit
```

Листинг 1. Файл сценария генерации РС на основе утилиты foamyQuadMesh

```
msh_run_subprocess = subprocess.Popen
(
[«bash» + «расположение скрипта»],
cwd = «расположение расчетного случая»,
shell = True, stdout = «файл с результатами выполнения процесса»,
stderr = «файл с ошибками выполнения процесса»
)
```

Листинг 2. Код запуска bash-сценария в отдельном процессе

По итогам проведенного исследования авторами предложены следующие подходы к проектированию созданного приложения, которые определяют элементы новизны разработки.

1. Исходные параметры РС из файлов foamyQuadMeshDict, surfaceFeatureExtractDict, extrude2DMeshDict разделить на несколько блоков, для каждого из которых определить соответствующую экранную форму.

2. Использовать форму инициализации сетки, на основе которой определяется базовая структура служебных файлов с параметрами сеточной модели и формируется структура остальных экранных форм.

3. Каждую форму набора определять как отдельный Python-класс, который затем импортировать в общий файл fQMD_window.py, где описывается внешний вид и логика работы окна для указания исходных данных сеточной модели.

4. Параметры каждой из подготовленных сеток сохранять в виде набора "упакованных" с помощью модуля Pickle объектов. Такой подход позволяет для отдельного расчетного случая создавать необходимое число РС. При редактировании существующей сеточной модели выполняется распаковка набора объектов и вывод находящихся в нем параметров сетки в элементы управления экранных форм.

5. Осуществлять программный запуск утилит генерации РС посредством генерации bash-сценариев и их исполнения в рамках дочерних Python-процессов.

6. Внешний вид программы

Разработанное авторами настоящей работы программное приложение для подготовки гексагональных доминирующих параметрических сеток на основе триангулированных и аналитических поверхностей в 2D-формате для ПС OpenFOAM (foamyQuadMesh_generator) относится к классу программного обеспечения с открытым исходным кодом.

Дистрибутив приложения размещен на вебресурсе GitHub [12], предоставляющем услуги хостинга IT-проектов. В дистрибутиве находятся файлы с исходным кодом программы и инструкции пользователя. Для тестирования работы приложения авторами использован один из стандартных учебных примеров (../opt/openfoam4/tutorials/mesh/foamyQuadMesh/jaggedBoundary), входящих в дистрибутив ПС OpenFOAM 4.0. На рис. 3 (см. вторую сторону обложки) представлено изображение основного окна приложения с открытым диалоговым окном выбора режима работы с программой, а на рис. 4 (см. вторую сторону обложки) — после сохранения исходных параметров РС. На рис. 5, 6 (см. третью сторону обложки) в основном окне программы представлены итоги генерации и визуализации РС.

Заключение

В работе, результаты выполнения которой представлены в настоящей статье, исследованы особенности подготовки гексагональных доминирующих сеточных моделей на основе триангулированных и аналитических поверхностей в 2D-формате в программной среде OpenFOAM. Представлены особенности проектирования программного приложения foamyQuadMesh_generator, обеспечивающего автоматизацию подготовки данного типа РС. Исходный код программы находится в свободном доступе для тестирования пользователями и решения практических задач по построению сеточных моделей.

По результатам практического применения приложения foamyQuadMesh_generator специалисты АО "ГРЦ им. Макеева" отмечают снижение затрат рабочего времени на подготовку параметрических сеток и простоту выполнения всех этапов такой подготовки благодаря наличию у программы полноценного русскоязычного графического интерфейса. Программа избавляет пользователей от необходимости изучения особенностей работы ПС OpenFOAM и ее утилит, от запоминания сложных консольных команд, а также обозначения и назначения каждого параметра РС из служебных файлов.

По итогам проведенной работы авторами разработано программное приложение с графическим пользовательским интерфейсом, автоматизирующее подготовку сеточных моделей для ПС OpenFOAM с помощью утилиты foamyQuadMesh. Для разработанной программы решены следующие задачи.

1. Создано основное окно, которое является центральным элементом приложения. Оно предусматривает диалоговое окно выбора одной из двух языковых версий интерфейса — русскоязычной и англоязычной.

2. В основное окно приложения встроены востребованные практикой интерфейсные блоки: панель инструментов; окно указания исходных параметров сетки; окно вывода результатов; окно вывода служебной информации.

3. В приложении предусмотрено окно выбора режима работы, а именно создание новой РС или редактирование существующей.

4. В приложении предусмотрены элементы графического интерфейса, отвечающие за запуск процессов генерации и визуализации РС.

5. В окне для указания исходных параметров сетки реализован набор элементов графического интерфейса для определения параметров сеточных моделей.

6. Для элементов управления интерфейса предусмотрено ограничение типов вводимых данных с помощью валидаторов и реализованы всплывающие подсказки.

7. В окне вывода служебной информации реализован механизм генерации оповещения пользователя о статусе выполненных им в программе действий.

8. Реализована возможность генерации любого числа сеточных моделей для отдельного расчетного случая.

В рамках дальнейших исследований на рассматриваемом в настоящей работе направлении авторы предполагают изучение других типов сеточных моделей и соответствующих утилит, реализованных в ПС OpenFOAM, а также проектирование программных приложений с графическим интерфейсом пользователя для автоматизации подготовки таких сеточных моделей.

Список литературы

1. **Читалов Д. И., Меркулов Е. С., Калашников С. Т.** Разработка графического интерфейса пользователя для программного комплекса OpenFOAM // Программная инженерия. 2016. № 12. С. 568—574.
2. **OpenFOAM.** URL: https://www.openfoam.com/ (дата обращения 01.09.2017).
3. **SALOME.** URL: http://www.salome-platform.org (дата обращения 20.09.2017).
4. **HELYX-OS.** URL: http://engys.com/products/helyx-os (дата обращения 20.09.2017).
5. **Visual-CFD** for OpenFOAM. URL: https://www.esi-group.com/software-solutions/virtual-environment/cfd-multiphysics/visual-cfd-openfoam (дата обращения 20.09.2017).
6. **The open** source CFD toolbox. URL: https://www.openfoam.com/documentation/tutorial-guide/index.php (дата обращения 05.10.2017).
7. **Paraview.** URL: https://www.paraview.org/ (дата обращения 05.10.2017).
8. **Прохоренок Н. А.** Python 3 и PyQt. Разработка приложений. СПб.: БХВ-Петербург, 2012. 704 с.
9. **PyQt4** Reference Guide. URL: http://pyqt.sourceforge.net/Docs/PyQt4/ (дата обращения 10.01.2018).
10. **Сохранение** объектов в файл — Модуль pickle Python. URL: http://python-3.ru/page/module-pickle-python (дата обращения 01.02.2018).
11. **Cooper M.** Искусство программирования на языке сценариев командной оболочки. URL: https://www.opennet.ru/docs/RUS/bash_scripting_guide/ (дата обращения 01.02.2018).
12. **Chitalov D.** foamyQuadMesh_generator. URL: https://github.com/DmitryChitalov/foamyQuadMesh_generator (дата обращения 05.05.2018).

Application Development for Meshes Preparation Using FoamyQuadMesh Utility for the OpenFOAM Toolbox

D. I. Chitalov, cdi9@yandex.ru, **S. T. Kalashnikov,** Federal State Budget Scientific Institution "South Ural Scientific Center", Chelyabinsk region, Miass, Ilmen reserve, 456317, Russian Federation

Corresponding author:

Chitalov Dmitry I., Junior Researcher, Federal State Budget Scientific Institution "South Ural Scientific Center", Chelyabinsk region, Miass, Ilmen reserve, 456317, Russian Federation,
E-mail: cdi9@yandex.ru

Received on May 07, 2018
Accepted on May 17, 2018

This article is devoted to the study of the features of preparation of computational meshes (hereinafter — CM) with the help of the utility foamyQuadMesh, which is included in the standard distribution package of OpenFOAM software environment (hereinafter — SE). The aim of the study is the development of software application (foamyQuadMesh_generator) with a graphical interface for the preparation of this type of CM. The shortcomings of existing software solutions with a graphical interface for working with computational meshes for OpenFOAM are given. The urgency of designing of a new graphical shell has been formulated. As part of the implementation of the aim, the authors have analyzed the process of CM preparation on the basis of the utility foamyQuadMesh and have proposed a list of development tools. The diagrams reflecting the structure of the created application and the algorithm of the user's work with the application are presented. Features of the implementation of the application logic are presented, the novelty of the development is determined. The list of software products required to use the foamyQuadMesh_generator application is given. The result of the work carried out by the authors is the creation of an original software solution with a graphical interface for the preparation of CM using the foamyQuadMesh utility of OpenFOAM SE. The article shows the results of testing the program on one of the training examples included in the standard OpenFOAM SE distribution package, a link to the GitHub service is provided, where the application is freely available. The practical significance of the development and the prospects for further research in this area have been determined.

Keywords: *graphical user interface, OpenFOAM, Python programming language, open source software, foamyQuadMesh, PyQt library, calculated meshes, Pickle, bash-scripting, ParaView*

For citation:

Chitalov D. I., Kalashnikov S. T. Application Development for Meshes Preparation Using FoamyQuadMesh Utility for the OpenFOAM Toolbox, *Programmnaya Ingeneria*, 2018, vol. 9, no. 7, pp. 311—317.

DOI: 10.17587/prin.9.311-317

References

1. **Chitalov D. I., Merkulov E. S., Kalashnikov S. T.** Razrabotka graficheskogo interfeysa pol'zovatelya dlya programmnogo kompleksa OpenFOAM (Development of a graphical user interface for the OpenFOAM software package), *Programmnaya Ingeneria*, 2016, no. 12, pp. 568-574 (in Russian).

2. **OpenFOAM.** The open source CFD toolbox, available at: https://www.openfoam.com/ (date of access 01.09.2017).

3. **Salome.** The Open Source Integration Platform for Numerical Simulation, available at: http://www.salome-platform.org (date of access 20.09.2017).

4. **HELYX-OS**. The market leading open-source GUI for OpenFOAM, available at: http://engys.com/products/helyx-os (date of access 20.09.2017).

5. **Visual-CFD** for OpenFOAM, available at: https://www.esi-group.com/software-solutions/virtual-environment/cfd-multiphysics/visual-cfd-openfoam (date of access 20.09.2017).

6. **OpenFOAM.** Tutorial Guide, available at: https://www.openfoam.com/documentation/tutorial-guide/index.php (date of access 05.10.2017).

7. **ParaView,** available at: https://www.paraview.org/ (date of access 05.10.2017).

8. **Prohorenok N. A.** *Python 3 i PyQt. Razrabotka prilojeniy* (Python 3 and PyQt. Application Development), St. Petersburg, BHV-Petersburg, 2012, 704 p. (in Russian).

9. **PyQt4** Reference Guide, available at: http://pyqt.sourceforge.net/Docs/PyQt4/ (date of access 10.01.2018).

10. **Serializatsiya** ob"ektov Python (Serialization of Python objects), available at: http://python-3.ru/page/module-pickle-python (date of access 01.01.2018) (in Russian).

11. **Cooper M.** Iskusstvo napisaniya Bash-skriptov (The art of writing Bash scripts), available at: https://www.opennet.ru/docs/RUS/bash_scripting_guide/ (date of access 01.02.2018) (in Russian).

12. **Chitalov D.** foamyQuadMesh_generator, available at: https://github.com/DmitryChitalov/ foamyQuadMesh_generator (date of access 15.03.2018).

УДК 004.75 DOI: 10.17587/prin.9.318-331

С. Е. Попов, канд. техн. наук, ст. науч. сотр., e-mail: popov@ict.sbras.ru,
Р. Ю. Замараев, канд. техн. наук, ст. науч. сотр., e-mail: zamaraev@ict.sbras.ru,
И. Е. Харлампенков, канд. техн. наук, науч. сотр., e-mail: kharlampenkov@ict.sbras.ru,
Федеральное государственное бюджетное учреждение науки Институт вычислительных технологий Сибирского отделения Российской академии наук, г. Новосибирск

Веб-сервис классификации сейсмических событий на базе системы распределенных вычислений Apache Spark[1]

Описаны ключевые моменты процесса разработки сервиса для быстрой автоматической классификации сейсмических сигналов на основе диагностических шаблонов. Представлены программные решения для предварительной обработки сигнала и алгоритмизации параллельных вычислений на математической модели выработки конечных заключений с использованием базы рейтингового голосования. Показаны возможности интеграции таких решений с системой распределенных вычислений Apache Spark. Проведены тесты производительности алгоритма классификации для набора суточных сигналов в различных программных средах. Показано, что запуск алгоритма классификации в контексте массивно-параллельного исполнения обеспечивает прирост производительности в несколько десятков раз. Сервис разработан с применением библиотек React и Redux. В качестве среды выполнения использована платформа NodeJS.

Ключевые слова: *веб-сервис, распределенные вычисления, Apache Spark, классификация сейсмических событий*

Введение

Региональный мониторинг и анализ региональной геодинамической ситуации характеризуются сложностью решаемых на этом направлении задач. Причина в том, что наряду с мощными возмущениями из известных очаговых зон приходится анализировать и классифицировать разнородный поток событий, среди которых промышленные взрывы различной мощности и глубины заложения, региональные и местные сейсмические события. В горнопромышленных регионах России (Кемеровской области, Красноярском крае и др.) функционирует большое число предприятий, регулярно проводящих массивные взрывные работы. В общей сложности за год может регистрироваться более 2,5 тыс. сейсмических событий со схожей магнитудой (1,5...2,5 балла), характерной как для региональных землетрясений, так и для типичных (по технологии и мощности) взрывов, например, на угольных разрезах или глубоких карьерах. Необходимо также учитывать разветвленную сеть сейсмостанций (например, по Красноярскому краю — 10, по Кемеровской области — 7), записывающих непрерывный поток данных с частотой в среднем 100 отсчетов (замеров) каждые 10 мс минимум по трем каналам измерений в каждой. Таким образом накапливаются огромные массивы информации, где только для одной станции недельный пул суточных записей составляет около 150 Мбайт (≈ 174 млн отсчетов), а с нескольких станций — более 1 Гбайта (1 млрд отсчетов). Учитывая данный фактор, процесс детектирования и классификации различных возмущений в наборе даже суточных записей с 4—6 станций требует значительных вычислительных ресурсов и затрат времени. Это же утверждение косвенно подтверждается анализом большинства исследований в области обработки сейсмических сигналов [1—17]. В этих работах описан ряд работоспособных подходов, которые опираются на различные комбинации процедур фильтрации, спектрального и вейвлет-анализа, и на автокорреляционные методы, интегрируемые с аппаратом искусственных нейронных сетей и кластерным анализом. В большинстве таких исследований все апробации алгоритмов на реальных сигналах осуществляются на малых таймфреймах (временных промежутках или отрезках) не более 60...80 с. Рассматриваются отрезки сигнала с априори достоверной информацией о присутствии на них существенных (детектируемых) воз-

[1] Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 18-07-00013 А.

мущений. Время работы представленных алгоритмов на полном сигнале (суточной записи) не приводится. Отдельно следует отметить работу [16], в которой авторы используют алгоритм *Fingerprint And Similarity Thresholding* (FAST) для детектирования природных землетрясений и проводят его тестирование на реальных сигналах недельного пула записей. В этой работе показано время исполнения около 1 ч 36 мин против 9 дней автокорреляционного метода (данные взяты с одной станции). Таким образом, при анализе даже 2—3-недельных таймфреймов с нескольких сейсмопостов время работы может увеличиваться до одного дня, а с использованием других методов — и до месяца.

Рассматривая проблему производительности алгоритмов с позиций повышения эффективности информационно-аналитического обеспечения процессов регионального мониторинга, можно обозначить основные требования: стабильное поступление массивов актуальных сейсмических данных; их оперативная обработка; анализ и выработка экспертных заключений с использованием быстрых алгоритмов. В мировой практике насчитывается большое число программных решений, реализующих различные функции обработки и анализа сейсмической информации. Среди средств сбора, хранения и доступа к сейсмической информации крупнейшим является IRIS DMC [18]. Среди средств обработки и анализа сейсмических данных можно выделить программное обеспечение для обнаружения волновых возмущений сигнала [19—21], для интерактивного моделирования сейсмического волнового поля и проверки геологических гипотез при классификации сейсмических данных [22], а также программные комплексы для решения обратной геофизической задачи, построенные на базе открытых GRID-систем [23—25].

Однако функциональные возможности существующих программных решений не поддерживают классификацию сейсмических событий, основанную на обработке множества реальных суточных записей, полученных с разных станций наблюдений. Обработка данных ведется в ручном режиме с последовательной загрузкой файлов и выделением мелких таймфреймов интересующего события. Интегрированные в такие программные средства алгоритмы классификации не поддерживают запуск в параллельном режиме обработки полного временного отрезка сигнала. Все это приводит к существенной деградации производительности. В большинстве случаев программные решения представляют собой статические приложения, ориентированные на работу со специализированной аппаратной частью. Данный факт значительно сужает возможности анализа, поиска и подтверждения характера сейсмопроявлений на основе информации, получаемой одновременно из различных источников их фиксации.

Учитывая изложенное выше, возникает актуальная задача разработки программного обеспечения для классификации сейсмических событий, поддерживающего высокопроизводительную обработку больших массивов данных и открытый доступ к ним на базе веб-технологий.

Цель и задачи исследования

Целью исследования является разработка веб-сервиса для классификации сейсмических событий, полученных в виде сейсмосигналов с нескольких станций наблюдения.

Задачи, которые предполагается решать в рамках такого исследования, заключаются в следующем.

1. Разработать оригинальный алгоритм, поддерживающий возможность автоматического поиска, детектирования и классификации всех возмущений сейсмического сигнала на суточном временном интервале.

2. Адаптировать представленный алгоритм для запуска в системе массово-параллельного исполнения заданий в целях уменьшения времени работы (не более 40...50 с на один суточный таймфрейм).

3. Предоставить механизмы открытого доступа к возможностям классификации сейсмических событий на базе разработанного алгоритма в виде открытого веб-сервиса в сети Интернет.

Исходные данные и шаблоны

Источником сейсмических данных служит региональная сеть из восьми станций с международными кодами ASR1, ELT, BRCR, KEM, LUZB, NVS, SALR, TASR[2]. Сигналы поступают в формате miniSEED, данные предоставляются по трем каналам, например, EHE, EHN, EHZ.

Суточная запись с каждого канала содержит около 8,5 млн отсчетов (замеров с датчиков) и время каждого отсчета, т. е. (24 ч) × (3600 с в часе) × (100 отсчетов в секунде (*sample rate*)). Начальные записи в каналах могут быть сдвинуты относительно начала суток (00:00:00). Для их синхронизации выбирают самый поздний по времени начальный отсчет (по максимальному времени от начала) и от этого времени извлекают все значения каждого канала. Далее выбирают минимальную длину из получившихся массивов (по минимальному времени от конца сигнала), оставшиеся массивы "обрезают справа" до этой длины. Таким образом, получают матрицу со значениями типа Double

$$\mathbf{CH}=\left\{ch_{ij},\ i\in[0,\ L_{ch}]\in Z;\ j\in[0,\ 2]\in Z\right\}, \qquad (1)$$

где L_{ch} — длина массива данных для каждого синхронизированного канала (в среднем 8,3...8,5 элементов); j — номер канала.

Типовые шаблоны построены на примере сигналов с одной станции BRCR. Использовались сейсмограммы (часть суточного сигнала) 35 достоверных

[2] Международные коды сейсмических станций (International Registry of Seismograph Stations (IR), http://www.isc.ac.uk/registries/).

промышленных взрывов и 19 региональных землетрясений с магнитудами от 1,8 до 2,4 балла. Исследования авторов показали, что для расстояний 220...430 км от станции до активных сейсмических зон характерное время прохождения и эффективного затухания сейсмического возмущения находится в диапазоне 50...75 с. Таким образом, длина расчетного окна в среднем составляет $m = 6145$ отсчетов или 61,45 с при частоте дискретизации сигнала (*sample rate*) 100 Гц.

Для каждой сейсмограммы были получены значения характеристической функции — массивы-шаблоны длиной $N = 6145$ (см. разд. **Алгоритм классификации**). Из них для совокупностей взрывов и землетрясений были выделены по три шаблона: средний и две его границы: для взрывов Blast (Blast $\pm S$), для землетрясений EarthQuake (EarthQuake $\pm S$). С использованием выражения $f(t) = A(t/Tn^2)^n \exp(n - t/Tn) + \varepsilon(t)$, где A, T, n — числовые параметры ($A = 1$, $T = 3800$, $n = 44$, подбираются индивидуально для станций в зависимости от расстояния до очаговых зон, $\varepsilon(t)$ — вектор случайных чисел, соответствует статистическому распределению модели сигнала "Белый шум"), добавлены абстрактные шаблоны. Они показывают последовательное прохождение сейсмического возмущения со сдвигом 100 отсчетов (1 с) через расчетное окно. Таким образом, получаем шаблоны со следующими названиями: на входе WaveFront-I, WaveFront-II и WaveFront-III; на выходе WaveRear-I, WaveRear-II и WaveRear-III; в середине окна со смещениями влево и вправо WaveMiddle, WaveLeft и WaveRight соответственно. Получае также шаблон равномерного распределения или белого шума WhiteNoise, отвечающий за имитацию сейсмического фона и возможные случайные возмущения. Всего получено 16 шаблонов — матрица со значениями типа Double: $C_{ij}^t, i \in [0,\ 6144] \in Z;\ \ j \in [0,\ 15] \in Z$.

Алгоритм классификации

Алгоритм классификации является оригинальной разработкой авторов [27]. В текущей версии алгоритма классификация на базе корреляционной функции расширена использованием функций статистических расстояний (дистанций). В связи с последним обстоятельством значительно улучшилась разрешающая способность алгоритма (число верных классификаций), подтвержденная протоколами и оперативными донесениями Кемеровской службы мониторинга геофизической обстановки.

Алгоритм позволяет анализировать полные трехкомонентные суточные сигналы. На вход алгоритму подается сформированная матрица (1): $\mathbf{CH} = \{ch_{ij},\ i \in [0,\ L_{ch}] \in Z;\ j \in [0,\ 2] \in Z\}$. Задается скользящее окно размером $m = 6145$ отсчетов, с шагом сдвига $step = 100$ отсчетов. На каждом шаге формируется сейсмограмма в виде матрицы $\mathbf{X} = \{x_{ij}, i \in [0,\ m] \in Z;\ j \in [0,\ 2] \in Z\}$, содержащая часть сигнала **CH** (1). Алгоритм определяет (классифицирует) тип сейсмограммы согласно шаблонам следующим образом.

Шаг 1. Компоненты матрицы $\mathbf{X} = \{x_{ij}\}$ заменяем квадратами размахов $sw_{i,j}$, что обеспечивает неотрицательность значений для дальнейших вычислений:

$$sw_{i,j} = \left(x_{i,j} - x_{i+1,j}\right)^2,\ i \in [0,\ m-1];\ j \in [0,\ 2]. \qquad (2)$$

Шаг 2. Вычисляем матрицу весов:

$$q_{i,j} = \frac{sw_{i,j}}{\sum_{i=0}^{m-1} sw_{i,j}},\ i \in [0, m-1];\ j \in [0,\ 2], \qquad (3)$$

и затем матрицу энтропий:

$$E_{i,j} = -q_{i,j} \ln\left(q_{i,j}\right),\ i \in [0,\ m-1];\ j \in [0,\ 2]. \qquad (4)$$

Энтропийная модель дискретного сигнала (3)—(4) обладает свойствами Шенноновской энтропии выборки [28]. Модель обеспечивает аддитивность элементов, относящихся к одному сигналу, и, главное, аддитивность элементов из различных сигналов в синхронных отсчетах.

Шаг 3. Вычисляем вектор обобщенной информации по трем каналам измерений:

$$H_i = E_{i,0} + E_{i,1} + E_{i,2},\ i \in [0,\ m-1]. \qquad (5)$$

Шаг 4. Строим характеристическую функцию в расчетном окне. Данный процесс называется аккумулированием сигнала:

$$C_i^s = \sum_{l=0}^{i} H_l,\ i \in [0,\ m-1]. \qquad (6)$$

За счет аккумулирования три компоненты сигналов приводятся к одномерной стационарной форме [29]. Стационарность модели (6) обеспечивает хорошую аппроксимацию по осредненным (сглаженным) данным (шаблонам с известными характеристиками).

Шаг 5. Добавляем справа к матрице C_{ij}^t, $i \in [0, 6144] \in Z;\ \ j \in [0, 15] \in Z$ вектор-столбец C^s, получаем матрицу $C_{ij}, i \in [0, m-1];\ \ j \in [0,\ n],\ n = 16$.

Согласно выражению (6), все шаблоны находятся в одном метрическом пространстве. Полагая шаблоны признаками, а отсчеты объектами (независимыми наблюдениями), можем дополнить их набор выборочной характеристической функцией и вычислить аналог диагностической матрицы по Байесу путем стандартизации в объектах:

$$S_{i,j} = \left(C_{i,j} - \mu_i\right) / \sigma_i, \qquad (7)$$

где $\mu_i = \frac{1}{n}\sum_{j=1}^{n} C_{i,j}$ и $\sigma_i = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(C_{i,j} - \mu_i\right)^2}$.

Теперь в матрице (7) можно оценивать подобие характеристической функции (6) каждому шаблону из набора как расстояние между двумя признаками (одно-

мерными векторами). Для этого фиксируем $j = 16$ и для каждого $S_{i,j}$, $j \in [0,15]$ формируем пару с $S_{i,16}$, получаем 16 пар.

Шаг 6. Для каждой пары рассчитываем значения расстояний (табл. 1).

Получаем матрицу расстояний

$$\mathbf{D} = \left\{ D_{k,j},\ k \in [0, 11],\ j \in [0, 15] \right\}, \tag{8}$$

где весовой коэффициент $w_i = \begin{cases} 1, i \in [0, 2(m-1)/3] \\ 0, \text{ в остальных случаях} \end{cases}$.

Априорных суждений о преимуществах тех или иных дистанций не существует, поэтому в текущей версии алгоритма используются все, пригодные для номинальных признаков с различными вариациями [30].

Теперь можно реализовать простейшую систему голосования, в которой каждая дистанция имеет один голос. Каждый голос отдается шаблону с минимальной дистанцией (8) до выборочной характеристической функции. Простое суммирование голосов у каждого шаблона определяет его простой рейтинг.

Шаг 7. Преобразуем матрицу **D** следующим образом: для каждого $k \in [0, 11]$

$$D_{k,j} = \begin{cases} 1, & D_{k,j} = \min(D_k), \\ 0, & \text{в остальных случаях.} \end{cases} \tag{9}$$

Таблица 1

Статистические расстояния

Расстояние	Формула для расчета	Формула для расчета с весовым коэффициентом
Bray-Curtis	$D_{0,j} = \dfrac{\sum_{i=0}^{m-1} \lvert S_{i,16} - S_{i,j} \rvert}{\sum_{i=0}^{m-1} \lvert S_{i,16} + S_{i,j} \rvert}$	—
Canberra	$D_{1,j} = \sum_{i=0}^{m-1} \dfrac{\lvert S_{i,16} - S_{i,j} \rvert}{\lvert S_{i,16} \rvert + \lvert S_{i,j} \rvert}$	$D_{2,j} = \sum_{i=0}^{2(m-1)/3} w_i \dfrac{\lvert S_{i,16} - S_{i,j} \rvert}{\lvert S_{i,16} \rvert + \lvert S_{i,j} \rvert}$
CityBlock	$D_{3,j} = \sum_{i=0}^{m-1} \lvert S_{i,16} - S_{i,j} \rvert$	—
Корреляция (нормированная)	$D_{4,j} = 1 - cov/(\sigma_1 \sigma_2),\ cov = \dfrac{\sum_{i=0}^{m-1} S_{i,16} S_{i,j}}{m-1} - \dfrac{\sum_{i=0}^{m-1} S_{i,16} \sum_{i=0}^{m-1} S_{i,j}}{(m-1)^2},$ $\sigma_1 = \sqrt{\dfrac{\sum_{i=0}^{m-1} S_{i,16}^2}{m-1} - \dfrac{\left(\sum_{i=0}^{m-1} S_{i,16}\right)^2}{(m-1)^2}},\ \sigma_2 = \sqrt{\dfrac{\sum_{i=0}^{n-1} S_{i,j}^2}{m-1} - \dfrac{\left(\sum_{i=0}^{m-1} S_{i,j}\right)^2}{(m-1)^2}}$	—
Евклидово	$D_{5,j} = \lVert S_{16} - S_j \rVert_2$	$D_{6,j} = \sqrt{\sum_{i=0}^{2(m-1)/3} w_i \left(S_{i,16} - S_{i,j}\right)^2}$
Евклидово второй степени	$D_{7,j} = \lVert S_{16} - S_j \rVert$	$D_{8,j} = \sum_{i=0}^{2(m-1)/3} w_i \left(S_{i,16} - S_{i,j}\right)^2$
Минковского третьей степени	$D_{9,j} = \lVert S_{16} - S_j \rVert_3$	$D_{10,j} = \sqrt[3]{\sum_{i=0}^{2(m-1)/3} w_i \left(S_{i,16} - S_{i,j}\right)^3}$
Косинусное	$D_{11,j} = 1 - \dfrac{S_{16} S_j}{\lVert S_{16} \rVert_2 \lVert S_j \rVert_2}$	—
Примечание. Прочерк означает отсутствие аналогичного расстояния с весовым коэффициентом.		

Шаг 8. Рассчитываем рейтинги

$$R_j = \sum_{k=0}^{11} D_{k,j} \qquad (10)$$

и находим максимум R_j.

Шаг 9. Формируем заключение классификации по следующей схеме, назовем ее рейтинговым голосованием:

а) если не существует единственного максимума, то заключение **"не определено (undefined)"**.

б) если единственный максимум больше 10, то заключение **"строгое (strictly)"** соответствие шаблону.

в) если единственный максимум больше 8, но меньше 11, то заключение **"нестрогое (not strictly)"** соответствие шаблону.

г) если единственный максимум меньше 9, то заключение **"возможное (perhaps)"** соответствие шаблону.

Таким образом, при сдвиге окна от начала сигнала в конец алгоритмом детектируются и идентифицируются согласно шаблону любые значимые возмущения суточного таймфрейма.

Оптимизация алгоритма

Для уменьшения времени работы программной реализации алгоритма и его адаптации к запуску в среде массово-параллельного исполнения заданий были проведены перечисленные далее действия (итерации).

1. Предварительный расчет элементов $sw_{i,j}$ (2) для трех каналов суточной записи. Получаем матрицу $swO_{ij},\ i \in [0,\ L_{ch} - 1]$. Заранее рассчитываем $\sum_{i=1}^{m-1} sw_{i,j}$ (2) для каждого шага сдвига. Получаем массив

$$sw\_sums_{j,s} = sw\_sum_{j,s-1} \pm \sum_{l=0}^{100} swO_{100s \pm l,j},\ \ s \in \left[1,\ \frac{L_{ch}}{100}\right],$$

где $sw\_sum_{j,0} = \sum_{i=0}^{m-1} sw_{i,j}$.

Массивы *swO* и *sw_sums* являются общедоступными константами для всех заданий в среде Apache Spark и передаются с помощью специального объекта **Broadcast**. Данная оптимизации позволит существенно сократить время расчета формул (2) и (3), так как на каждом сдвиге окна вычисляются суммы только предыдущих и последующих 100 элементов матрицы *sw*.

2. Для матрицы C^t расчет суммы $CSum_i = \sum_{j=1}^{n} C^t_{i,j}$. $CSum_i$ объявляем общедоступной константой (**Broadcast**). Тогда на каждом шаге выражение $\mu_i = \frac{1}{n}\sum_{j=1}^{n} C_{i,j}$ в формуле (7) можно заменить на $\mu_i = \frac{1}{n}\left(C_{i,16} + CSum_i\right)$, что также позволяет сократить число операций с суммой элементов.

3. Используемые алгоритмы нахождения расстояний согласно [14] в своих расчетах содержат повторяющиеся выражения. Например, расстояние Bray-Curtis будет содержать выражение $S_{i,16} - S_{i,j}$, которое также есть в Canberra, или $\sum\left|S_{i,16} - S_{i,j}\right|,\ \ j \in [0,\ 15]$ в City Block. Расчет корреляции через ковариацию позволит получить выражения $\sum {S_{i,16}}^2,\ \ \sum S^2_{i,j}$, $\sum S_{i,16} S_{i,j}$, которые присутствуют в расчете евклидова и косинусного расстояний. Учитывая, что весовой коэффициент принимает значение 1 при двух третьих от общего числа итераций, а остальные значения равны 0, то при расчете сумм в соответствующих расстояниях без весового коэффициента необходимо зафиксировать значение суммы на номере итерации, равном $2(m - 1)/3$ и использовать это значение при получении значения расстояния с весовым коэффициентом. То есть если евклидово расстояние есть $\sum S^2_{i,16} + \sum S^2_{i,j} - 2\sum S_{i,16} S_{i,j},\ \ i \in [0,\ m-1]$, то евклидово расстояние с весовым коэффициентом будет вычисляться так же, только для $i \in [0,\ 2(m-1)/3]$.

4. Анализ алгоритма классификации (шаги 1—9) показал независимость расчетов на каждой итерации сдвига расчетного окна. Этот факт означает, что заключение классификации получается как единичное абстрактное значение одного из признаков (см. шаг 9 из разд. **Алгоритм классификации**). Таким образом, возможно разделить всю суточную запись длиной L_{ch} на части (**partitions**) и вычислять модель (2)—(10) параллельно. Назовем этот этап **Map**. Затем после завершения всех заданий Map будем запускать процесс объединения полученных заключений, сортируя их по времени в начальном сигнале (получение карты классификаций). Данный этап назовем **Reduce**. Таким образом возможно организовать вычисления по классической схеме **MapReduce**.

Такого рода оптимизации позволяют значительно сократить время работы алгоритма, так как при каждом сдвиге приходится выполнять одни и те же вычислительные операции на одних и тех же наборах данных по несколько раз. Учитывая число шагов (в среднем $\frac{L_{ch}}{step}$) и размер окна ($m - 1 = 6144$ отсчета), потери производительности можно считать существенными.

Технологический стек

Вычислительное ядро сервиса (**BACKEND**), который предлагается в настоящей работе, реализовано на базе Apache Spark API (Java), менеджера ресурсов Apache YARN и сервиса удаленного запуска заданий Apache Livy. Система обработки HTTP-запросов пользователя (**MIDDLEWARE**) функционирует под платформы управлением NodeJS, Chart.js (ECMAScript 6 (ES6)). Графический интерфейс сервиса построен с применением библиотек React + + Redux, Plot.ly, Semantic UI (ES6) под управлением сервера Nginx (**FRONTEND**). Операции ввода/вывода

BACKEND- и **MIDDLEWARE-**компонентов реализуются на базе распределенной файловой системы HDFS (ОС Ubuntu 16.04).

Описание веб-сервиса

Разработанный веб-сервис логически разделен на три составляющие: **BACKEND**, **MIDDLEWARE**, **FRONTEND**.

BACKEND-компонент представлен java-классами, реализующими непосредственно расчетную часть алгоритма классификации, и дополнительными классами, имплементирующими методы предварительной обработки сейсмического сигнала и работы с объектами Apache Spark API (рис.1).

ClassificationProcessor — это основной класс, который отвечает за запуск процесса классификации (рис. 2). Он содержит подкласс **classify,** наследующий объект **Function** Spark API для передачи его в функцию **map**. Подкласс **classify** реализует программный алгоритм классификации (классы **SignalProcessor** и **DistanceClassifier**), адаптированный для работы в распределенном режиме на узлах кластера. Класс **ClassificationProcessor** обеспечивает настройку среды исполнения (**Executor**) заданий посредством объекта **SparkContext**. В этом классе реализована процедура размещения неизменяемых объектов-констант (**Broadcast**), содержащих предварительно рассчитанные данные (см. разд. **Оптимизация алгоритма**) в классах **TemplateProcessor** и **MiniSEEDProcessor**. Данные константы доступны со всех узлов кластера в общей памяти текущего контекстного объекта.

Данные шаблонов хранятся в HDFS в CSV-файле. Класс **TemplateProcessor** поддерживает методы чтения, обработки данных CSV и построение массива C_{ij}^{t} и $\sum_{j=1}^{n} C_{i,j}$ для каждого *i*-го отсчета, для добавления в пул общедоступных констант (**Broadcast**, см. разд. **Оптимизация алгоритма**).

Класс **MiniSEEDProcessor** содержит методы работы с файлами miniSEED-формата с помощью библиотеки iris-WS.jar. Она позволяет открывать, декодировать и считывать данные из файлов каналов сейсмических записей. **MiniSEEDProcessor** реализует процедуру синхронизации каналов по времени и формирования константы **Broadcast** для матрицы **CH** (1). Также в данном классе рассчитываются вспомогательные данные: длина сигнала и метаинформация по каждому каналу (название, частота дискретизации, начальное/конечное время записи).

Результатом работы метода **classify** является JSON-файл (рис. 3), хранящийся в HDFS, где каждая часть (выполненное задание) идентифицируется ключом **partition**, содержащим свойства того или иного заключения карты классификаций.

MIDDLEWARE-компонент (рис. 4) реализован на базе объектов языка ES6. Этот компонент имплементирует методы программного каркаса библиотеки NodeJS API. Он функционирует как прокси-уровень между **FRONTEND** и **BACKEND**, выступает в качестве обработчика пользовательских HTTP-запросов для вызовов их методов.

Для запуска задания на стороне **BACKEND-**компонента **MIDDLEWARE**-компонент использует объекты **router** и **request,** предоставляемые стандартной библиотекой NodeJS Express API. Данные объекты перенаправляют POST-запросы со стороны **FRONTEND-**компонента сервису Apache Livy (URL: http://livy-server:8998/batches). POST-запрос содержит настройки параметров среды Apache Spark, менеджера ресурсов Apache YARN, параметров расчетного модуля и управляющей программы-драйвера (рис. 5).

Ответом на POST-запрос является JSON-объект, состоящий из частей (partition) по количеству параллельно выполненных задач. Для их объединения используется метод **getResultAsJSON** (рис. 6).

Для простоты взаимодействия объектов **MIDDLEWARE-**компонента с распределенной файловой системой HDFS последняя была смонтирова-

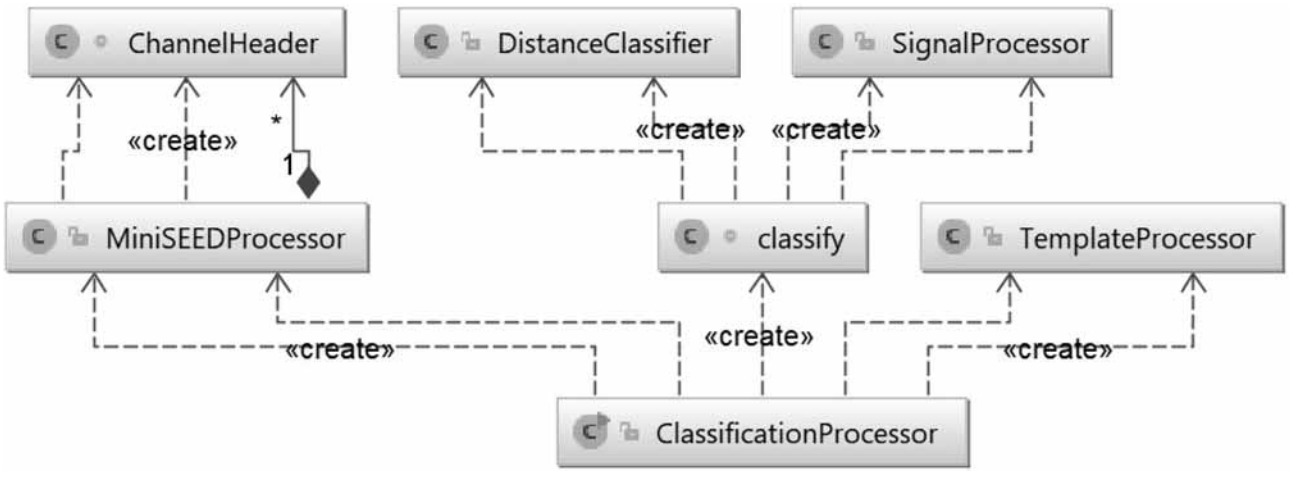


Рис. 1. Диаграмма объектов BACKEND-компонента

```
// Обработчик шаблона
TemplateProcessor templateProcessor = new TemplateProcessor();
templateProcessor.process(templatesFile);
// Обработчик каналов
MiniSEEDProcessor miniSEEDProcessor = new MiniSEEDProcessor(...);
miniSEEDProcessor.process(chFiles[0], chFiles[1], chFiles[2], true);
// Настройка контекста запуска
SparkConf sparkConf = new SparkConf();
JavaSparkContext sc = new JavaSparkContext(sparkConf);
// Размещение общедоступных данных
Broadcast<double[][]> CBroadcast = sc.broadcast(templateProcessor.C);
...
// Запуск задания
List<String> classificationMapParts = sc.parallelize(IntStream.range(0,partitionCount)
        .boxed().collect(Collectors.toList()), partitionCount)
        .map(new classify(...,..., CBroadcast,...
       )).collect();
...
// Имплементация объекта Function для метода map (параллельно для каждого задания)
class classify implements Function<Integer, String> {
...
public classify(...,...,Broadcast<double[][]> CBroadcast,...) {
        this.CBroadcast = CBroadcast;}

    @Override
    public String call(Integer core) {

        SignalProcessor signalProcessor = new SignalProcessor(...);
        DistanceClassifier distanceClassifier = new DistanceClassifier(...);
        ...
        // Доступ к общим данным внутри задания SparkContext
        double[][] C = CBroadcast.value();
        ...
        ...
        // Расчет алгоритма классификации
        double[][] D = signalProcessor.process(ch1RawData, ch2RawData, ch3RawData, C, CSums);
        conclusionResult = distanceClassifier.process(D);
        ...
        return conclusionResult // Результат в виде JSON-строки
...
```

Рис. 2. Фрагмент кода настройки контекста Spark и запуска расчетного задания класса ClassificationProcessor

на как обычная директория в операционной системе (/mnt/hdfs/...) с использованием программного интерфейса FUSE (*Filesystem in Userspace*). Это позволило осуществлять операции ввода/вывода на базе методов объекта **fs (NodeJS FileSystem)** и обрабатывать соответствующие GET/POST-запросы со стороны **FRONTEND** непосредственно в **MIDDLEWARE-**компоненте (рис. 7).

FRONTEND-компонент (рис. 8) представлен программными объектами языка ES6, наследующими и расширяющими функциональные возможности типизированных классов библиотеки React. Эти классы отвечают за графический интерфейс и взаимодействие с пользователями веб-сервиса.

Каждый элемент графического интерфейса поддерживает модель управления состоянием приложения (Redux) в виде JSON-объектов, содержащих пары ключ/значение свойств данного элемента. Изменение состояния осуществляется за счет функций-действий (action), привязанных к тому или иному объекту. **FRONTEND-**компонент имеет древовидную структуру объектов (родительский/дочерний элемент). Состояние каждого объекта может быть изменено на любом из уровней. Файлы index.js (рис. 8) включают код размещения (функция **render()** на рис. 9) js-элементов более низкого уровня согласно древовидной структуре, вплоть до последнего дочернего элемента. Такая структура позволяет легко масштабировать систему в целом и выявлять ошибки кода, локализуя их на более низких уровнях.

Основной функцией **FRONTEND** является удаленный запуск расчета алгоритма классификации в системе Apache Spark и визуализация результатов в виде карты классификаций (рис. 10). Программные объекты **FRONTEND** взаимодействуют с компонентами **MIDDLEWARE** через протокол HTTP с помощью библиотеки axios-js (см. рис. 9).

Веб-сервис поддерживает следующие функциональные возможности: добавление/удаление файлов

```
{
  ...
  // Номер части (соответствует номеру задания Spark)
  "partition004": {...},
  "partition005": {
    "undefined": {...},
    // Тип заключения классификации
    "strictly": {
      // Координаты для отображения на карте классификаций
      "x": [15952,16030,16031],
      "y": [1, 1, 1],
      // Временная метка, соответствует окну сдвига в 1 с
      "times": [
        "2013-10-08 04:26:01",
        "2013-10-08 04:27:19",
        "2013-10-08 04:27:20"
      ]
    },
    "notstrictly": {...},
    "perhaps": {...}
  },
  "partition006": {...},
  "partition007": {...},
  // Названия каналов классифицируемого сигнала
  "channel1": "AN.BRCR.81.EHE.D.2013.281",
  "channel2": "AN.BRCR.81.EHN.D.2013.281",
  "channel3": "AN.BRCR.81.EHZ.D.2013.281",
  // Параметры фильтрации, используются на стороне FRONTEND
  "filter": {
    "startTime": "", "endTime": "", "isApply": false,
    "onlyBlastStrictly": false
  },
  // Начальное и конечное время сигнала, используется для фильтрации
  // на стороне FRONTEND
  "signalStartTime": "2013-10-08 00:00:09",
  "signalEndTime": "2013-10-09 00:00:01"
}
```

Рис. 3. Результат классификации суточной записи сейсмического сигнала в виде карты классификаций в формате JSON

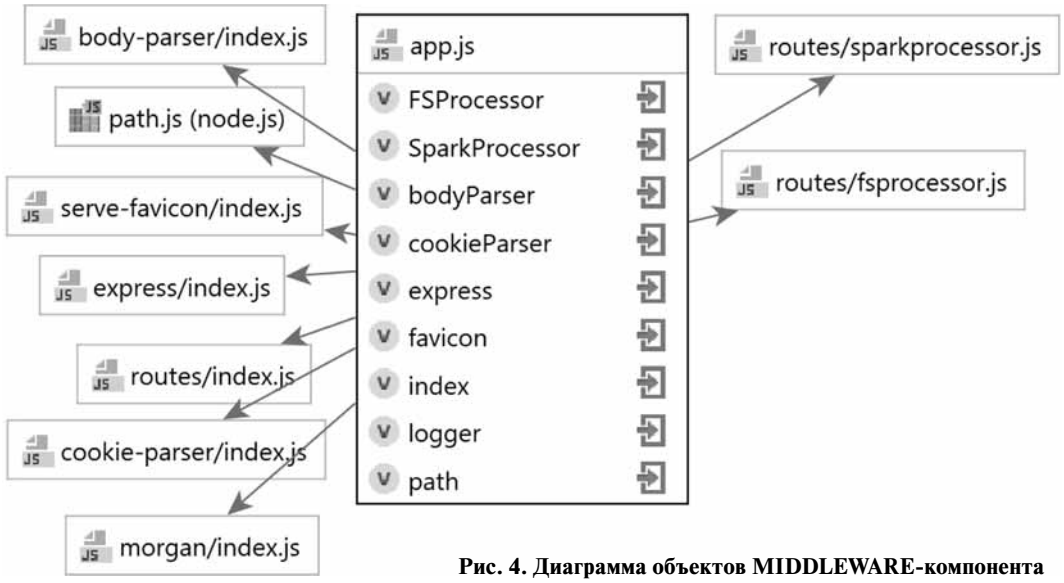


Рис. 4. Диаграмма объектов MIDDLEWARE-компонента

```
{
  // Основной java-класс
  "className": "org.myapp.seismatica.classifiers.ClassificationProcessor",
  // Путь к файлу программы
  "file": "hdfs://10.101.81.203/user/jars/seismatica/seismatica-classifier-1.0.jar",
  "name": "Seismatica - Classifier",
  "args": [
    username, // Имя пользователя
    "DistanceClassifier", // Название классификатора
    chFilesArg, // Массив путей к файлам каналов
    templateFile, // Путь к файлу шаблонов
    "100", // Шаг сдвига окна
    "8" // Количество задач (исполняются параллельно)
  ],
  "conf": {
    "spark.executor.instances": "4", // Количество Executor-объектов (работают параллельно)
    "spark.task.cpus": "1", // Количество CPU на одно задание
    "spark.executor.cores": "2", // Количество задач на один Executor
    "spark.executor.memory": "2g", // Выделяемая память для одного Executor
    "spark.driver.memory": "3g", // Выделяемая память для управляющей программы
    "spark.driver.extraClassPath": "/mnt/hdfs/user/jars/seismatica/*", // Путь к java-классам
    "spark.executor.extraClassPath": "/mnt/hdfs/user/jars/seismatica/*" // Путь к java-классам
  }
}
```

Рис. 5. JSON-объект (параметр body) в POST-запросе к сервису Apache Livy на удаленный запуск Spark-задания

```
function getResultAsJSON(resultFile) {

    // Чтение файла результата из HDFS, преобразование в JSON-объект
    let result = JSON.parse(fs.readFileSync(resultFile, 'utf8'));
    // Получение частей
    let partitions = Object.keys(result).filter(key => key.includes('partition'))
        .filter(key => Object.keys(result[key]).length > 0).sort();

    // Объединение частей (для каждого из заключений классификации)
    let reducedResult = {
        undefined: {
            x: reduceResultPartition(result, partitions, 'undefined', 'x'),
            y: reduceResultPartition(result, partitions, 'undefined', 'y'),
            times: reduceResultPartition(result, partitions, 'undefined', 'times')
        },strictly: {...}, notstrictly: {...}, perhaps: {...}
    };

    // Объединение временных меток (для каждого из заключений классификации)
    reducedResult.times = [...reducedResult.undefined.times,
    ...reducedResult.strictly.times,...reducedResult.notstrictly.times,
    ...reducedResult.perhaps.times].sort();

    // Добавление остальных параметров в JSON-объект
    reducedResult.channel1 = result.channel1;
    reducedResult.channel2 = result.channel2;
    reducedResult.channel3 = result.channel3;
    ...
    return reducedResult;
}

// Функция редукции одного из элементов из каждой части JSON-объекта
function reduceResultPartition(result, resultParts, conclusion, conclusionKey) {
    return resultParts.reduce((accumulator, currentValue, currentIndex, array) => {
            return [...accumulator, ...result[currentValue][conclusion][conclusionKey]];
        }, []);
}
```

Рис. 6. Фрагмент кода метода getResultAsJSON

```
router.get('/getResult', function (req, res, next) {

    // Установка заголовка ответа для доступа со стороны FRONTEND
    res.setHeader("Access-Control-Allow-Origin", "*");
    res.setHeader("Access-Control-Allow-Headers",
        "Origin, X-Requested-With, Content-Type, Accept");

    // Путь к файлу результата в HDFS
    let resultFile = '/mnt/hdfs/user/' + req.query.username
                   + '/seismatica/results/' + req.query.appId + '.json';
    if (fs.existsSync(resultFile)) {
        // Чтение файла результата из HDFS
        // и отправка его в виде JSON-объекта на сторону FRONTEND
        res.send({[req.query.appId]: getResultAsJSON(resultFile)});
    }
    else {
        res.send({});
    }
});
```

Рис. 7. Взаимодействие с распределенной файловой системой HDFS (на примере GET-запроса /getResult)

каналов сейсмостанции в формате miniSEED (сохраняются в HDFS, доступны со всех узлов кластера) и формирование сейсмического сигнала; запуск процесса классификации для выбранного сигнала, получение результатов с сервера, удаление результата из HDFS; фильтрация отображения результата по времени; построение карты классификации выбранного результата; аутентификации на базе системы Apache Hue; просмотр данных каналов в соответствии с выбранным уровнем масштабирования карты классификаций; поддержка русского/английского языка интерфейса.

На рис. 11 представлена диаграмма потоков данных при различных взаимодействиях пользователя с веб-сервисом. Диаграмма отражает основные процессы, происходящие на стороне **BACKEND-** и **MIDDLEWARE**-компонентов, их взаимодействие с системой Apache Spark и HDFS, типы данных, передаваемые между процессами, и места их расположения в контексте всей системы.

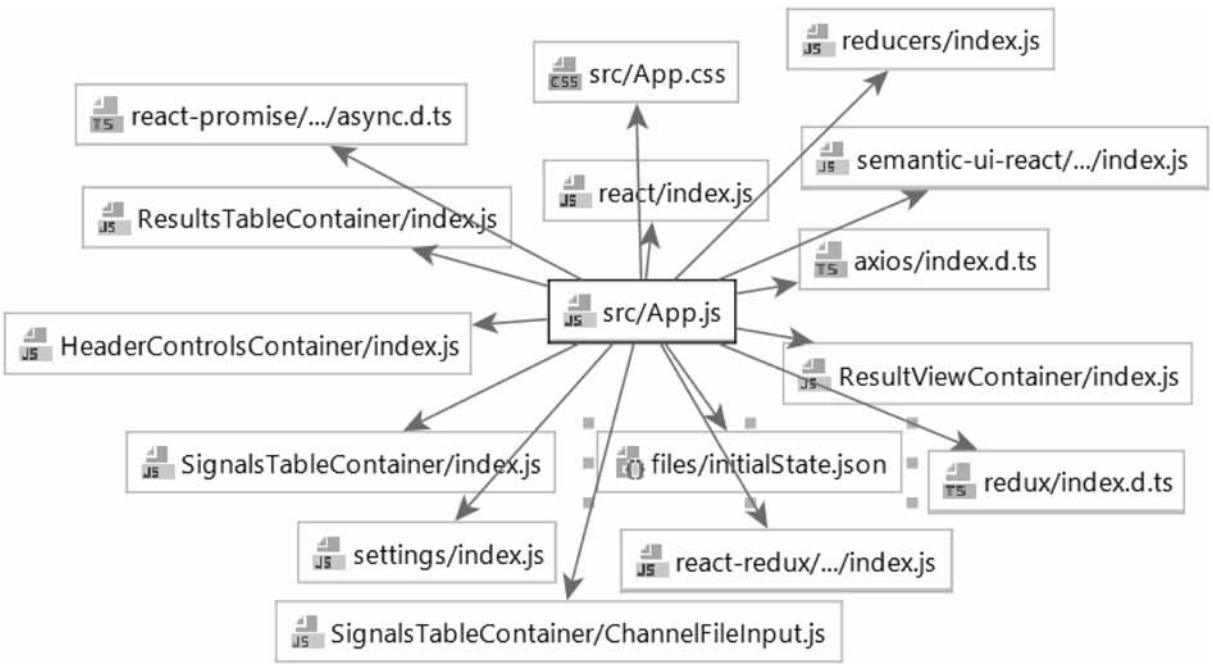


Рис. 8. Диаграмма объектов FRONTEND-компонента

```
class SubmitButton extends React.PureComponent {
  ...
  onSubmitHandler() {
     ...
     // Передача параметров HTTP-запросу
     let templateFile = '/mnt/hdfs/user/seismo_usr/seismatica' +
                        ' /templates/SeisPatterns.csv';
     axios.get('http://' + settings.MIDDLEWARE_IP + ':'
         + settings.MIDDLEWARE_PORT + '/' + settings.SPARK_PROCESSOR
         + '/sparkSubmit?' + 'username=' + this.props.login.userName + '&'
         + 'chFiles='+ signal.channels[0] + ',' + signal.channels[1] + ','
         + signal.channels[2] + '&' + 'templateFile=' + templateFile
     // Выполнение GET-запроса, получение ответа и его обработка
     ).then((response) => {
        this.props.onSubmitSignal(response.data.id);})}

     // Размещение элемента на странице
     render() {
        return (
         <Button floated='right' icon labelPosition='left' size='small'
                 onClick={this.onSubmitHandler.bind(this)}
                 ...
         </Button>);}
```

Рис. 9. Фрагмент кода отправки задания в систему Apache Spark в компоненте FRONTEND (SubmitButton.js)

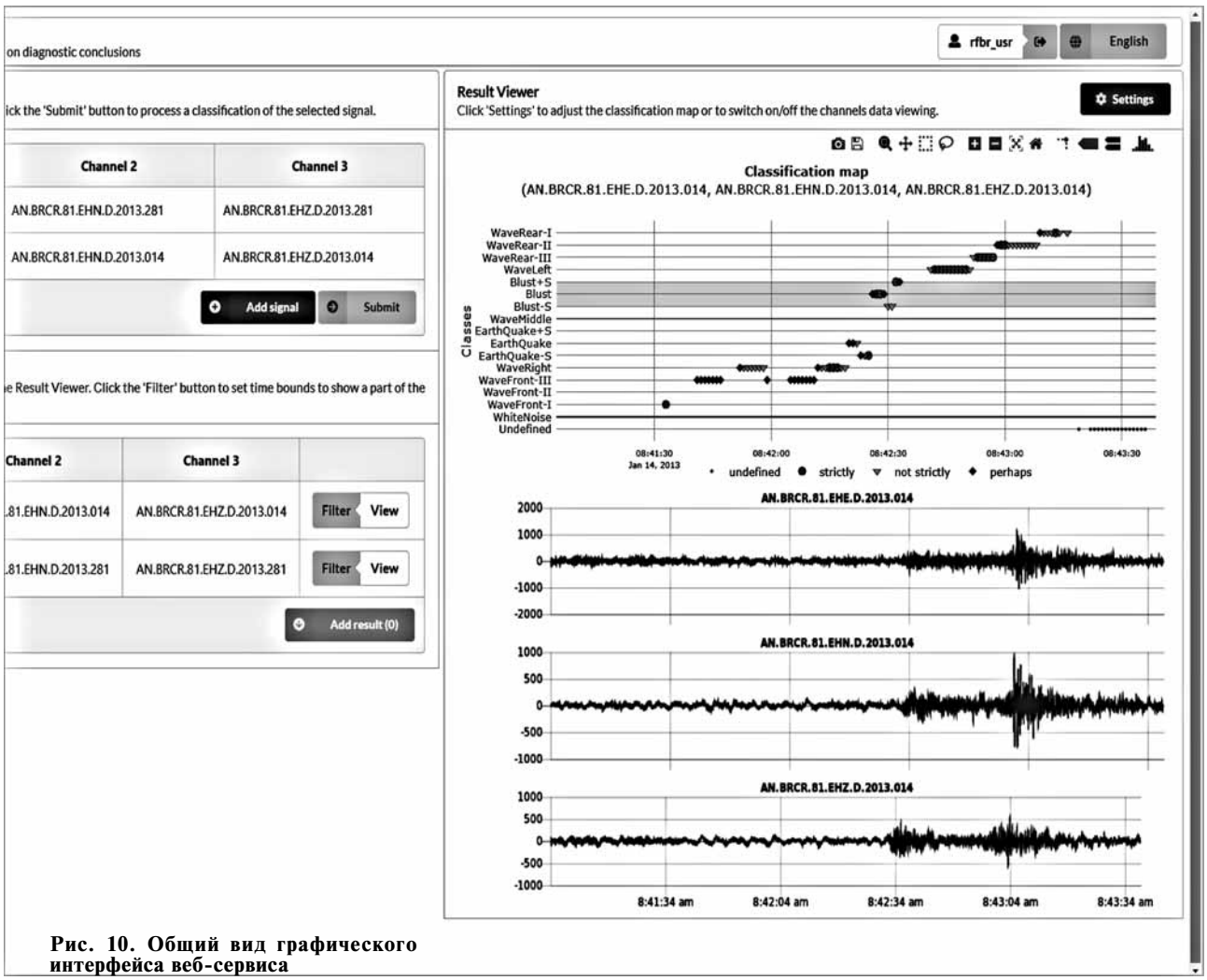


Рис. 10. Общий вид графического интерфейса веб-сервиса

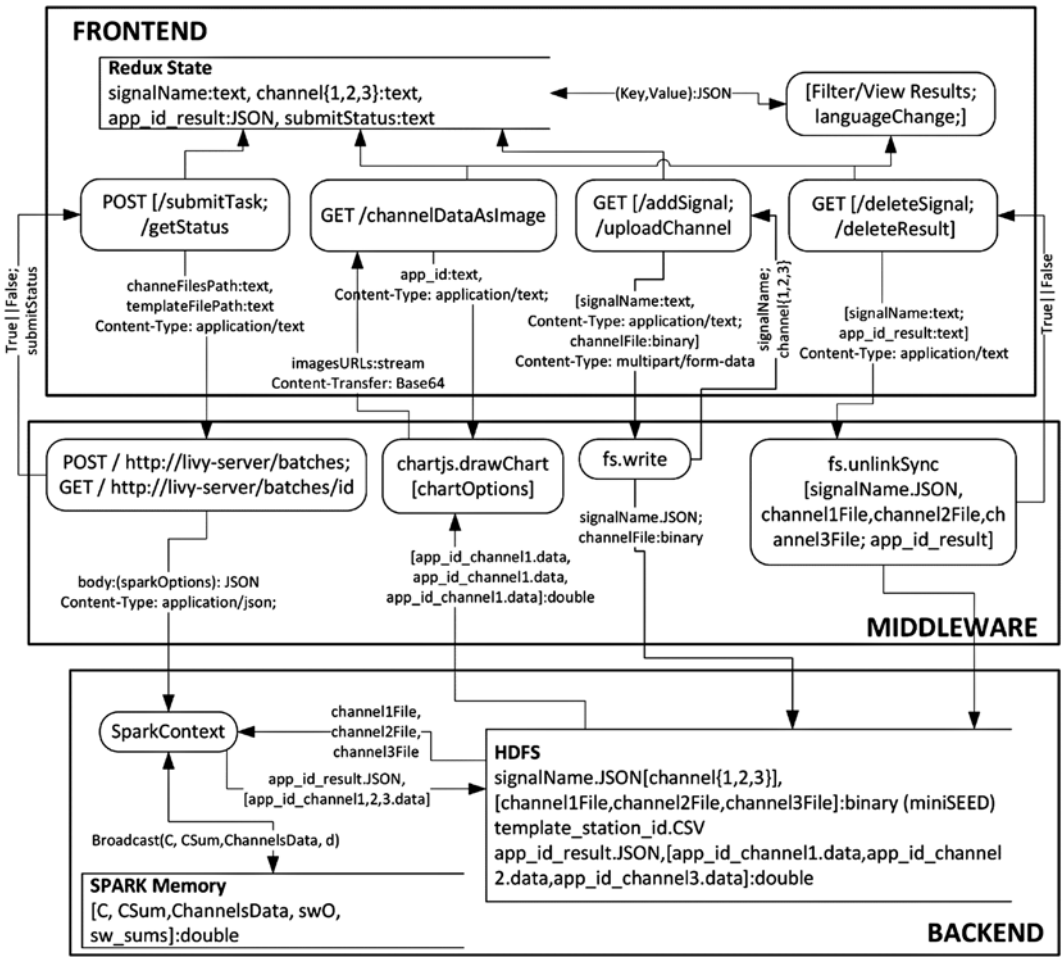


Рис. 11. Диаграмма потоков данных (Data Flow Diagram) процессов взаимодействия компонентов веб-сервиса

Тест по оценке производительности

Тест, направленный на оценку производительности системы, проводился на примере запуска процесса классификации набора суточных записей со станции BRCR. Было проведено 150 запусков. Файлы сигналов (три канала) не повторялись. В табл. 2 указано среднее время работы алгоритма. Представлены четыре программные реализации алгоритма в средах Matlab, Java (консольное приложение) — локальный тест, Java, Python (Spark API приложение) — распределенный тест. Фиксировалось только время расчета от подачи входных параметров (файлы каналов, файл шаблонов, параметры настройки Spark и др.) до получения JSON-файла карты классификаций.

В качестве пояснения необходимо указать, что сравнение с другими алгоритмами классификаций сейсмических событий, которых насчитывается большое количество, выходит за рамки данной работы и исследования в целом. Не представляется возможным проанализировать математические модели, на основе которых построены эти алгоритмы и их программные реализации. Как следствие, сложно дать оценку возможности запуска в параллельном/распределенном режимах, написать такой программный код и провести его оптимизацию.

Поставленная в рамках рассматриваемого исследования задача решалась с позиции получения оптимального времени исполнения разработанного алго-

Таблица 2

Тест производительности программной реализации представленного алгоритма классификации (время работы, с)

Java Spark API	Python Spark API	Matlab	Java
Программный код запускается параллельно на 30 вычислительных ядрах, с разбивкой на partitions		Программный код запускается последовательно, без применения Apache Spark API	
Суточная запись			
27	34	3574	801
Недельный таймфрейм			
224	283	25 145	5599

Примечание. Суточная запись, синхронизированная по каналам в среднем, составляла 8 355 839 отчетов с интервалом в 10 мс или 83558 сдвигов. Аппаратное обеспечение: 2 сервера (AMD Ryzen 1700 (8 + 8 cores (Simultaneous Multi-Threading)) 3.2 GHz, 16Gb RAM, 1Gb/s скорость передачи данных между серверами). Локальный тест проводился на одном сервере.

ритма, в целях дальнейшего применения его в режиме потокового приема и обработки сигнала с сохранением точности результатов. Однако даже в сравнении с алгоритмом FAST [16], представленным в обзоре, получено 25-кратное увеличение производительности. При этом при увеличении рабочих узлов в кластере и, соответственно, количества ядер пропорционально будет уменьшаться и время работы алгоритма, что станет возможным за счет запуска большего числа заданий как на один суточный сигнал, так и за счет увеличения одновременно работающих процессов по количеству сигналов в недельном таймфрейме.

Авторами проведены сравнения с протоколами наблюдений службы геофизического мониторинга Кемеровской области. Полученные результаты в 95 % случаев (выборка 2013 г., всего около 500 событий (промышленный взрыв) с двух станций) полностью совпадали по типам заключений.

Заключение

Разработан веб-сервис для детерминирования и идентификации сейсмических событий с возможностью построения визуальных карт классификаций. Графическое представление элементов (заключений), распределенных во времени прохождения волны в заданном интервале, описывается классами возмущений сейсмического сигнала на базе характеристических функций (шаблонов). Алгоритм классификации, применяемый в вычислительном ядре сервиса, успешно адаптирован для его запуска в распределенном режиме реализации на основе массово-параллельного исполнения заданий в среде Apache Spark. Проведенные тесты производительности показали, что предложенный подход к оптимизации математической модели и программной реализации алгоритма позволяет применять его для потоковой обработки сигнала в виду очень малого времени выполнения программного кода.

В работе продемонстрированы механизмы интеграции современных веб-технологий построения интернет-приложений с элементами кластерной инфраструктуры. По мнению авторов, такой подход позволит разрабатывать подобные решения и в других областях научно-технической деятельности.

Список литературы

1. **Scarpetta S., Giudicepietro F., Ezin E. C., Petrosino S., Del Pezzo E., Martini M., Marinaro M.** Automatic Classification of Seismic Signals at Mt. Vesuvius Volcano, Italy, Using Neural Networks // Bulletin of the Seismological Society of America. 2005. Vol. 95, No. 1. P. 185—196.
2. **Benbrahim M., Daoudi A., Benjelloun K., Ibenbrahim A.** Discrimination of Seismic Signals Using Artificial Neural Networks // Proceedings of world academy of science, engineering and technology. 2005. Vol. 4. P. 4—7.
3. **Diersena S., Leeb E-J., Spearsc D., Chenb P., Wanga L.** Classification of Seismic Windows Using Artificial Neural Networks // Procedia Computer Science. 2011. Vol. 4. P. 1572—1581.
4. **Hamer R. M., Cunningham J. W.** Cluster analyzing profile data confounded with interrater differences: A comparison of profile association measures // Applied Psychological Measurement. 1981. Vol. 5. P. 63—72.
5. **Kedrov E. O., Kedrov O. K.** Spectral time method of identification of seismic events at distances of 15°—40° // Izvestiya, Physics of the Solid Earth. 2006. Vol. 42, No. 5. P. 398—415.
6. **Langer H., Falsaperla S., Powell T., Thompson G.** Automatic classification and a-posteriori analysis of seismic event identification at Soufrière Hills volcano, Montserrat // Journal of Volcanology and Geothermal Research. 2006. Vol. 153, No. 1. P. 1—10.
7. **Lyubushin Jr. A. A., Kaláb Z., Častová N.** Application of Wavelet Analysis to the Automatic Classification of Three-Component Seismic Records // Izvestiya, Physics of the Solid Earth. 2004. Vol. 40, No. 7. P. 587—593.
8. **Musil M., Pleginger A.** Discrimination between Local Microearthquakes and Quarry Blasts by Multi-Layer Perceptrons and Kohonen Maps // Bulletin of the Seismological Society of America. 1996. Vol. 86, No. 4. P. 1077—1090.
9. **Ryzhikov G. A., Biryulina M. S., Husebye E. S.** A novel approach to automatic monitoring of regional seismic events // IRIS Newsletter. 1996. Vol. XV, No. 1. P. 12—14.
10. **Shimshoni Y., Intrator N.** Classification of Seismic Signals by Integrating Ensembles of Neural Networks // IEEE transactions on signal processing. 1998. Vol. 46, No. 5 P. 1194—1201.
11. **Ryan T. M., Borisov D., Lefebvre M., Tromp J.** SeisFlows — Flexible waveform inversion software // Computers & Geosciences. 2018. Vol. 115. P. 88—95.
12. **Lesage P.** Interactive Matlab software for the analysis of seismic volcanic signals // Computers & Geosciences. 2009. Vol. 35, Iss. 10. P. 2137—2144.
13. **Wenxiang Jiang, Haiying Yu, Li Li, Lei Huang.** A Robust Algorithm for Earthquake Detector // Proceedings of the 15 World Conference on Earthquake Engineering. Lisbon. Portugal. 2012. URL: https://www.iitk.ac.in/nicee/wcee/article/WCEE2012_1098.pdf
14. **Alvarez I., García L., Mota S., Cortes G., Benítez C., De la Torre A.** An Automatic P-Phase Picking Algorithm Basedon Adaptive Multiband Processing // IEEE Geoscience and remote sensing letters. 2013. Vol. 10, No. 6. P. 1488—1492.
15. **Guilherme M., António R.** A neural network seismic detector // IFAC Proceedings Volumes. 2009. Vol. 42, Iss. 19. P. 304—309.
16. **Clara E. Y., Ossian O'R., Karianne J. B., Beroza G. C.** Earthquake detection through computationally efficient similarity search // Science Advances. 2015. Vol. 1. P. e1501057(1—13).
17. **Paul B. Q., Pierre G., Yoann C., Munkhuu U.** Detection and classification of seismic events with progressive multichannel correlation and hidden Markov models // Computers & Geosciences. 2015. Vol. 83. P. 110—119.
18. **IRIS.** Incorporated Research Institutions for Seismology. URL: https://www.iris.edu/hq/ (дата обращения 04.05.2018).
19. **Romero J. E., Titos M., Bueno Á., Álvarez I., García L., de la Torre Á., Benítez C.** APASVO: A free software tool for automatic P-phase picking and event detection in seismic traces // Computers & Geosciences. 2016. Vol. 90. Part A. P. 213—220.
20. **GeoSeisQC.** URL: http://www.geoleader.ru/index.php/ru/produkty-ru/geoseicqc (дата обращения 07.05.2018).
21. **ZETLAB** Детектор STA/LTA. URL: https://zetlab.com/shop/programmnoe-obespechenie/funktsii-zetlab/analiz-signalov/detektor-sta-lta/ (дата обращения 07.05.2018).
22. **Stratimagic.** URL: http://www.pdgm.com/products/stratimagic/ (дата обращения 07.05.2018).
23. **Разработка** и создание Грид-приложений для решения прикладных задач геофизики (гранты РФФИ 10-07-00491-а). URL: http://www.rfbr.ru/rffi/ru/project_search/o_49145 (дата обращения 07.05.2018).
24. **Использование** слабо связанных вычислительных систем для решения обратных задач геофизики" (гранты РФФИ 11-05-00988-а). URL: http://www.rfbr.ru/rffi/ru/project_search/o_43212 (дата обращения 07.05.2018).
25. **Разработка** GRID-системы и вычислительных сервисов для исследования геодинамических пространственно-временных процессов по данным ДЗЗ (гранты РФФИ 11-07-12045-офи). URL: http://www.rfbr.ru/rffi/ru/project_search/o_46676 (дата обращения 07.05.2018).
26. **Distance** computations URL:https://docs.scipy.org/doc/scipy/reference/spatial.distance.html (дата обращения 11.05.2018).
27. **Замараев Р. Ю., Попов С. Е., Логов А. Б.** Алгоритм классификации сейсмических событий на основе энтропийного отображения сигналов // Физика Земли. 2016. № 3. С. 31—37.
28. **McKay D.** Information Theory, Inference, and Learning Algorithms // Cambridge: Cambridge University Press, 2003. 631 p.
29. **Kortström J., Uski M., Tiira T.** Automatic classification of seismic events within a regional seismograph network // Computers & Geosciences. 2016. Vol. 87. P. 22—30.
30. **Guojun Gan, Chaoqun Ma, Jianhong Wu.** Data clustering: theory, algorithms, and applications (ASA-SIAM series on statistics and applied probability). Society for Industrial and Applied Mathematics Philadelphia. PA. USA. 2007. 451 p.

The Web-Service for the Classification of Seismic Events Based on Apache Spark API

S. E. Popov, e-mail: popov@ict.sbras.ru, **R. Ju. Zamaraev,** e-mail: zamaraev@ict.sbras.ru, **I. E. Kharlampenkov,** e-mail: kharlampenkov@ict.sbras.ru, Institute of Computational Technologies SB RAS, Novosibirsk, 630090, Russian Federation

Corresponding author:

Popov Semion E., Senior Researcher, Institute of Computational Technologies SB RAS, Novosibirsk, 630090, Russian Federation. E-mail: popov@ict.sbras.ru

Received on May 29, 2018
Accepted on June 13, 2018

The article describes the key points of the service development process for fast automatic classification of seismic signals based on diagnostic templates. The software solutions for the preliminary signal processing and algorithm of parallel computations of the mathematical model for the development of final conclusions on the basis of rating voting are presented. Their integration with the Apache Spark distributed computing system is shown. Performance tests of the classification algorithm for a set of daily signals in various software environments were conducted. It is shown that the launch of the classification algorithm in the context of massively-parallel execution of the problem gives a gain in productivity (a decrease in the operating time) by several tens of times. The service was developed using the libraries React and Redux. The NodeJS platform is used as the runtime environment.

Keywords: *web service, distributed computing, Apache Spark, classification of seismic events*

Acknowledgements: *The study was carried out with the financial support of the Russian Foundation for Basic Research within the framework of a research project No. 18-07-00013A*

For citation:

Popov S. E., Zamaraev R. Ju., Kharlampenkov I. E. The Web-Service for the Classification of Seismic Events Based on Apache Spark API, *Programmnaya Ingeneria*, 2018, vol. 9, no. 7, pp. 318—331.

DOI: 10.17587/prin.9.318-331

References

1. **Scarpetta S., Giudicepietro F., Ezin E. C., Petrosino S., Del Pezzo E., Martini M., Marinaro M.** Automatic Classification of Seismic Signals at Mt. Vesuvius Volcano, Italy, Using Neural Networks, *Bulletin of the Seismological Society of America,* 2005, vol. 95, no. 1, pp. 185—196.
2. **Benbrahim M., Daoudi A., Benjelloun K., Ibenbrahim A.** Discrimination of Seismic Signals Using Artificial Neural Networks, *Proceedings of world academy of science, engineering and technology,* 2005, vol. 4, pp. 4—7.
3. **Diersena S., Leeb E-J., Spearsc D., Chenb P., Wanga L.** Classification of Seismic Windows Using Artificial Neural Networks, *Procedia Computer Science*, 2011, vol. 4, pp. 1572—1581.
4. **Hamer R. M., Cunningham J. W.** Cluster analyzing profile data confounded with interrater differences: A comparison of profile association measures, *Applied Psychological Measurement,* 1981, vol. 5, pp. 63—72.
5. **Kedrov E. O., Kedrov O. K.** Spectral time method of identification of seismic events at distances of 15°...40°, *Izvestiya, Physics of the Solid Earth*, 2006, vol. 42, no. 5, pp. 398—415.
6. **Langer H., Falsaperla S., Powell T., Thompson G.** Automatic classification and a-posteriori analysis of seismic event identification at Soufrière Hills volcano, Montserrat, *Journal of Volcanology and Geothermal Research,* 2006, vol. 153, no. 1, pp. 1—10.
7. **Lyubushin Jr. A. A. Kaláb Z, Častová N.** Application of Wavelet Analysis to the Automatic Classification of Three-Component Seismic Records, *Izvestiya, Physics of the Solid Earth*, 2004, vol. 40, no. 7, pp. 587—593.
8. **Musil M., Pleginger A.** Discrimination between Local Microearthquakes and Quarry Blasts by Multi-Layer Perceptrons and Kohonen Maps, *Bulletin of the Seismological Society of America*, 1996, vol. 86, no. 4, pp. 1077—1090.
9. **Ryzhikov G. A., Biryulina M. S., Husebye E. S.** A novel approach to automatic monitoring of regional seismic events, *IRIS Newsletter,* 1996, vol. XV, no. 1, pp. 12—14.
10. **Shimshoni Y., Intrator N.** Classification of Seismic Signals by Integrating Ensembles of Neural Networks, *IEEE transactions on signal processin,* 1998. vol. 46, no. 5, pp. 1194—1201.
11. **Ryan T. M., Borisov D., Lefebvre M., Tromp J.** SeisFlows — Flexible waveform inversion software, *Computers & Geoscience*s, 2018, vol. 115, pp. 88—95.
12. **Lesage P.** Interactive Matlab software for the analysis of seismic volcanic signals, *Computers & Geosciences*, 2009, vol. 35, Issue 10, pp. 2137—2144.
13. **Wenxiang Jiang, Haiying Yu, Li Li, Lei Huang.** A Robust Algorithm for Earthquake Detector, Proceedings of the 15 World Conference on Earthquake Engineering. Lisbon. Portugal. 2012, available at: https://www.iitk.ac.in/nicee/wcee/article/WCEE2012_1098.pdf
14. **Alvarez I., García L., Mota S., Cortes G., Benítez C., De la Torre A.** An Automatic P-Phase Picking Algorithm Basedon Adaptive Multiband Processing, *IEEE Geoscience and remote sensing letters*, 2013, vol. 10, no. 6, pp. 1488—1492.
15. **Guilherme M., António R.** A neural network seismic detector, *IFAC Proceedings Volumes,* 2009, vol. 42, iss. 19, pp. 304—309.
16. **Clara E. Y., Ossian O'R., Karianne J. B., Beroza G. C.** Earthquake detection through computationally efficient similarity search, *Science Advances*, 2015, vol. 1, pp. e1501057(1—13).
17. **Paul B. Q., Pierre G., Yoann C., Munkhuu U.** Detection and classification of seismic events with progressive multichannel correlation and hidden Markov models, *Computers & Geosciences*, 2015, vol. 83, pp. 110—119.
18. **IRIS.** Incorporated Research Institutions for Seismology, available at: https://www.iris.edu/hq/ (date of access 04.05.2018).
19. **Romero J. E., Titos M., Bueno Á., Álvarez I., García L., de la Torre Á., Benítez C.** APASVO: A free software tool for automatic P-phase picking and event detection in seismic traces, Computers & Geosciences, 2016, vol. 90, part A, pp. 213—220.
20. **GeoSeisQC,** available at: http://www.geoleader.ru/index.php/ru/produkty-ru/geoseicqc (date of access 07.05.2018).
21. **ZETLAB** Detektor STA/LTA, available at: https://zetlab.com/shop/programmnoe-obespechenie/funktsii-zetlab/analiz-signalov/detektor-sta-lta/ (date of access 07.05.2018).
22. **Stratimagic**, available at: http://www.pdgm.com/products/stratimagic/ (date of access 07.05.2018).
23. **Razrabotka** i sozdanie Grid-prilozhenij dlja reshenija prikladnyh zadach geofiziki (grant RFFI 10-07-00491-a), available at: http://www.rfbr.ru/rffi/ru/project_search/o_49145 (date of access 07.05.2018) (in Russian).
24. **Ispol'zovanie** slabo svjazannyh vychislitel'nyh sistem dlja reshenija obratnyh zadach geofiziki (grant RFFI 11-05-00988-a, available at: http://www.rfbr.ru/rffi/ru/project_search/o_43212 (date of access 07.05.2018) (in Russian).
25. **Razrabotka** GRID-sistemy i vychislitel'nyh servisov dlja issledovanija geodinamicheskih prostranstvenno-vremennyh processov po dannym DZZ (grant RFFI 11-07-12045-ofи), available at: http://www.rfbr.ru/rffi/ru/project_search/o_46676 (date of access 07.05.2018) (in Russian).
26. **Distance** computations, available at: https://docs.scipy.org/doc/scipy/reference/spatial.distance.html (date of access 07.05.2018).
27. **Zamaraev R. Ju., Popov S. E., Logov A. B.** Algoritm klassifikacii sejsmicheskih sobytij na osnove jentropijnogo otobrazhenija signalov (Algorithm for classification of seismic events based on entropy display of signals), *Fizika Zemli,* 2016, no 3, pp. 31—37 (in Russian).
28. **McKay D.** *Information Theory, Inference, and Learning Algorithms*. Cambridge, Cambridge University Press, 2003, 631 p.
29. **Kortström J., Uski M., Tiira T.** Automatic classification of seismic events within a regional seismograph network, *Computers & Geosciences,* 2016, vol. 87, pp. 22—30.
30. **Guojun Gan, Chaoqun Ma, Jianhong Wu.** *Data clustering: theory, algorithms, and applications (ASA-SIAM series on statistics and applied probability),* Society for Industrial and Applied Mathematics Philadelphia. PA. USA. 2007, 451 p.

УДК 004.4 DOI: 10.17587/prin.9.332-336

А. А. Скворцов, канд. техн. наук, доц. кафедры, e-mail: skvalexei@mail.ru,
Вятский государственный университет, г. Киров

Применение конечных автоматов при разработке программного обеспечения станков со сложной структурой

Рассмотрены вопросы, возникающие в процессе разработки программного обеспечения станков со сложной структурой, включая вопросы развития интерфейса управления и повышения производительности таких станков на примере токарно-карусельного станка модели 1516. Показано, что разработка программы, автоматизирующей процессы управления станком, расширяющей возможности интерфейса управления им, а также повышающей производительность оператора, упрощается путем моделирования работы устройств станков конечными автоматами.

Ключевые слова: *разработка программы, моделирование, конечный автомат, switch-технология, управляющее устройство, станок, система управления, микроконтроллер, интерфейс, производительность*

Введение

В последние годы в России активно проводят работы, направленные на замену релейно-контактных управляющих схем станков на микроконтроллерные. Такая замена позволяет значительно уменьшить габаритные размеры, потребляемую мощность и стоимость управляющих такими станками устройств. Снижение габаритных размеров и применение готовых электронных блоков, взаимодействующих с микроконтроллерами по стандартным протоколам, также позволяет расширить возможности интерфейса управления станком, сделать его более удобным и гибким. Однако расширение таких возможностей влечет за собой усложнение и без того сложного программного обеспечения станка. Кроме того, при переходе к микроконтроллерной системе управления необходимо сохранить или повысить производительность оператора станка. Предполагается, что задачу одновременного роста производительности оператора станка и расширения функциональных возможностей интерфейса управления позволит решить подход на основе моделирования устройств станков конечными автоматами.

Целью работы является оценка возможности и эффективности применения методологии на основе конечных автоматов при разработке программного обеспечения систем управления станками со сложной структурой.

Методика

Исследовалось применение автоматного подхода в программировании при проектировании программы управления токарно-карусельным станком модели 1516, обрабатывающим круглые заготовки больших диаметров. Такой станок является объектом со сложным поведением [1], так как различные устройства станка имеют разную реакцию на нажатие одних и тех же кнопок пульта управления в зависимости от режима работы. Кроме того, во время плавного многоступенчатого набора скорости вращения обрабатываемой детали программа должна позволить оператору выполнять другие подготовительные действия. Согласно [1, 2] работу объекта со сложным поведением удобно моделировать конечным автоматом. На рис. 1 представлен общий вид станка и перечень его составных частей.

Изначально система управления станком состояла из нескольких блоков релейно-контактных схем. Обновленная микроконтроллерная система управления станком состоит из трех блоков, соединенных по интерфейсу RS-485. К их числу относятся: контроллер конечников, контроллер пульта управления (ПУ) и контроллер муфт.

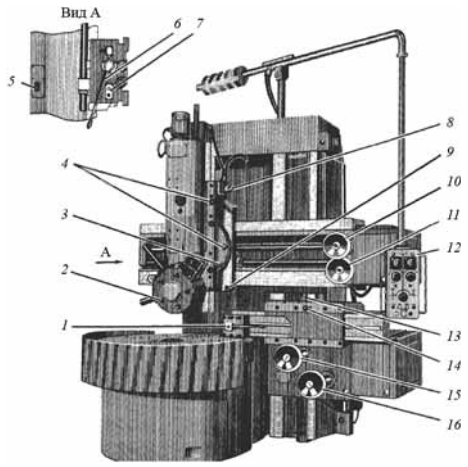


Рис. 1. Устройство одностоечного токарно-карусельного станка 1516:

1 — стол; *2* — ограждение планшайбы; *3* — вертикальный суппорт; *4* — подвесной пульт управления; *5* — подвеска пульта управления; *6* — поперечина; *7* — механизм перемещения поперечины; *8* — станина; *9* — механизм перемещения вертикального суппорта; *10* — коробка подач вертикального суппорта; *11* — коробка скоростей; *12* — кожух; *13* — механизм передачи движения на подачу; *14* — отверстие для заливки смазки; *15* — горизонтальный суппорт; *16* — коробка подач горизонтального суппорта

Контроллер конечников в ответ на запрос от контроллера ПУ передает байты конечных выключателей (конечников). Контроллер ПУ обрабатывает полученные с контроллера конечников байты, а также байты с расширителей портов кнопок и скоростей. В результате обработки поступившей на вход информации контроллер ПУ передает байты в расширители портов ламп и в контроллер муфт. Последний из перечисленных контроллер, в свою очередь, передает сигналы на муфты коробок подач суппортов и муфты коробки скоростей планшайбы.

Контроллер ПУ выполняет функции анализа входных данных и управления устройствами станка со сложным поведением. Программа такого контроллера ПУ вполне может рассматриваться как объект, для которого целесообразно применение автоматного подхода при его проектировании и реализации. Такой подход использован при создании программного обеспечения контроллера ПУ. Его суть заключалась в выполнении описанной далее последовательности действий.

1. По словесному описанию работы оператора станка для контроллера ПУ составлена схема связей. Обобщенная схема связей представлена на рис. 2.

Несколько устройств станка имеют сложное поведение, так как дают разную реакцию на нажатие одних и тех же кнопок: револьверная головка; планшайба; вертикальный суппорт; горизонтальный суппорт. Например, суппорты не должны перемещаться, пока не выбрано направление перемещения. Этот факт означает, что в начальном состоянии суппорты не должны реагировать на кнопку запуска движения "Пуск/стоп", но должны запомнить направление перемещения при нажатии соответствующей кнопки направления. В следующем состоянии — наоборот: суппорты реагируют на нажатие кнопки запуска движения и запускают перемещение, но на кнопки направления не реагируют. Так как в системе есть элементы со сложным поведением, необходимо перейти к п. 2 — моделированию системы управляющими автоматами.

2. Работа устройств станка со сложным поведением моделируется соответствующими конечными автоматами. Алгоритмы рабочего цикла и данных конечных автоматов представлены в разделе "Результаты исследования". Автоматы, моделирующие работу устройств, вызываются из рабочего цикла станка. В каждой итерации рабочего цикла выполняется чтение входных сигналов, выполнение шага (такта) автоматов, выдача выходных сигналов на исполнительные устройства и устройства индикации. Итерация рабочего цикла происходит через заданный такт, записанный в аппаратный таймер микроконтроллера. Таким образом, программа органично сочетает в себе элементы императивного подхода (рабочий цикл) и автоматного подхода (модели управляющих автоматов для устройств станка). Выполнение шага каждого автомата в одной итерации рабочего цикла обеспечивает параллельную работу отдельных устройств станка.

3. По схеме связей введены переменные, соответствующие входным и выходным воздействиям программы ПУ, а также переменные состояний для каждого моделируемого автомата.

4. По алгоритму рабочего цикла станка и описанию автоматов разработан код программы на языке Си в среде разработки MPLAB IDE [3]. Программная модель каждого управляющего автомата представляет собой блок switch-case на Си-подобных языках программирования или аналогичный блок на других языках. В таком автомате каждый блок case состоит из действий в данном состоянии и условий перехода в другие состояния, а именно:

```
switch (state) {
    case 0: < действия в состоянии 0 >
            < условия перехода в другие состояния >
            break;
    case 1: < действия в состоянии 1 >
            < условия перехода в другие состояния >
            break;
    ...
    case n: < действия в состоянии n >
            < условия перехода в другие состояния >
}
```

5. В настоящее время разработанный код привязан к микроконтроллеру PIC16F877A фирмы Microchip. Этот микроконтроллер представляет собой однокристальную восьмиразрядную микроЭВМ, состоящую из микропроцессора с RISC-архитектурой, ПЗУ программ объемом 8000 слов по 14 бит, ОЗУ 368 байт, ПЗУ данных объемом 256 байт, а также нескольких встроенных периферийных устройств. К числу таких устройств относятся: три таймера, модуль 10-разрядного АЦП, модуль ведущего синхронного последовательного порта MSSP (аппаратная поддержка обмена данными по протоколам SPI, I2C), модуль универсального синхронно-асинхронного приемопередатчика USART (последовательный порт с уровнями напряжений 0 и 5 В), пять портов ввода-вывода.

В данной программе один из таймеров использовался для отсчета времени такта рабочего цикла. Модуль USART использовался для обмена данными между микроконтроллерами ПУ, конечников и муфт.

Привязка программы к портам ввода-вывода микроконтроллера осуществляется с помощью зарезервированных переменных

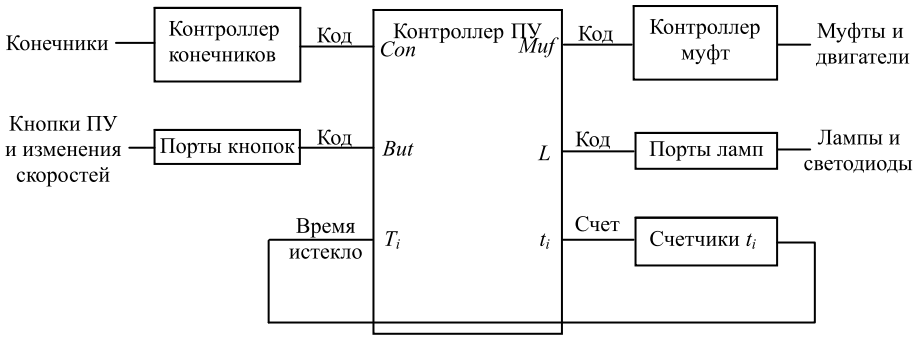


Рис. 2. Обобщенная схема связей контроллера ПУ станка:

Con — массив байтов конечников; *But* — массив байтов кнопок и скоростей; T_i — флаги счетчиков "Время истекло"; t_i — время, отсчитываемое счетчиками; *L* — массив байтов ламп; *M* — массив байтов муфт

компилятора Hitech PICC, а именно переменных PORTA, PORTB, PORTC, PORTD и PORTE. Возможно обращение к отдельным выводам портов с помощью переменных RA0...RA5, RB0...RB7, RC0...RC7, RD0...RD7 и RE0...RE3. После привязки программы к портам ввода-вывода микроконтроллера программа была собрана компилятором Hitech PICC в hex-файл (файл прошивки) для записи в память программ микроконтроллера. Программы микроконтроллера конечников и микроконтроллера муфт также разработаны в среде MPLAB IDE и собраны компилятором Hitech PICC.

6. Откомпилированный программный код протестирован во встроенном симуляторе MPLAB SIM, который позволяет смоделировать различные типы входных сигналов для выводов и пошагово проверить работу программы каждого микроконтроллера (микроконтроллер ПУ, микроконтроллер конечников, микроконтроллер муфт) в отдельности. После небольших доработок код протестирован на управляющих платах контроллера конечников, муфт, пульта управления, соединенных по интерфейсу RS-485. На конечном этапе программа тестировалась на переоборудованном станке.

Результаты исследования

В результате исследования разработаны схема связей контроллера ПУ, схема алгоритма рабочего цикла контроллера ПУ, описание конечных автоматов, моделирующих устройства станка со сложным поведением, на базе которых разработано программное обеспечение микроконтроллеров станка. На рис. 3 представлена схема алгоритма программы контроллера ПУ, а также в качестве примера представлены описание автомата и упрощенная схема алгоритма шага автомата (рис. 4) некоторых устройств станка со сложным поведением горизонтального и вертикального суппортов.

Далее приведено описание автоматов управления вертикальным и горизонтальным суппортом.

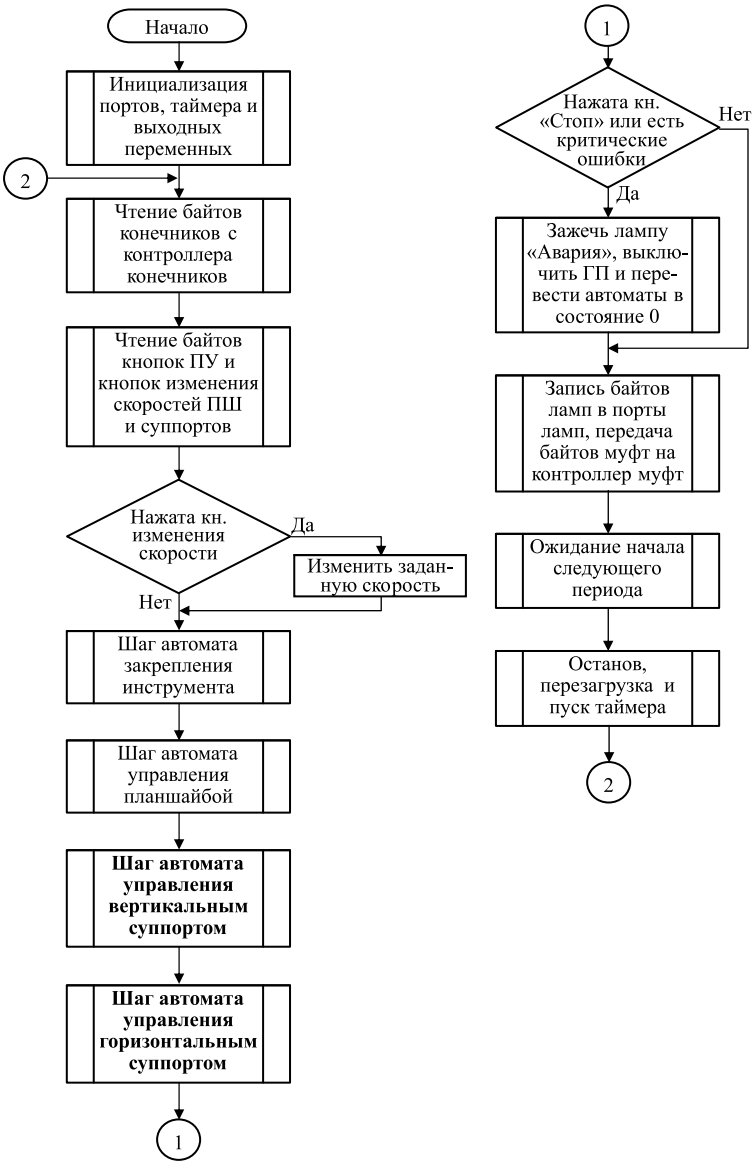


Рис. 3. Блок-схема алгоритма программы контроллера ПУ:
ГП — главный привод; ПШ — планшайба; кн. — кнопка

Состояние 0: выбор направления перемещения.
Если кнопка направления нажата, включить муфту направления, выключая муфты других направлений, запомнить направление, перейти в состояние 1.

Состояние 1: ожидание запуска движения.
Если нажата кнопка "Пуск/стоп":
если конечник направления выключен:
включить лампу центр. кнопки;
включить муфты движения и перейти в состояние 2.
Иначе выключить муфты направления и движения и перейти в состояние 0.

Состояние 2: Движение.
Если конечник направления сработал или кнопка "Пуск/стоп" нажата, включить тормозные муфты, остальные выключить и перейти в состояние 3.
Иначе если заданная скорость изменилась, включить муфты другой скорости.

Состояние 3: Задержка 2 с.
Инкремент t_1.
Если $t_1 = 40$, $t_1 = 0$, выключить тормозные муфты и перейти в состояние 0.

При включении станка происходит одновременный запуск программ контроллера конечников, контроллера ПУ и контроллера муфт. Контроллеры конечников и муфт являются ведомыми устройствами на шине RS-485, а контроллер ПУ — ведущим. После инициализации портов и периферийных устройств контроллеры конечников и муфт переходят в состояние ожидания обращения к ним со стороны контроллера ПУ по интерфейсу RS-485 последовательного порта с полудуплексным обменом. В это время контроллер ПУ проводит инициализацию портов, периферийных устройств, а также выходных переменных, связанных с лампами, муфтами и двигателями. После этого программа контроллера ПУ переходит в рабочий цикл.

В рабочем цикле сначала выполняется чтение входных сигналов с конечных выключателей (конечников): по интерфейсу RS-485 передается запрос на контроллер конечников и принимается ответ в виде нескольких байтов с битами состояния конечников (1 — нажат, 0 — не нажат). Затем выполняется чтение с расширителей портов ПУ байтов состояния кнопок ПУ и байтов установленных скоростей планшайбы и суппортов.

После чтения входных сигналов выполняется шаг (такт) управляющих автоматов устройств станка (автомат закрепления инструмента, автомат управления планшайбой, автомат управления вертикальным суппортом и автомат управления горизонтальным суппортом), которые в зависимости от состояния и входных сигналов переходят в другие состояния с формированием соответствующих выходных сигналов.

После выполнения шага автоматов программа передает байты выходных сигналов в контроллер муфт по интерфейсу RS-485. Контроллер муфт через расширители портов передает эти сигналы на лампы, муфты и двигатели.

Затем программа ожидает следующего "тика" таймера, задающего такт рабочего цикла. Дождавшись прерывания от таймера, программа останавливает таймер, загружает в него значение для нового отсчета, запускает таймер и переходит в начало рабочего цикла.

В случае команды аварийного или планового останова устройств станка программа переводит автоматы в состояния останова.

Благодаря почти одновременному выполнению шагов автоматов устройств станка в одном рабочем цикле программа ПУ позволяет оператору параллельно управлять планшайбой и суппортами станка, что обеспечивает повышенную производительность. Разработанная система управления станком успешно внедрена на производстве и в настоящее время активно используется.

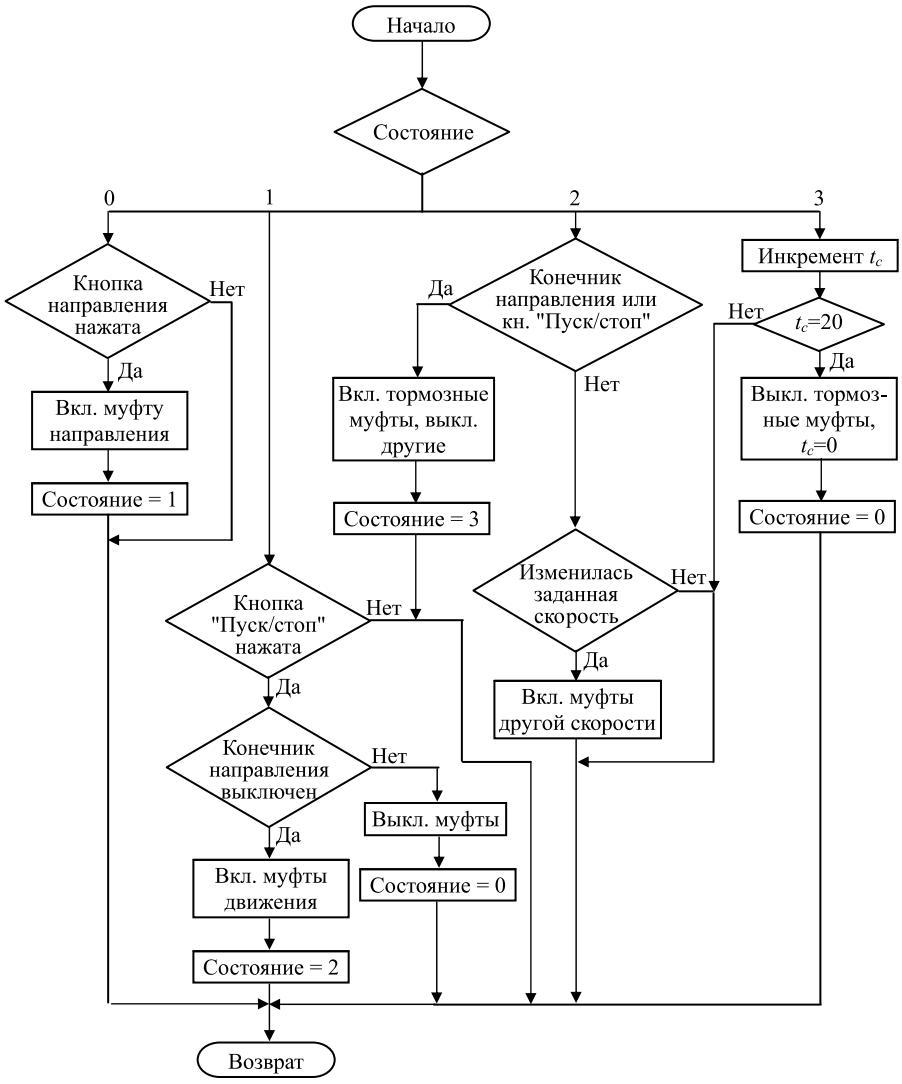


Рис. 4. Блок-схема алгоритма автоматов управления вертикальным и горизонтальным суппортом:

t_c — счетчик задержки перед торможением суппортов; значение 20 соответствует 2 с, так как время такта рабочего цикла — 100 мс

Заключение

Моделирование поведения устройств станка конечными автоматами оказалось довольно эффективным способом разработки программы для ПУ станка 1516. Применение такого подхода позволило: перевести станок с релейно-контактной системы управления на микроконтроллерную, не нарушая при этом логику управления; расширить возможности интерфейса взаимодействия с оператором за счет индикации скоростей и режимов работы; распараллелить выполнение отдельных операций

на станке и повысить производительность оператора. По результатам проделанной работы можно с высокой долей уверенности предполагать, что представленный подход будет эффективен и для программирования пультов управления другими станками со сложной структурой.

Список литературы

1. **Поликарпова Н. И., Шалыто А. А.** Автоматное программирование: Учебно-методическое пособие. СПб., 2007. 108 с.
2. **Шалыто А. А.** Автоматное программирование // Труды конференции "Технические и программные средства систем управления, контроля и измерения". М.: Институт проблем управления им. В. А. Трапезникова РАН, 2010, С. 156—67.
3. **MPLAB® IDE** User's Guide. Microchip Technology Inc., 2009. 288 p.

The Use of Finite State Machines in the Development of Software for Machines with Complex Structure

A. A. Skvortsov, skvalexei@mail.ru, Vyatka State University, Kirov, 610000, Russian Federation

Corresponding author:

Skvortsov Alexsey A., Associate Professor, Vyatka State University, Kirov, 610000, Russian Federation. E-mail: skvalexei@mail.ru

Received on February 03, 2018
Accepted on May 31, 2018

The article discusses the decrease in dimensions, the extension management interface, and improving performance of control devices of machine tools, for example lathe-turning lathe, model 1516. The extension management interface, and improving the performance of the operator is simplified by the use of automata-based programming. Replacement of relay-contact control circuit machine on the microcontroller to reduce size, as well as the expansion interface of the machine control and increased productivity leads to the complication of the software of the machine. In this case, often the machine and its individual devices become entities with state-dependent behavior. As we know from the works of A. A. Shalyto [1,2], automata-based programming is suitable for programming entities with state-dependent behavior, where the reaction entity with the same input action can be different and depends on the prehistory of input. The technology of imperative programming for such devices leads to the multiplication of temporary variables (flags) that increases the number of unforeseen states of the program and reduces its reliability. This also reduces the readability of the program, increases its complexity and memory usage. The aim of this work is to evaluate the use and effectiveness of automata-based programming machines with complex control system. An example of the development of a program for lathe-turning lathe model 1516 with the use of combination of technologies on a imperative and automata-based programming. The use of automata-based programming has brought the machine with relay-contact control system on the microcontroller, without disturbing the control logic, to extend the interface of interaction with the operator through indication of speeds and modes, and to parallelize the execution of individual operations on the machine and increase operator productivity. Similarly a program can be developed for other machines with complex control system.

Keywords: *automata-based programming, switch-technology, modelling, state machine, state-dependent behavior, control device, programming of machines, microcontroller, interface, performance*

For citation:

Skvortsov A. A. The Use of Finite State Machines in the Development of Software for Machines with Complex Structure, *Programmnaya Ingeneria*, 2018, vol. 9, no. 7, pp. 332—336.

DOI: 10.17587/prin.9.332-336

References

1. **Polikarpova N. I., SHalyto A. A.** *Avtomatnoe programmirovanie* (Automata-based programming): Uchebno-metodicheskoe posobie, Saint-Petersburg, 2007, 108 p. (in Russian).
2. **Shalyto A. A.** Avtomatnoe programmirovanie (Automata-based programming), *Trudy konferencii "Tekhnicheskie i programmnye sredstva sistem upravleniya, kontrolya i izmereniya"*, Moscow, Institut problem upravleniya im. V. A. Trapeznikova RAN, 2010, pp. 156—167 (in Russian).
3. **MPLAB®** IDE User's Guide. Microchip Technology Inc., 2009, 228 p.

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева*. Корректор *Н. В. Яшина*

Сдано в набор 15.06.2018 г. Подписано в печать 23.07.2018 г. Формат 60×88 1/8. Заказ PI718
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**

Рисунки к статье Д. И. Читалова, С. Т. Калашникова

«РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОДГОТОВКИ РАСЧЕТНЫХ СЕТОК С ПОМОЩЬЮ УТИЛИТЫ foamyQuadMesh ПЛАТФОРМЫ OpenFOAM»

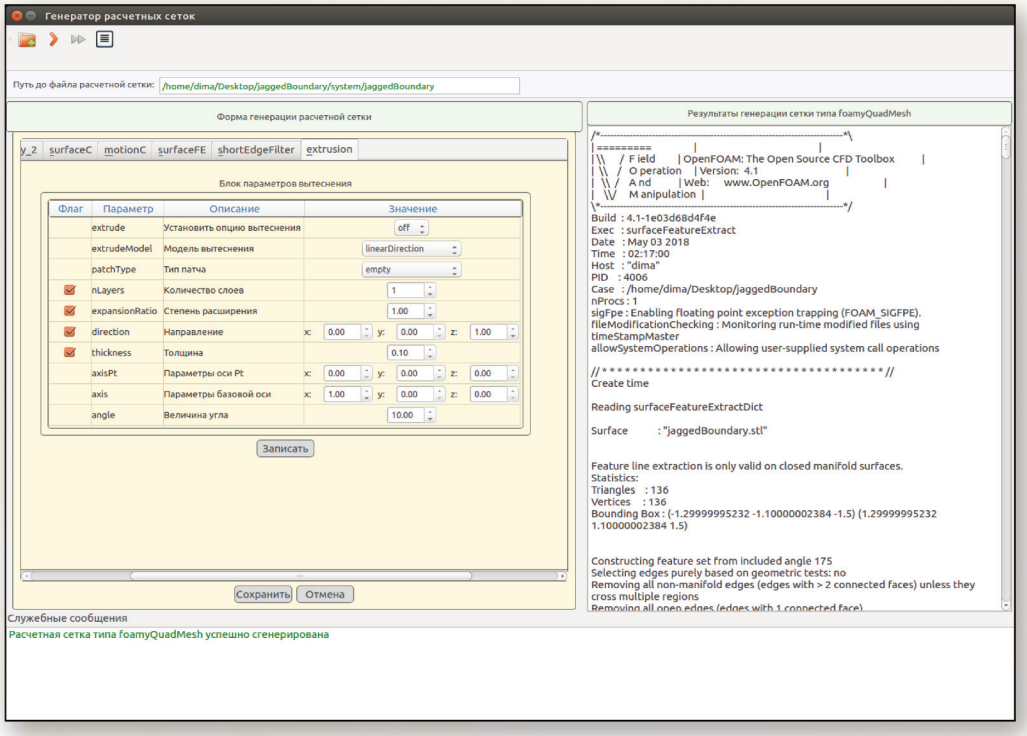


Рис. 5. Основное окно приложения foamyQuadMesh_generator после генерации РС

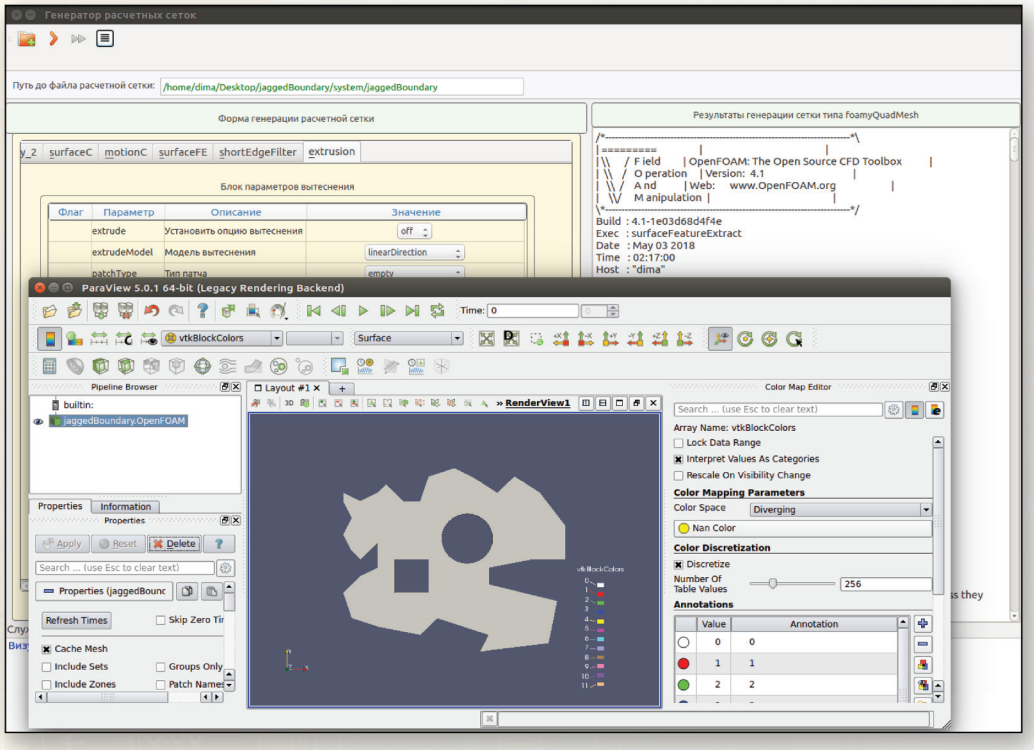


Рис. 6. Основное окно приложения foamyQuadMesh_generator после визуализации РС

SECR

14-я Научно-Практическая Конференция

ВСЕ АСПЕКТЫ РАЗРАБОТКИ ПО

ТЕХНОЛОГИИИ И СРЕДСТВА ПРОГРАММИРОВАНИЯ

УПРАВЛЕНИЕ ПРОДУКТАМИ И КОМАНДАМИ

ВЕДЕНИЕ БИЗНЕСА, ПОДБОР И ОБУЧЕНИЕ КАДРОВ

ТРЕНДОВЫЕ ВОПРОСЫ: AI, VR, IoT, BLOCKHAIN И ДР.

12-13 ОКТЯБРЯ, МОСКВА. WWW.SECRUS.ORG

ISSN 2220-3397. Программная инженерия. 2018. Том 9. № 7. 289–336. Подписной индекс по Объединенному каталогу «Пресса России» – 39795