ISSN 2220-3397

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

Том 14. № 2. 2023

Поздравляем юбиляра!

5 февраля 2023 г. исполнилось 75 лет главному редактору журнала «Программная инженерия» **Валерию Александровичу Васенину** – доктору физико-математических наук, профессору, лауреату Премии Правительства РФ, заведующему межфакультетской кафедрой «Математическое моделирование и компьютерные исследования» МГУ имени М. В. Ломоносова, заведующему лабораторией «Автоматизация экспериментальных исследований» НИИ механики МГУ.

Вся научная деятельность Валерия Александровича неразрывно связана с Московским государственным университетом имени М. В. Ломоносова, механико-математический факультет которого он окончил в 1972 г. Здесь в 1977 г. им была защищена кандидатская диссертация, а в 1998 г. – докторская.

Профессор В. А. Васенин – известный специалист в области математических методов, алгоритмического и программного обеспечения безопасных сетевых и информационно-вычислительных технологий. Он автор более 290 научных работ, в том числе 14 монографий. Премия Правительства РФ была присуждена ему за разработку научно-организационных основ и создание Федеральной университетской сети России RUNNet.

Результатом научно-педагогической деятельности В. А. Васенина является подготовка более чем 100 специалистов. Под его научным руководством защищены 23 кандидатские и 2 докторские диссертации.

Валерий Александрович сотрудничает с издательством «Новые технологии» около 20 лет – сначала как член редакционных советов журналов «Информационные технологии» и «Программная инженерия», работу по изданию которого он возглавил в качестве главного редактора с начала 2012 г.

Уважаемый Валерий Александрович!
Издательство «Новые технологии», редакционный совет и редакционная коллегия журнала «Программная инженерия» поздравляют Вас с юбилеем, желают крепкого здоровья и новых достижений в профессиональной деятельности!

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

Том 14
№ 2
2023

ПрИн

Учредитель: Издательство "НОВЫЕ ТЕХНОЛОГИИ"

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

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 подписных агентствах (индекс по Объединенному каталогу "Пресса России" — **22765**) или непосредственно в редакции (для юридических лиц).
Тел.: (499) 270-16-52.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в Российский индекс научного цитирования (РИНЦ) и Russian Science Citation Index (RSCI).
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.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 14
N 2
2023

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

CONTENTS

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

УДК 004.062, 004.046

DOI: 10.17587/prin.14.55-61

В. А. Васенин, д-р физ.- мат. наук, проф.,vasenin@msu.ru, **А. А. Зензинов,** мл. науч. сотр., **В. А. Роганов**, ст. науч. сотр., radug-a@ya.ru,
МГУ имени М. В. Ломоносова

Повышение производительности информационных сервисов в системах, ориентированных на работу с большими, редко модифицируемыми данными

Поступила в редакцию 10.12.2022
Принята к публикации 21.12.2022

Представлены результаты исследования, разработки и предварительного тестирования продвинутого механизма мемоизации вычислений, ориентированного на информационные системы, работающие с большими объемами редко модифицируемых первичных данных. Производительность информационных сервисов таких систем может быть существенно увеличена при помощи предоставляемых механизмов аккуратной работы с кеш-памятью, когда инвалидируются результаты тех и только тех вычислений, которые затрагивают проведенные изменения первичных данных.

Ключевые слова: *высокопроизводительные сервисы, функциональное программирование, мемоизация вычислений, трекинг зависимостей*

Для цитирования:

Васенин В. А., Зензинов А. А., Роганов В. А. Повышение производительности информационных сервисов в системах, ориентированных на работу с большими, редко модифицируемыми данными // Программная инженерия. 2023. Том 14, № 2. С. 55—61. DOI: 10.17587/prin.14.55-61.

Введение

Перманентный рост объемов обрабатываемых информационно-аналитическими системами данных в настоящее время ставит перед разработчиками острый вопрос повышения производительности прикладных информационных сервисов таких систем. В отличие от системных компонентов (ОС, СУБД, web-серверов и т. д.), оптимизированных за многие годы командами высококвалифицированных специалистов, прикладные программы зачастую проектируются и развиваются в условиях меняющихся потребностей заказчика и при объективном дефиците специалистов надлежащей квалификации. Эти обстоятельства не позволяют уделять вопросам масштабирования должного внимания. Как следствие, многие успешные информационно-аналитические системы испытывают серьезные трудности, особенно на этапах роста их популярности, числа пользователей и объемов данных.

По этой причине особый интерес представляют программные средства, способные значительно повысить производительность уже разработанных информационных систем без существенных дополнительных трудозатрат. В настоящей статье рассмотрены технологические аспекты подходов к ускорению выполнения запросов в информационно-аналитических системах со сложной логикой, но не слишком часто модифицируемыми данными.

Разработанный новый подход к ускорению базируется на гибридной мемоизации вычислений с трекингом зависимостей, обеспечивающей, с одной стороны, прямой доступ http-серверу во вторичную кеш-память, а с другой стороны, транзакционную надежность сохраняемых списков зависимостей, адаптированных к возможностям современных СУБД. В сочетании с параллелизмом и асинхронными операциями ввода-вывода такой подход позволяет обрабатывать сотни тысяч запросов в секунду на современном серверном обо-

рудовании. Инструментальные средства подобного рода на настоящее время особенно востребованы в России, так как позволяют обеспечить импортозамещение в части перехода на отечественные серверные платформы при решении практически важных задач.

Потенциал для повышения производительности типичных web-сервисов

Современное серверное оборудование и системное программное обеспечение (ОС, СУБД, http-серверы) способны обрабатывать миллионы запросов в секунду. Объемы многоканальной оперативной памяти могут превышать терабайт на сервер, число современных процессорных ядер может исчисляться сотнями, а объемы и скорость работы накопителей на базе энергонезависимой памяти последнего поколения на 1—2 порядка превышают скорость работы традиционных магнитных дисков. Если бы сегодня уделялось столь же серьезное внимание вопросам оптимизации программ, как 30 лет назад (когда основными языками программирования были язык ассемблера и Фортран), то производительности нескольких серверов было бы достаточно, чтобы обслужить население небольшой страны.

Однако технологии программирования, в настоящее время ориентированные в основном на удобство разработки, редко оказываются «удобными» для компьютеров. Производительность приложений, реализованных на базе дружественных по отношению к разработчику web-фреймворков и на популярных интерпретируемых языках, зачастую оказывается на 2-3 порядка ниже физических возможностей аппаратуры.

Одними из первых с такой ситуацией столкнулись (и не захотели мириться) такие интернет-гиганты, как Google и Facebook[1] [1, 2]. Это произошло на стадии расширения спектра предоставляемых ими услуг за пределы начального, тщательно оптимизированного высококвалифицированными программистами набора сервисов. Переходы на фреймворки и языки высокого уровня заметно ускоряли разработку, но не менее заметно снижали производительность разработанных сервисов. Компании начали проводить исследования, направленные на ускорение web-приложений, реализованных на базе фреймворков [3, 4]. Такие исследования активно продолжаются и в настоящем, что способствует в том числе улучшению высокоуровневых средств программирования.

[1] Данная социальная сеть запрещена в России.

Известные подходы к ускорению web-сервисов

Значительную часть традиционных информационных систем представляют собой надстройки над реляционными СУБД, служащими средством хранения и декомпозиции (нормализации) первичных данных. Эти системы обеспечивают структурированную выборку данных, совмещенную с элементами аналитической обработки. После нескольких циклов вида «запрос к СУБД — вычисления» формируется основа для графического представления, визуализируемая браузером или специально разработанным клиентским приложением в соответствии с принятым дизайном.

Низкая производительность типовых web-сервисов на базе фреймворков является следствием нескольких причин. Отсутствие универсальных рецептов для ускорения произвольных программ не позволяет решить задачу «в общем виде». Меняющиеся при этом требования к системе неизбежно влекут за собой частые изменения в ее программном коде, что делает классический, основательный подход к оптимизации программ малоперспективным занятием. В этих условиях единственной основой для оптимизации являются те свойства программы, которые не будут меняться по мере разработки ее новых версий.

Характерной чертой широкого класса информационных систем является значительное преобладание выборки и добавления новой информации над модификацией ранее введенных данных. При этом отмеченное свойство сохраняется в процессе эволюции таких систем, что может служить основой для долговременной стратегии оптимизации. Программы с неизменными входными данными изучают в теории функционального программирования. Они хорошо распараллеливаются, а эффективным средством оптимизации вычислений является мемоизация функций [5].

Ускорение программ, работающих с редко изменяемыми первичными данными, требует дополнительных механизмов. При изменении первичных данных необходимо либо инвалидировать, либо пересчитывать закешированные результаты, прямо или косвенно зависящие от проведенных изменений. Для простоты иногда прибегают к полному сбросу кеша. Однако оказывается, что намного эффективнее сбрасывать те и только те закешированные результаты, которые затронуты изменениями. Классическим механизмом здесь являются списки зависимостей, которые позволяют проводить минимально возможную адресную инвалидацию закешированных результатов.

Разработанная модель ускорителя web-сервисов

Как известно из теории функционального программирования, кешировать имеет смысл прежде всего компактные (по занимаемой памяти) результаты вычисления ресурсоемких функций. Это наводит на мысль о том, что программист, зная специфику информационной системы, может указать такие функции в коде при помощи аннотаций.

В разработанной модели используются динамическая схема формирования графа зависимостей и его хранение в реляционной СУБД. СУБД при этом настраивается таким образом, что изменение любых первичных данных приводит к лавинообразному удалению соответствующего транзитивно-замкнутого по отношению зависимости подграфа вместе с ассоциированными ячейками таблиц, хранящих закешированные результаты.

Модельная задача

В качестве модельной задачи для ускорения была выбрана генерация упорядоченного по годам списка публикаций научных работников по данным в информационно-аналитической системе «ИСТИНА» [6, 7].

Упрощенная модель данных, соответствующая данной задаче, может быть представлена при помощи библиотеки PonyORM в следующем виде:

```
class Man(db.Entity):
   name = Required(str)
   auths = Set('Auth')

class Journal(db.Entity):
   name = Required(str)
   pubs = Set('Article')

class Article(db.Entity):
   title = Required(str)
   year = Required(int)
   journal = Required(Journal)
   auths = Set('Auth')

class Auth(db.Entity):
   man = Required(Man)
   article = Required(Article)
   order = Required(int)
   PrimaryKey(article,order)
```

Модель `Article` описывает публикацию в журнале, который реализуется моделью `Journal`. Модель `Man` описывает отдельных авторов (пользователей), а модель `Auth` описывает принадлежность автора к конкретной статье. На рисунке показана схема БД, соответствующая данным моделям. Как видно из кода модели, публикации находятся с авторами в отношении многие-ко-многим. Кроме того, они имеют ссылку на журнал, в котором были опубликованы.

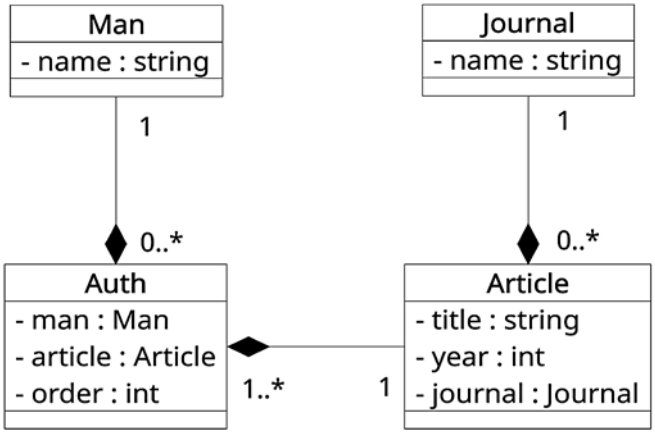

Схема БД

Для формирования текстового (в реальной системе — JSON-подобного) представления для отдельной публикации и для списка публикаций конкретного автора нужно выполнить запросы, используя вспомогательные классы для доступа к СУБД, генерируемые подсистемой ORM (*Object-Relational Mapping*). Для указания того, что результаты работы этих функций следует закешировать, достаточно пометить их декоратором[1] @memoize, код которого дан в разделе «Структура ускорителя». Приведенный ниже код реализует представление данных об отдельной публикации и списке публикаций автора с кешированием результатов.

```
@memoize
def article(a):
    authors=','.join(map(lambda u: u.man.name,
                         a.auths.sort_by(lambda u: u.order)))
    return str(a.year)+''+a.title+' in '+a.journal.name+
           ' by '+authors

@memoize
def man_pubs(m):
    return '; '.join(map(lambda u: article(u.article), m.auths))
```

Пример представления публикации: 2014 Среда моделирования для исследования средств обеспечения информационной безопасности в Grid и Cloud-системах in Программная инженерия by Васенин В. А., Роганов В. А., Зензинов А. А.

Далее рассмотрена схема работы ускорителя в целом и реализация функции memoize в частности.

[1] Декоратор в языке Python — это функция высшего порядка, модифицирующая те определения, перед которыми она помещена.

Структура ускорителя

Для хранения графа зависимостей между промежуточными закешированными результатами используются две отдельные таблицы в БД: таблица узлов `Node` и таблица ребер `Use` соответственно:

```
class Node(db.Entity):
   id = PrimaryKey(str)
   val = Required(Json)
   ins = Set('Use')
   outs = Set('Use')

class Use(db.Entity):
   user = Required(Node,reverse = 'ins')
   uses = Required(Node,reverse = 'outs')
   PrimaryKey(user,uses)
```

Процессу инвалидации кешей при таком способе соответствует удаление всех зависимых от модифицированных первичных данных вершин и ребер графа. Такая операция выполняется силами самой СУБД, так как в данном случае ORM создает таблицы с условием целостности по связям, поддерживаемым автоматическим (каскадным) удалением оставшихся без вершин ребер.

Кроме этого определяется триггер, который удаляет все вершины, зависящие от удаляемых, а также триггер, который вызывают специальную функцию, инвалидирующую внешний кеш, хранящий результаты в файловой системе на быстром NVMe (Non-volatile memory)-носителе:

```
@db _ session
def add _ triggers():
    db.execute("""
    CREATE TRIGGER Use _ del AFTER DELETE ON Use
        BEGIN
            DELETE FROM Node WHERE id == OLD.user;
        END
    """)
    db.execute("""
    CREATE TRIGGER Node _ del AFTER DELETE ON Node
        BEGIN
            SELECT inv _ ext(OLD.id);
        END
    """)
add _ triggers()
```

Функция `inv _ ext(id)`, вызываемая триггером внутри транзакции СУБД, выясняет, в каком именно кеше хранится результат при помощи поиска префикса ключа в хеш-таблице, динамически заполняемой при вычислении мемоизуемых функций:

```
def mpath(key):
    """Функция получения пути к закешированному
       значению"""
    return '/tmp/cache/'+key

def mkey(pfx,obj):
    """Функция получения ключа для кешируемого
       объекта"""
    return pfx+'-'+str(obj.id)

def mpfx(key):
    """Функция получения префикса из ключа
       кешируемого объекта"""
    return key.split('-')[0]

ext _ pfx = {}

def inv _ ext(key):
    """Функция инвалидации внешнего кеша"""
    if ext _ pfx.get(mpfx(key)):
       try:
           os.unlink(mpath(key)) # удалить
           файл с закешированным результатом
           debug('removed:',key)
       except:
           debug('not fs-cached:',key)
```

Таким образом, если функция `inv _ ext` опознает ключ промежуточного результата как внешний, она пытается удалить соответствующий файл, в котором этот результат мог быть сохранен ранее. По завершении транзакции из внешнего и внутреннего кешей будут удалены все данные, утратившие актуальность после модификации БД.

Ниже приведен упрощенный код декоратора memoize:

```
par = None

def memoize(f):
    def mf(obj):
    v = None;
    def fv():
        nonlocal v
        if not v:
            global par; saved = par
            par = n; v = f(obj); par = saved
            debug(' * * * * * computed:',me,' = ',v)
        return v
```

```
    me = mkey(f. __ name __ ,obj) #построение ключа
    path = mpath(me)
    n = Node.select(lambda n: n.id == me).first()
    if not n: # если нет соответствующего узла, то вычисляем его
        n = Node(id = me,val = '?')
        n.val = fv() if par else '@' + path
    if par: # если вызов верхнего уровня
        if not Use.select(lambda u: u.user == par and u.uses == n):
           Use(user = par,uses = n) # добавляем связь с родительским узлом
        return n.val
    else:
        n.val = '@' + path
        global ext _ pfx
        ext _ pfx[f. __ name __ ] = True
    if not os.path.exists(path):
        with open(path + ".tmp",'w') as file:
           file.write(fv()) # сохранение кеша в файл
        os.rename(path + ".tmp",path)
        debug('written:',path)
    with open(path,'r') as file:
        return file.read()
return mf
```

На первом шаге здесь из имени функции-аргумента `f` и идентификатора объекта первичных данных `obj` строится ключ. Если вызов является вызовом верхнего уровня (не имеет родительского вызова, что соответствует условию `par==None`), то результаты кешируются в файловой системе, чтобы быть непосредственно доступными для считывания http-фронтендом. В противном случае результат кешируется в таблице БД Node, а между текущим и родительским вызовами мемоизованных функций устанавливается отношение зависимости, информация о котором сохраняется в таблице Use. При «промахе» внешнего кеша http-фронтенд обращается к микросервису, который вычисляет список публикаций, попутно сохраняя вычисленный результат во внешней кеш-памяти:

```
from bottle import get,post,run
@get('/pubs/<id>')
@db _ session
def srv _ pubs(id):
    m = Man.select(lambda m: m.id == id).first()
    return man _ pubs(m)
@post('/inv')
def srv _ inv():
    inv();
    return "invalidated\n"
run(host = 'localhost', port=8080, quiet=True)
```

Далее приведен упрощенный код самого простого http-фронтенда, реализованного с использованием высокопроизводительного web-фреймворка Sanic, отображающий список публикаций заданного автора по его идентификатору.

```
from sanic import Sanic
from sanic.response import text
from urllib import request
app = Sanic('BenchApp')

@app.get('/pubs/<id>')

def benchmark(rq,id):
    try:
        with open('/tmp/cache/man _ pubs-'
             +id,"r") as f:
           return text(f.read())
    except:
        with request.urlopen('http://
             localhost:8080/pubs/'+id) as u:
           return text(u.read().decode())
```

Кроме описанной выше логики мемоизации для корректной работы ускорителя требуется задать еще несколько триггеров, которые будут удалять соответствующие вершины графа `Node` при модификациях характеристик журнала, атрибутов автора, а также при добавлении, удалении и модификации статей. Такие триггеры можно генерировать автоматически, используя схему БД с небольшими дополнительными аннотациями.

Достигнутая производительность на модельной задаче

Для измерения производительности работы ускорителя на модельной задаче использовался стенд с процессором Intel Xeon E5-2620 v2 @ 2.10GHz и 64 ГБ оперативной памяти.

При отсутствии изменения данных в БД производительность составила 30 000 запросов в секунду. При редактировании статьи с расчетной

частотой 2 раза в минуту средняя скорость работы сервиса заметно не менялась.

Критическая частота модификации данных, при которой средняя скорость обработки запросов снижалась в 2 раза (до 15 000 запросов в секунду), составила 20 запросов в секунду.

Различные способы оптимизации на уровне СУБД

Каскадный сброс графа зависимостей хорош тем, что гарантирует консистентность внутренних данных. Однако при некоторых способах реализации сброса можно столкнуться с ограничениями на число рекурсивно вызываемых триггеров. Поэтому реализация каскадной инвалидации силами самой СУБД требует определенной аккуратности.

Для первичных данных и зависимых от них закешированных результатов функций целесообразно использовать разные физические носители. Такие СУБД, как PostgreSQL позволяют указать, на каком носителе следует физически располагать те или иные таблицы. При этом первичные данные можно хранить в надежном RAID-хранилище, а внутренний кеш — на быстрых NVMe-накопителях. При выходе из строя последних достаточно заменить их на исправные накопители и «прогреть» кеши.

Скорость извлечения данных при операциях типа JOIN можно существенно ускорить, если для мемоизации задействовать материализованные представления. PostgreSQL очень быстро создает материализованные представления, которые также можно хранить на быстрых NVMe-накопителях. Поскольку полная перестройка материализованных представлений занимает заметное время, для их эффективного использования требуются механизмы типа «черный список» для тех их фрагментов, которые утратили актуальность после модификации первичных данных.

Горизонтальное масштабирование и параллельная актуализация содержимого кешей

Для обслуживания миллионов одновременно работающих пользователей и обеспечения отказоустойчивости необходимо использование нескольких серверов для фронтенда и СУБД, а также организация параллельных вычислений для кешируемых функций приложения.

Внешний кеш при этом распределяется по внешним, коммуникационным серверам фронтенда, а его инвалидация и ревалидация инициируются внутренними серверами с запущенными на них процессами СУБД и кодами мемоизуемых функций.

В целях достижения горизонтальной масштабируемости и расширения сферы применения средств долговременного кеширования авторами была разработана формальная модель и реализован программный прототип A-машины — асинхронного обобщения графовой G-машины, известной из теории функционального программирования.

A-машина проводит вычисления при помощи пула процессов, каждый из которых ведет запись зависимостей для своего сегмента графа. При наличии пересечения сегментов осуществляется автоматическая дедупликация (избежание повторного вычисления одних и тех же функций на идентичных значениях). Также при помощи A-машины может быть реализован режим отказоустойчивых вычислений, что актуально при использовании множества серверов.

Упреждающие вычисления

Поскольку производительность рассматриваемой системы в случаях с наполненным и пустым внешним кешем отличаются примерно на 2 порядка, то при «холодном» старте (т. е. при незаполненном кеше, например, после модернизации ПО) система может не справиться с расчетным потоком внешних запросов.

Для решения этой задачи могут использоваться упреждающие вычисления, которые за несколько десятков минут работы системы при ее старте заполнят наиболее часто запрашиваемые ячейки внешних таблиц, хранящих закешированные результаты. Информация о том, какие запросы стоит заранее выполнить, доступна во внутренних таблицах, хранящих закешированные результаты, даже после полного сброса внешнего кеша.

Заключение

В результате экспериментальных исследований показана возможность значительного повышения производительности информационных систем, работающих с большими, но редко модифицируемыми данными при помощи техники многоуровневого кеширования с динамическим трекингом зависимостей между вызовами мемоизуемых функций.

Комбинированная схема мемоизации позволяет динамически определять вызовы мемоизованных функций верхнего уровня и сохранять их актуальные результаты непосредственно в быстрой вторичной памяти, откуда фронтенд может рассылать их клиентам при помощи системного вызова `sendfile()`.

Разработанная и описанная методика предоставляет возможность использовать менее производи-

тельные и более энергоэффективные вычислительные узлы, в том числе на базе процессоров, создаваемых в России по программе импортозамещения.

В настоящее время авторы ведут работу по реализации описанных методов ускорения для популярного web-фреймворка Django [8] в рамках задачи, схожей с описанной в разделе «Модельная задача», а также для ускорения большой информационно-аналитической системы «ИСТИНА». На модельной задаче удалось достичь десятикратного ускорения.

Список литературы

1. **Gupta P., Zeldovich N., Madden S.** A Trigger-Based Middleware Cache for ORMs // Middleware 2011 / Eds F. Kon, A. M. Kermarrec, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. 2011. Vol. 7049. P. 329—349. DOI: 10.1007/978-3-642-25821-3_17.
2. **Bornhövd C., Altinel M., Mohan C.** et al. Adaptive Database Caching with DBCache // IEEE Data Eng. Bull. 2004. Vol. 27, No. 2. P. 11—18.
3. **Система** кеширования для Django ORM CachaLot. URL: https://django-cachalot.readthedocs.io/en/latest/index.html
4. **Система** кеширования для Django CacheOps. URL: https://github.com/Suor/django-cacheops
5. **Роганов В. А.** Мемоизация, инкрементальные и упреждающие вычисления в контексте повышения производительности информационных систем, работающих с большими данными // Ломоносовские чтения. Научная конференция. Секция механики. 20—26 апреля 2021 г. Тезисы докладов. М.: Изд-во Московского университета, 2021. С. 186—187.
6. **Интеллектуальная** система тематического исследования научно-технической информации (ИСТИНА) / Под. ред. В. А. Садовничего, С. А. Афонина, А. В. Бахтина и др. М.: Изд-во Московского университета, 2014. 262 с.
7. **Садовничий В. А., Васенин В. А.** Интеллектуальная система тематического исследования наукометрических данных: предпосылки создания и методология разработки. Часть 1 // Программная инженерия. 2018. Том 9, № 2. С. 51—58. DOI: 10.17587/prin.9.51-58.
8. **Зензинов А. А.** Использование А-машины в задаче автоматизации импорта и верификации данных в наукометрической информационной системе // Ломоносовские чтения. Научная конференция. Секция механики. 18—22 апреля 2022 г. Тезисы докладов. М.: Изд-во Московского университета, 2022. С. 81—82.

Increasing the Performance of Information Services in Systems Focused with Large, Rarely Modified Data

V. A. Vasenin, Professor, vasenin@msu.ru, **A. A. Zenzinov,** Junior Researcher,
V. A. Roganov, Senior Researcher,
Lomonosov Moscow State University, Moscow, 119991, Russian Federation

Corresponding author:

Vladimir A. Roganov, Senior Researcher,
Lomonosov Moscow State University, Moscow, 119991, Russian Federation
E-mail: radug-a@ya.ru

Received on December 10, 2022
Accepted on December 21, 2022

The article presents the results of the study, development and preliminary testing of the advanced mechanism of computing memoization oriented to information systems that work with large volumes of rarely modified primary data. The productivity of the information services of such systems can be significantly increased using the provided mechanisms for accurate work cache memory, when the results of those and only those calculations that affect the changes made by primary data are disabled.

Keywords: *high-performance services, functional programming, computing memoization, dependency tracking*

For citation:

Vasenin V. A., Zenzinov A. A., Roganov V. A. Increasing the Performance of Information Services in Systems Focused with Large, Rarely Modified Data, *Programmnaya Ingeneria,* 2023, vol. 14, no. 2, pp. 55—61. DOI: 10.17587/prin.14.55-61 (in Russian).

References

1. **Gupta P., Zeldovich N., Madden S.** A Trigger-Based Middleware Cache for ORMs, *Middleware 2011,* Eds F. Kon, A. M. Kermarrec, *Lecture Notes in Computer Science,* Springer, Berlin, Heidelberg, 2011, vol. 7049, pp. 329—349. DOI: 10.1007/978-3-642-25821-3_17.
2. **Bornhövd C., Altinel M., Mohan C.** et al. Adaptive Database Caching with DBCache, *IEEE Data Eng. Bull.*, 2004, vol. 27, no. 2, pp. 11—18.
3. **Django** CachaLot. Django ORM caching system, available at: https://django-cachalot.readthedocs.io/en/latest/index.html
4. **Django** CacheOps caching system, available at: https://github.com/Suor/django-cacheops
5. **Roganov V. A.** Memoization, Incremental and Predictive Computing in the Context of Improving the Performance of Information Systems Working with Big Data), *Lomonosovskie chtenija. Nauchnaja konferencija. Sekcija mehaniki*, 20—26 April 2021, Tezisy dokladov, Moscow, Izd-vo Moskovskogo universiteta, 2021, pp. 186—187 (in Russian).
6. **Intellektual'naja** sistema tematicheskogo issledovanija nauchno-tehnicheskoj informacii (ISTINA) (Intellectual System of Thematic Investigation of Scientometrical Data) / Eds V. A. Sadovnichij, S. A. Afonin, A. V. Bahtin et al. Moscow, Izd-vo Moskovskogo universiteta, 2014, 262 p. (in Russian).
7. **Sadovnichij V. A., Vasenin V. A.** Intellectual System of Thematic Investigation of Scientometrical Data: Background of Creation and Methodology of Development Programmnaya Ingeneria. Part 1, *Programmnaya Ingeneria,* 2018, vol. 9, no. 2, pp. 51—58. DOI: 10.17587/prin.9.51-58 (in Russian).
8. **Zenzinov A. A.** Using the A-machine in the task of automating the data import and verification in a scientometric information system, *Lomonosovskie chtenija. Nauchnaja konferencija. Sekcija mehaniki,* 18—22 April 2022, Tezisy dokladov, Moscow, Izd-vo Moskovskogo universiteta, 2022, pp. 81—82 (in Russian).

УДК 621.382
DOI: 10.17587/prin.14.62-68

П. Н. Бибило, д-р техн. наук, проф., зав. лаб., bibilo@newman.bas-net.by,
Объединенный институт проблем информатики Национальной академии наук Беларуси, Минск

Аппаратная реализация в FPGA операционных устройств с пониженным энергопотреблением

Поступила в редакцию 15.12.2022
Принята к публикации 22.12.2022

Описаны результаты экспериментов по аппаратной (схемной) реализации в составе FPGA различных VHDL-моделей операционных устройств, ориентированных на сокращение энергопотребления. Операционные устройства называют также конечными автоматами с трактами данных. Установлено, что наиболее эффективная для заказных СБИС VHDL-модель, основанная на clock gating, не является реализуемой для FPGA. Эффективными для FPGA являются VHDL-модели, основанные на обнулении неиспользуемых операндов либо сохранении их значений в дополнительных регистрах памяти. Статья является продолжением статьи автора [1], в которой подробно описаны исследуемые модели и приведены результаты экспериментов по схемной реализации тех же операционных устройств в составе заказных КМОП СБИС.

Ключевые слова: *цифровое устройство, конечный автомат с трактом данных, синтез логической схемы, VHDL, система проектирования Vivado, FPGA*

Для цитирования:

Бибило П. Н. Аппаратная реализация В FPGA операционных устройств с пониженным энергопотреблением // Программная инженерия. 2023. Том 14, № 2. С. 62—68. DOI: 10.17587/prin.14.62-68.

Введение

Среди программируемых логических интегральных схем (ПЛИС) центральное место занимают FPGA (FPGA — *Field-Programmable Gate Array* — программируемая пользователем вентильная матрица), которые имеют значительные преимущества перед другими ПЛИС как по техническим характеристикам, так и по удобству их проектирования с помощью свободно распространяемых САПР, обеспечивающих полный цикл проектирования: от моделирования исходных алгоритмических описаний на языках Verilog и VHDL (*Very high speed integrated circuits Hardware Description Language* — язык описания аппаратуры сверхскоростных интегральных схем) до получения файлов конфигураций FPGA. В новых системах проектирования микросхем FPGA, таких как система Vivado [2], в качестве языков для описания проектов цифровых систем используются также языки программирования C, C++ и SystemC. Функциональные возможности FPGA постоянно совершенствуются. Расширение функциональных возможностей FPGA обусловлено тем, что они усложняются: увеличивается число CLB (*Configurable Logic Block* — конфигурируемый логический блок), в состав FPGA включается большое число макроблоков DSP (*Digital Signal Processor* — цифровой сигнальный процессор), умножителей и блоков памяти [3].

Наряду с функциональной верификацией важнейшей проблемой при автоматизированном проектировании является снижение энергопотребления заказных цифровых КМОП СБИС (сверхбольших интегральных схем) и систем на кристалле. Проблема снижения энергопотребления СБИС решается практически на всех этапах проектирования — от алгоритмического до топологического [4]. Для сложных кристаллов заказных СБИС управление энергопотреблением выполняется специальной подсистемой на основе формата UPF [5]. В случае кристаллов FPGA, имеющих фиксированную архитектуру, проектная ситуация другая: уменьшать энергопотребление можно, меняя ис-

ходное алгоритмическое описание реализуемого устройства и изменяя опции синтеза, которые управляют преобразованием («вложением») исходного описания проекта в структуру FGPA.

Операционные цифровые устройства давно нашли широкое применение в практике проектирования и развиваются в настоящее время преимущественно с использованием для их описания HDL-моделей. Операционные устройства называют конечными автоматами с каналом данных (*Finite State Machine* (FSM) *with datapath* (FSM-D)) [6] либо с трактом данных [7]. Как утверждается в работе [8], важность такой модели цифровой схемы обуславливается еще и тем, что модели конечных автоматов с трактами данных используются при преобразовании программ на языке C в синтезируемое описание на языках описания аппаратуры. Создаваемое по C-программе синтезируемое RTL-описание представляет собой конечный автомат (т. е. схему, выполненную в синхронном стиле, что предпочтительно для современных FPGA), управляющий потоками вычислений. Результаты зависят от проектных ограничений (*constraints*), а также, причем в большей степени, от директив компилятора.

Одним из эффективных подходов к снижению динамического энергопотребления не только операционных устройств, но и других типов цифровых устройств является создание такого алгоритмического описания VHDL-проекта, в котором предусматривается отключение тех блоков, функционирование которых не требуется в одном либо нескольких (многих) тактах функционирования синхронной схемы.

Данная статья является продолжением статьи [1], в которой были рассмотрены и экспериментально исследованы способы алгоритмического VHDL-описания операционных устройств, ориентированного на снижение динамического энергопотребления, связанного с переключениями сигналов, при реализации такого класса устройств в составе заказных КМОП СБИС. В работе [1] подробно описаны VHDL-модели и приведены результаты экспериментов по схемной реализации операционных устройств в составе заказных СБИС. В данной статье для тех же примеров операционных устройств и тех же VHDL-моделей, что и в работе [1], проведена их схемная реализация в FPGA. Сравнение схем проводится по таким параметрам, как число программируемых элементов LUT, число триггеров, число блоков DSP, задержка схемы, потребляемая мощность (*Total Power*). Показано, что изменение алгоритмических описаний, приводящих к некоторому увеличению аппаратной сложности структур FPGA, приводит к снижению потребляемой мощности. Это позволяет проектировщику выбирать подходящий способ алгоритмического описания операционного устройства, исходя из требований быстродействия, аппаратной сложности и энергопотребления.

1. Пример операционного устройства

Подробное описание данного примера дано в работе [1]. Цифровое операционное устройство, пример которого приведен на рис. 1, состоит из композиции управляющего (FSM) и операционного (ALU) блоков, на входе схемы устройства имеется регистр (REG) синхронизируемых триггеров (элементов памяти). Штриховые связи на рис. 1 показывают, что не для всех далее рассматриваемых способов алгоритмического описания операционных устройств данные связи будут нужны.

Управляющий блок FSM (конечный автомат) будем называть также управляющим автоматом. В рассматриваемом примере цифрового устройства данный автомат задан графом переходов *G* (рис. 2). Функции переходов даны в табл. 1. Двоичные входные векторы (порты) *A, B* назовем *операндами* операционного блока ALU. В экспериментах число *n* разрядов каждого из операндов *A, B* равно 16 либо 32. Функции ALU заданы в табл. 2. Функции ALU2 — второго варианта операционного блока, содержащего только арифметические VHDL-команды умножения (*) и вычитания (−), приведены в табл. 3.

Функционирование VHDL-модели и реализующей ее синхронной логической схемы осуществляется по тактам. Смена состояния управляющего автомата выполняется по переднему фронту синхросигнала *clk*. В текущем такте вырабатывается

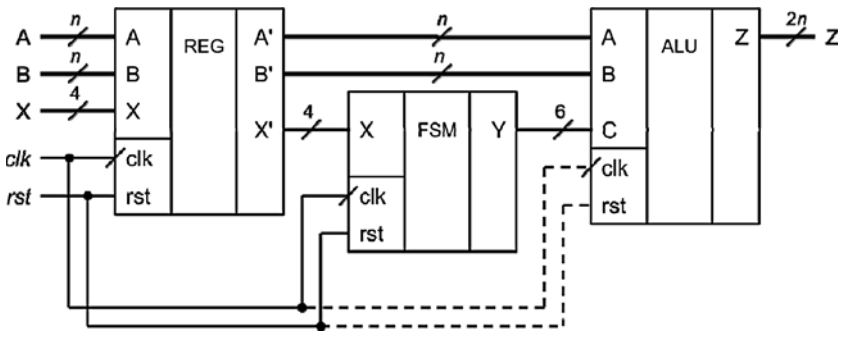

Рис. 1. Операционное устройство — композиция блоков: регистра REG элементов памяти, управляющего автомата FSM и арифметико-логического устройства ALU

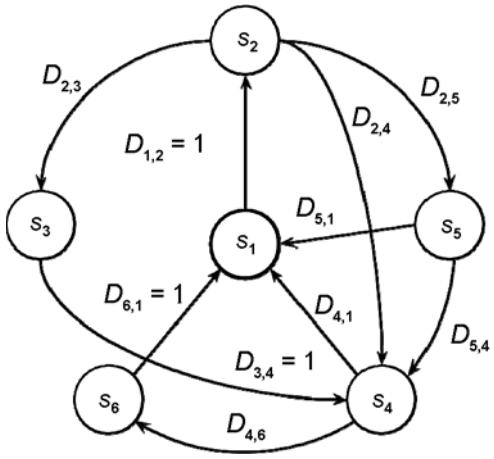

Рис. 2. Граф переходов *G* управляющего автомата FSM

Таблица 1

Таблица переходов управляющего автомата FSM

s_i	s_j	Условия перехода	Признак операции y_i
s_1	s_2	$D_{1,2} = 1$	y_2
s_2	s_3	$D_{2,3} = x_1\bar{x}_2\bar{x}_3 \vee x_1x_2$	y_3
	s_4	$D_{2,4} = x_1\bar{x}_2x_3$	y_4
	s_5	$D_{2,5} = \bar{x}_1$	y_5
s_3	s_4	$D_{3,4} = 1$	y_4
s_4	s_1	$D_{4,1} = \bar{x}_2$	y_1
	s_6	$D_{4,6} = x_2$	y_6
s_5	s_1	$D_{5,1} = \bar{x}_1x_4$	y_1
	s_4	$D_{5,4} = \bar{x}_4 \vee x_1x_4$	y_4
s_6	s_1	$D_{6,1} = 1$	y_1

Таблица 2

Операции ALU

y_i	VHDL-операция	Тип операции
y_1	$Z = A$ and B	Логическая
y_2	$Z = A$ or B	
y_3	$Z = A$ xor B	
y_4	$Z = A$ xnor B	
y_5	$Z = A + B$	Арифметическая
y_6	$Z = A^* B$	

Таблица 3

Арифметические операции ALU2

y_i	Арифметическая VHDL-операция
y_1	$Z = A - B$
y_2	$Z = B - A$
y_3	$Z = 3^*A^*B$
y_4	$Z = 5^*A^*B$
y_5	$Z = A^*A$
y_6	$Z = B^*B$

признак операции и для операндов A, B блок ALU (либо ALU2) выполняет только одну из шести операций y_i.

2. VHDL-описания операционных устройств

Способ 1 — традиционный: каждая из операций выполняется в операционном блоке, однако на выход устройства подается лишь результат той операции, выполнение которой требуется в данном такте. В рассматриваемом примере операционного устройства ALU основные аппаратные затраты приходятся на схему, реализующую операцию арифметического умножения.

Способ 2 — обнуление операндов для каждой из невыполняемых операций в текущем такте функционирования устройства.

Способ 3 — установка единичных значений всех разрядов операндов для невыполняемых операций.

Способ 4 — сохранение значений операндов для невыполняемых операций. Заметим, что данный способ требует значительного числа дополнительных регистров, в которых сохраняются значения операндов.

Способ 5 — сохранение значений операндов только для подсхемы, реализующей операцию арифметического умножения, и обнуление для остальных операций. Это связано с тем, что нецелесообразно использовать способ 4 для простых логических операций, так как энергопотребление дополнительных регистров будет превышать энергопотребление схем, реализующих логические операции.

3. Эксперименты

В этом разделе описана организация экспериментов и приведены их результаты. Целью экспериментов являлось сравнение эффективности предложенных способов алгоритмического

VHDL-описания операционных устройств по критериям энергопотребления, задержки и занятым ресурсам (аппаратным затратам) микросхемы FPGA xa7a12tcsg325-2I семейства Artix-7 и микросхемы xc7k70tfbg676-3 семейства Kintex-7 [3]. Эксперименты выполнялись в системе Vivado (версия 2019.1), стратегия синтеза *Vivado Synthesis Defaults*, стратегия имплементации — *Vivado Implementation Defaults*. Если не использовались блоки DSP, то значение опции -max_dsp синтеза полагалось равным нулю. В эксперименте 3 для ALU2 сохранялось значение операции y_6 $Z = B*B$ (см. табл. 3).

Эксперимент 1. Схемная реализация VHDL-описаний для размерности $n = 16$ операндов и использовании ALU (см. табл. 2) с логико-арифметическими командами.

Эксперимент 2. Схемная реализация VHDL-описаний для размерности $n = 32$ операнда и использовании ALU (см. табл. 2) с логико-арифметическими командами.

Эксперимент 3. Схемная реализация VHDL-описаний для размерности $n = 32$ операнда и использовании ALU2 (см. табл. 3) с арифметическими командами.

Результаты экспериментов 1—3 приведены в табл. 4—6 соответственно. Способ 3 проигрывает способу 2 и в экспериментах 2, 3 он не исследовался. В табл. 4—6 использованы следующие обозначения: LUT — число программируемых элементов FPGA (LUT — *Look-Up Table* — таблица, реализующая логическую функцию); FF — число триггеров (элементов памяти); DSP — число макроблоков DSP; Delay (нс) — задержка схемы; Total Power — потребляемая мощность (Вт); Dinamic Power (%) — доля динамической мощности в Total Power. Динамическая мощность практически всегда составляет 99 % общей мощности. Символом * отмечены лучшие (меньшие) значения мощности и задержки.

Для способа 6, исследованного в работе [1] и называемого clock gating — «тактовый сигнал, проходящий через логический вентиль» [9], эффективного для снижения энергопотребления схем из библиотечных КМОП-элементов, было установлено, что задание в Vivado различных значений опции gated_clock_conversion [2, стр. 396] не позволяет получить правильно функционирующую логическую схему, что было проверено моделированием получаемых схем. Иначе говоря, синтез схемы по VHDL-модели, соответствующей способу 6, в системе Vivado выполнялся, однако результаты моделирования (реакции) синтезированной схемы в целом отличались от выходных реакций, получаемых при моделировании правильных исходных VHDL-описаний операционных устройств.

Таблица 4

Результаты эксперимента 1 для логико-арифметических команд ALU, $n = 16$

VHDL-описание	Использовался 1 блок DSP					Не использовались DSP				
	LUT	FF	Delay, нс	Total Power, Вт	Dinamic Power, Вт (%)	LUT	FF	Delay, нс	Total Power, Вт	Dinamic Power, Вт (%)
Artix-7, микросхема xa7a12tcsg325-2I										
Способ 1	96	39	12,331	6,309	6,235 (99)	399	39	*16,308	9,365	9,273 (99)
Способ 2	114	39	13,268	*1,082	1,021 (94)	446	39	17,185	1,942	1,879 (97)
Способ 3	112	39	14,009	4,363	4,296 (98)	446	39	17,488	5,270	5,200 (99)
Способ 4	114	205	*9,577	2,713	2,650 (98)	418	237	16,496	*1,481	1,419 (96)
Способ 5	114	71	13,685	1,840	1,804 (98)	449	71	18,697	1,582	1,521 (96)
Kintex-7, микросхема xc7k70tfbg676-3										
Способ 1	97	39	9,267	6,300	6,204 (98)	399	39	*10,928	9,247	9,141 (99)
Способ 2	114	39	9,972	*1,114	1,031 (93)	446	39	11,460	9,028	1,93 (96)
Способ 3	113	39	10,267	4,384	4,294 (98)	446	39	11,789	5,339	5,246 (99)
Способ 4	113	205	*7,630	2,725	2,638 (97)	418	237	10,393	*1,525	1,441 (93)
Способ 5	114	71	10,114	1,861	1,777 (95)	449	71	12,616	1,636	1,552 (95)

Таблица 5

Результаты эксперимента 2 для логико-арифметических команд ALU, $n = 32$

VHDL-описание	Использовались 4 блока DSP					Не использовались DSP				
	LUT	FF	Delay, нс	Total Power, Вт	Dinamic Power, Вт (%)	LUT	FF	Delay, нс	Total Power, Вт	Dinamic Power, Вт (%)
Artix-7, микросхема xa7a12tcsg325-2I Dinamic Power, Вт (%)										
Способ 1	224	71	*15,805	14,946	14,783 (99)	1 466	71	*22,755	30,357	30,092 (99)
Способ 2	274	71	18,715	*2,282	2,219 (97)	1 510	71	26,122	*5,521	5,449 (99)
Способ 4	313	263	18,137	7,188	7,110 (99)	1 720	263	28,185	8,170	8,086 (99)
Способ 5	281	135	16,861	5,703	5,631 (99)	1 688	135	30,730	6,294	6,220 (99)
Kintex-7, микросхема xc7k70tfbg676-3										
Способ 1	226	71	*12,116	14,960	14,827 (99)	1 466	71	*15,833	29,817	29,530 (99)
Способ 2	264	71	13,559	*2,273	2,187 (96)	1 510	71	18,429	*5,187	5,095 (98)
Способ 4	313	263	12,605	7,277	7,178 (99)	1 720	263	21,199	7,561	7,461 (99)
Способ 5	281	135	12,833	5,872	5,777 (98)	1 688	135	20,265	6,034	5,939 (98)

Таблица 6

Результаты эксперимента 3 для арифметических команд ALU2, $n = 32$

VHDL-описание	Использовались 16 блоков DSP					Не использовались DSP				
	LUT	FF	Delay, нс	Total Power, Вт	Dinamic Power, Вт (%)	LUT	FF	Delay, нс	Total Power, Вт	Dinamic Power, Вт (%)
Artix-7, микросхема xa7a12tcsg325-2I										
Способ 1	484	71	*18,555	46,218	45,954 (99)	5 249	105	*30,278	125,978	125,714 (99)
Способ 2	591	71	21,042	*10,741	10,637 (99)	5 522	73	30,893	24,675	24,410 (99)
Способ 4	617	391	20,936	22,099	21,835 (99)	5 786	391	37,463	35,759	35,494 (99)
Способ 5	600	103	19,066	11,855	11,739 (99)	5 611	103	33,609	*21,619	21,355 (99)
Kintex-7, микросхема xc7k70tfbg676-3										
Способ 1	484	71	14,107	46,208 #	45,492 (98)	5 249	105	22,190	125,432 #	124,402 (99)
Способ 2	586	71	*13,581	*10,763	10,652 (99)	5 522	73	*20,527	23,879	23,672 (99)
Способ 4	617	391	13,837	21,677	21,492 (99)	5 786	391	22,565	34,969	34,584 (99)
Способ 5	591	103	13,731	11,564	11,448 (99)	5 611	103	20,545	*21,475	21,293 (99)

Проведем анализ результатов экспериментов, представленных в табл. 4—6. Использование одного макроблока DSP сокращает аппаратные затраты примерно на 300 LUT, немного уменьшает задержки и потребляемую мощность. Традиционный способ 1 требует меньше аппаратных затрат (LUT и триггеров), имеет незначительно лучшие показатели по быстродействию по сравнению со способами 2—5, однако в 5—6 раз проигрывает лучшему из способов 2, 4, 5 по энергопотреблению. Способ 2 (обнуление неиспользуемых операндов) оказался наиболее эффективным, так как является простым в реализации, не требует добавления элементов памяти и имеет хорошие показатели по

сокращению энергопотребления. Изменение состава операций ALU (как это было проверено для случая ALU2) требует проведения соответствующих экспериментов для выбора лучшего способа описания. Если аппаратная сложность операций ALU примерно одинакова, то эффективным может оказаться способ 4 сохранения значений неиспользуемых операндов либо способ 5 сохранения значений операндов для одной или нескольких наиболее сложных операций. Это справедливо, когда сложность схемы для отдельной операции значительно превышает сложность добавляемых элементов памяти, в противном случае надо использовать способ 2 обнуления операндов. Выбор одной (либо нескольких) сложных и наиболее часто повторяющихся операций для способа 5 может быть сделан после проведения моделирования исходного VHDL-описания, составленного по способу 1. Эти выводы справедливы как для микросхемы семейства Artix-7, так и для микросхемы семейства Kintex-7.

В целом семейство Kintex-7 на рассмотренных примерах обеспечивает меньшие задержки схем в среднем на 30 % при использовании блоков DSP, и на 18 % при реализации проектов только на LUT (без блоков DSP). Что касается значения потребляемой мощности (Total Power), то данное семейство обеспечивает и меньшую потребляемую мощность. Следует заметить, что для ALU2, в котором все операции являются трудоемкими, способ 1 (традиционный) не позволяет реализовать операционное устройство вследствие превышений ограничения по мощности, что в табл. 6 помечено символом # — выдается сообщение “Junction temperature exceeded — Превышена температура соединения”.

Это говорит о том, что добиваться только правильной функциональности VHDL-описаний цифровых устройств при их реализации на FPGA может оказаться недостаточно, так как проект может оказаться нереализуемым на выбранной микросхеме FPGA по ограничениям потребляемой мощности. Особенно это касается проектов, в которых требуется одновременное (в одном такте) выполнение многих трудоемких арифметических команд умножения, вычисления остатка от деления и др. Поэтому предложенные в статье способы сокращения потребляемой мощности могут оказаться полезными при реализации проектов не только операционных устройств, но и других сложных цифровых проектов. Полезными могут оказаться для синтеза на FPGA и полученные сравнительные результаты схемных реализаций одних и тех же проектов операционных устройств на микросхемах семейств Artix-7 и Kintex-7.

Заключение

Способ 2 — обнуление операндов — является простым и эффективным способом сокращения потребляемой мощности при реализации операционных устройств в FPGA, хотя для нерегулярных схем операционных устройств, синтезируемых в базисе библиотечных элементов заказных КМОП СБИС, лучшим способом сокращения потребляемой мощности оказался [1] способ clock gating. Проведя экспериментальное сравнение различных способов для конкретного операционного устройства (либо другого цифрового устройства), проектировщик может выбрать подходящий (компромиссный) способ VHDL-описания, рассматривая полученные значения параметров энергопотребления, быстродействия и аппаратной сложности.

Список литературы

1. **Авдеев Н. А., Бибило П. Н.** Автоматизированное проектирование цифровых операционных устройств с пониженным энергопотреблением // Программная инженерия. 2021. Том 12, № 2. С. 63—73. DOI: 10.17587/prin.12.63-73.

2. **Тарасов И. Е.** ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования. М.: Горячая линия — Телеком, 2020. 538 с.

3. **Соловьев В. В.** Архитектуры ПЛИС фирмы Xilinx: FPGA и CPLD 7-й серии. М.: Горячая линия — Телеком, 2016. 392 с.

4. **Рабаи Ж. М., Чандракасан А., Николич Б.** Цифровые интегральные схемы. М.: Вильямс, 2007. 912 с.

5. **Шашков А. С.** Проектирование цифровых систем с пониженным энергопотреблением с применением технологии UPF-описания подсистемы питания // Информатика. 2015. № 3. С. 90—104.

6. **Schaumont P. R.** Finite State Machine with Datapath // A Practical Introduction to Hardware/Software Codesign. Springer, Boston, MA. 2010. P. 95—132.

7. **Томас Д.** Логическое проектирование и верификация систем на SystemVerilog / пер. с англ. А. А. Слинкина, А. С. Камкина, М. М. Чупилко; науч. ред. А. С. Камкин, М. М. Чупилко. М.: ДМК Пресс, 2019. 384 с.

8. **Тарасов И. Е.** Проектирование для ПЛИС Xilinx с применением языков высокого уровня в среде Vivado HLS // Компоненты и технологии. 2013. № 12. С. 40—48.

9. **Kaxiras S., Martonosi M.** Idle unit switching activity: clock gating. Architectural Techniques for Low Power. Morgan & Claypool Publishers, 2008. 207 p.

Hardware Implementation of Digital Operational Low Power Units in FPGA

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

Corresponding author:

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

Received on December 15, 2022
Accepted on December 22, 2022

The results of experiments on hardware implementation of various VHDL models of operating devices in FPGA oriented to reducing power consumption are described. Operating devices are also called finite state machines with data paths. It is established that the VHDL model based on clock gating, which is most effective for custom VLSI, can not be implemented in FPGA. Effective models for FPGA are VHDL models based on zeroing unused operands or storing their values in additional memory registers. After conducting an experimental comparison of various methods for a specific operating device (or other digital device), the designer can choose a suitable (compromise) method of VHDL description, considering the obtained values of the parameters of power consumption, performance and hardware complexity. The article is a direct continuation of the previous article [1], which describes in detail the models under study and presents the results of experiments on hardware implementation of the same operating devices as part of custom CMOS VLSI.

Keywords: *digital device, finite state machine with datapath, digital logic synthesis, power consumption, VHDL, Vivado, FPGA*

For citation:

Bibilo P. N. Hardware Implementation of Digital Operational Low Power Units in FPGA, *Programmnaya Ingeneria,* 2023, vol. 14, no. 2, pp. 62—68. DOI: 10.17587/prin.14.62-68 (in Russian).

References

1. **Avdeev N. A., Bibilo P. N.** Design of Digital Operational Units with Low Power Consumption, *Programmnaya Ingeneria,* 2021, vol. 12, no. 2, pp. 63—73. DOI: 10.17587/prin.12.63-73 (in Russian)..

2. **Tarasov I. E.** *XILINX FPGA. Hardware Description Languages VHDL and Verilog, CAD, Design Techniques,* Moscow, Goryachaya liniya — Telekom, 2020, 538 p. (in Russian).

3. **Solovyov V. V.** *XILINX FPGA Architectures: FPGA and CPLD 7-Series.* Moscow, Goryachaya liniya — Telekom, 2016, 392 p. (in Russian).

4. **Rabai M., Chandrakasan A., Nicolic B.** *Digital Integrated Circuits: A Design Perspective*. Moscow, Vil'jams, 2007, 912 p. (in Russian).

5. **Shashkov A. S.** Design of low-power electronic systems using UPF power intent specification technology, *Informatika*, 2015, no. 3, pp. 90—104 (in Russian).

6. **Schaumont P. R.** Finite State Machine with Datapath, *A Practical Introduction to Hardware/Software Codesign,* Springer, Boston, M. A., 2010, pp. 95—132.

7. **Tomas D.** *Logical design and verification using SystemVerilog,* Moscow, DMK Press, 2019, 384 p. (in Russian).

8. **Tarasov I. E.** Designing for Xilinx FPGAs using high-level languages in Vivado HLS environment, *Komponenty i tekhnologii,* 2013, no. 12, pp. 40—48 (in Russian).

9. **Kaxiras S., Martonosi M.** *Idle unit switching activity: clock gating, Architectural Techniques for Low Power*, Morgan & Claypool Publishers, 2008, 207 p.

УДК 004.021
DOI: 10.17587/prin.14.69-76

Р. В. Душкин, директор по науке и технологиям,
В. А. Лелекова, аналитик, lv@aiagency.ru,
К. Ю. Эйдемиллер, канд. геогр. наук, академический директор,
Агентство Искусственного Интеллекта, Москва

Система операций над ассоциативно-гетерархической памятью

Поступила в редакцию 01.09.2022
Принята к публикации 13.11.2022

Обработка текстов на естественном языке остается важной задачей для области разработки методов и средств искусственного интеллекта. С учетом авторского подхода к построению агентов искусственного интеллекта обработка естественного языка должна вестись на двух уровнях: на нижнем, при помощи методов восходящей парадигмы, и на верхнем, при помощи символьных методов нисходящей парадигмы. Авторами статьи уже был введен новый математический формализм, основанный на понятии гиперграфа — ассоциативно-гетерархическая память. Такая память должна упростить процесс обработки естественного языка с помощью новых технологий. Статья будет интересна разработчикам методов и средств искусственного интеллекта, математикам и специалистам в сфере обработки текстов на естественном языке.

Ключевые слова: *ассоциативно-гетерархическая память, обработка текстов на естественном языке, natural language processing, гиперграф, математический формализм, искусственный интеллект, символьный метод, абстрактные символы, система операций, ИИ-агент*

Для цитирования:

Душкин Р. В., Лелекова В. А., Эйдемиллер К. Ю. Система операций над ассоциативно-гетерархической памятью // Программная инженерия. 2023. Том 14, № 2. С. 69—76. DOI: 10.7587/prin.14.69-76.

Введение

Обработка естественного языка с применением методов и средств искусственного интеллекта (NLP, *Natural Language Processing*) является актуальной проблемой современности [1]. Конечная цель NLP — создание программных средств, способных обрабатывать и понимать естественные языки. Под естественным языком подразумевается язык, на котором говорят или пишут люди [2]. Для решения этой задачи применяется большое число методов искусственного интеллекта [3], основанных на двух парадигмах [4]: нисходящей и восходящей. Методами восходящей парадигмы строятся модели когнитивных процессов, которые базируются на больших объемах данных, и таким образом создается модель класса «черный ящик», точность которой можно довести до необходимого значения. Однако процессы принятия решений в такой модели сложно (или даже практически невозможно) интерпретировать с человеческой точки зрения. Методами же нисходящей парадигмы строятся модели, основанные на знаниях, которые представляют собой модели класса «белый ящик». Однако в таких моделях сложно поддерживать актуальность состояния при изменении структуры проблемной области, а само построение таких моделей — сложный и трудоемкий процесс. В кратком изложении их суть состоит в следующем. Агент искусственного интеллекта (ИИ-агент) — это полноценная кибернетическая машина, которая имеет систему управления, непрерывно получающую информацию с сенсорных систем агента и воздействующую на окружающую среду при помощи исполнительных устройств (или актуаторов). С учетом авторского подхода к построению ИИ-агентов [5] обработка естественного языка должна вестись на двух уровнях: на нижнем, при помощи методов восходящей парадигмы, и на верхнем, при помощи символьных методов нисходящей парадигмы.

Для решения задач обработки текстов на естественном языке существуют приложения во многих сферах, результаты которых по-разному выражены в медицинской [6, 7], химической и материаловедческой [8] областях, при проектировании процессов рабочего процесса [9] и т. п.

Авторами предложен новый математический формализм [10] — ассоциативно-гетерархическая память (АГ-память), функционирование которого основано как на бионических принципах, так и на достижениях обеих парадигм искусственного интеллекта. АГ-память представляет собой строгую математическую структуру, основанную на понятии гиперграфа [11].

Структура АГ-памяти описана с помощью следующего кортежа:

$$AG = <S, C, P, H, L>,$$

где S — множество абстрактных символов, решающее проблему привязки символов [12]; C — гиперсеть общих знаний; P — гиперсеть частных знаний; H — гиперсеть личной истории; L — множество ассоциативных связей между всеми объектами АГ-памяти. Множество L образуют шесть множеств связей элементов друг с другом. Взаимодействие этих множеств можно выразить в виде графа [13] (рис. 1).

Эта структура подробно описана в статье авторского коллектива [10]. В настоящей статье будет представлена система операций над АГ-памятью.

Анализ существующих моделей обработки естественного языка

Первый подход обработки текста на естественном языке на основе описания модели памяти в виде формальной грамматики на настоящее время рассматривается для высокоструктурированных (запрограммированных) языков, к которым естественные языки не относятся. При таком подходе не учитываются ни контекст, ни языковые особенности обрабатываемого текста. Второй подход — статистический метод построения

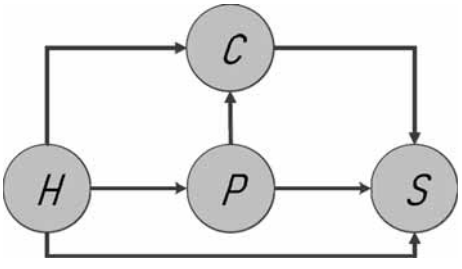

Рис. 1. Граф связей между объектами АГ-памяти

модели — показал свою ненадежность и сейчас практически не используется в силу того, что не «понимает» текст и просто считает разные элементы. Предпосылками создания новой авторской модели являются существенные недостатки уже существующих подходов. Они оба не подходят для обработки естественного языка.

Предложенный авторами формализм представляет собой конвергенцию нейросетевых моделей, являющихся расширением статистических моделей, также не способных интерпретировать результаты обработки текста, и семантической свертки, фактически предназначенной исключительно для информационного поиска документов. Тем самым авторская модель поддерживает лучшие черты этих двух моделей. Она не включает в АГ-память негативные черты (неинтерпретируемость нейросетевых моделей, высокая сложность обработки семантической свертки). Еще одним преимуществом АГ-памяти является наличие трех способов машинного вывода: семантического, логического и лингвистического. В АГ-памяти появляется возможность интерпретации благодаря свойству семантических цепей.

В первую очередь модель АГ-памяти ориентирована на текстовые данные, однако поддерживает использование аудиоданных, предварительно переведенных в текст. Причем, предположительно, АГ-память должна поддерживать любые естественные языки, поскольку в них есть трехчастная структура «субъект — предикат — объект». Будущие эксперименты с АГ-памятью будут проводиться в том числе в этой области, где будет рассматриваться применимость модели на разных языках.

Система операций над АГ-памятью

Система операций над элементами АГ-памяти расширяемая. Для конкретного ИИ-агента, кроме базовых операций в ее рамках могут быть реализованы дополнительные операции, облегчающие разработку системы ИИ-агента с учетом особенностей его когнитивной архитектуры.

Например, для когнитивного агента, работающего через чат-бота и представляющего собой интерфейс «вопрос-ответ» в какой-либо информационной системе, набор операций должен быть дополнен вызовами функций, которые вызываются в случае необходимости и содержатся в программном интерфейсе приложения соответствующей информационной системы. Разберем следующий случай: пользователь запрашивает у чат-бота отчет PNL (*Profit and Loss*) за прошедший месяц. Кроме операций, которые добавляют и удаляют элементы

АГ-памяти, система операций над АГ-памятью будет содержать в себе операцию для вызова функции генерации отчета. АГ-память генерирует параметры ответа, а именно то, какой будет отчет, как он будет выражен, как он будет оформлен, к какому времени относятся его результаты и т. д.

Другой пример — программно реализуемый интеллектуальный робот, т. е. ИИ-агент, функционирующий в кибернетическом теле робота. Для него дополнительными операциями над АГ-памятью будут вызовы моторных функций управления манипуляторами. В данном случае АГ-память является элементом системы управления роботом, его гибридной когнитивной архитектурой. Робот принимает решения на символьном уровне, а затем отправляет низкоуровневые моторные команды своим сервомеханизмам.

Основной принцип системы операций над элементами АГ-памяти заключается в возможности добавления и изменения объектов, входящих в состав АГ-памяти. Однако для удаления объектов специальных операций не предусмотрено. Поэтому в состав когнитивной архитектуры следует добавлять сборщика мусора для освобождения памяти от таких узлов гиперграфа, которые больше не связаны с другими узлами АГ-памяти. Это означает, что все веса всех связей, которые входят или исходят из таких узлов, получили значение 0. Механизм сборки мусора также может убирать из АГ-памяти связанные подграфы, не обусловленные хотя бы одной ассоциативной связью или гиперсвязью к абстрактному символу из множества S.

На рис. 2 показана часть когнитивной архитектуры ИИ-агента, включающего в свой состав АГ-память и механизм сборки мусора. Сенсорные потоки собираются сенсорными системами, чтобы затем перейти в центр мультисенсорной интеграции. Результатом его работы является формирование абстрактных символов внутреннего представления смысла, к которым привязываются множества иконических символов различных сенсорных модальностей, доступных для ИИ-агента. Центр мультисенсорной интеграции собирает выделенные и распознанные образы со всех сенсорных модальностей ИИ-агента и осуществляет построение интегрированного абстрактного символа, который привязан ко всем распознанным образам сенсорного восприятия. Таким образом, ИИ-агент решает проблему привязки символов [12]. Затем информация из центра мультисенсорной интеграции поступает в АГ-память и центр принятия решений. АГ-память обрабатывает поступившие данные с учетом работы сборщика мусора, удаляющего лишние и оборванные связи и гиперсвязи.

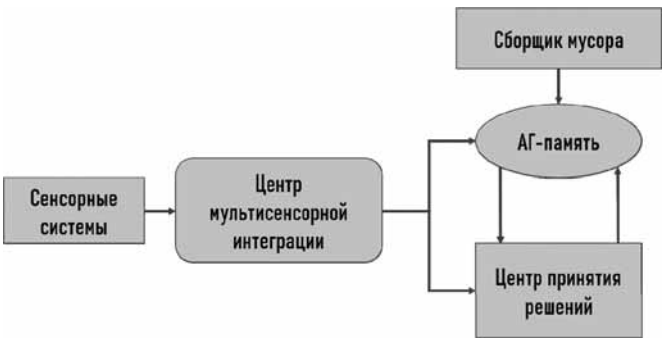

Рис. 2. Элементы когнитивной архитектуры ИИ-агента

Учитывая состав множеств, формирующих АГ-память, возможно вывести основные операции, которые можно воспроизвести при ее использовании.

Далее перечислим основные операции над АГ-памятью.

- Добавление нового абстрактного символа в состав множества S:

$$addAbstractSymbol\colon\ S \times s \to S.$$

***Вход*:** множество абстрактных символов S и новый символ s.

***Выход*:** новое состояние множества абстрактных символов S.

Следует учитывать, что новый символ в множестве изначально не имеет связей с какими-либо элементами АГ-памяти, а потому должен игнорироваться сборщиком мусора на время создания связей. Такое время первоначального существования добавленного символа является гиперпараметром АГ-памяти.

- Изменение абстрактного символа в составе множества S:

$$editAbstractSymbol\colon\ S \times UID \times s \to S.$$

***Вход*:** текущее множество абстрактных символов S, идентификатор имеющегося в множестве абстрактного символа UID и новый абстрактный символ s.

***Выход*:** новое состояние множества S, в котором абстрактный символ заменен на новый с заданным UID.

- Добавление нового элемента АГ-памяти в состав множества C, P или H:

$$addElement\colon\ (C|P|H) \times e \to (C|P|H)$$

***Вход*:** текущая часть АГ-памяти C, P или H и новый символ e.

***Выход*:** новое состояние части АГ-памяти *C*, *P* или *H*.

Новый элемент также добавляется в часть АГ-памяти без связей с любыми другими элементами в случае, если он сам не является гиперсвязью, а потому имеет время первоначального существования.

- Изменение элемента в составе части АГ-памяти *C*, *P* или *H*:

$$editElement\colon (C \mid P \mid H) \times UID \times e \to (C \mid P \mid H).$$

***Вход*:** текущая часть АГ-памяти *C*, *P* или *H*, уникальный идентификатор элемента *UID* и новый элемент *e*.

***Выход*:** новое состояние части АГ-памяти (*C*, *P* или *H*), где элемент с идентификатором *UID* заменен на новый элемент.

Эта операция также может быть использована через набор конкретизированных операций для каждой части и каждого типа элемента АГ-памяти.

- Операции добавления и изменения свойства или метасвойства элемента:

$$addProperty\colon (Pr \mid Mt) \times p \to (Pr \mid Mt),$$

$$editProperty\colon (Pr \mid Mt) \times UID.name \times p \to (Pr \mid Mt).$$

***Вход*:** множества свойств или метасвойств *Pr* или *Mt*, новое свойство *p*, наименование свойства *UID.name*.

***Выход*:** обновленные множества свойств или метасвойств *Pr* или *Mt*.

Эти операции также могут быть использованы с помощью конкретизированных параметров для каждого типа множества свойств или метасвойств.

- Операция добавления новой ассоциативной связи между элементами:

$$addLink\colon L \times l \to L.$$

***Вход*:** множество ассоциативных связей *L*, новая связь *l*.

***Выход*:** новое состояние множества *L*.

Утилитарные операции над АГ-памятью

Кроме операций создания и редактирования элементов АГ-памяти в составе перечня функциональных возможностей ИИ-агента, могут быть реализованы утилитарные операции по выборке и поиску элементов АГ-памяти. К таблице, систематизирующей все утилитарные операции над элементами АГ-памяти, необходимо добавить ряд следующих дополнений.

Под «пустым объектом», который возвращается операциями в случае, если конкретный объект не найден, подразумевается некоторый служебный объект соответствующего типа. Такой объект позволяет ИИ-агенту идентифицировать ситуацию, когда конкретный объект не найден.

Множество *X* в сигнатурах утилитарных операций означает множество элементов АГ-памяти.

Как видно из данных таблицы, 16 представленных утилитарных операций позволяют осуществлять функции выборки поиска и выборки конкретных типов элементов АГ-памяти для их использования в дальнейшем функционировании ИИ-агента. Для АГ-памяти может использоваться язык SQL для нахождения выборок из баз данных. Фактически, таблица представляет собой список утилитарных функций, которые являются аналогами языковых конструкций SQL по поиску и выборке конкретных элементов АГ-памяти.

Все операции из представленной таблицы со знаком «|» имеют альтернативные варианты использования. Они могут быть реализованы через набор конкретизированных операций для каждого типа соответствующего операнда.

Для конкретного ИИ-агента работа с АГ-памятью может быть реализована с помощью проблемно-ориентированного языка (DSL — *domain specific language*) [14] для исполнения запросов к АГ-памяти. Этот язык предназначен для упрощения работы всех компонентов когнитивной архитектуры ИИ-агента с его АГ-памятью, он должен включать в себя все приведенные в настоящем разделе операции над элементами АГ-памяти, а также может быть расширен любыми дополнительными операциями, которые упрощают поиск, выборку и работу с элементами АГ-памяти на любом уровне. Команды и синтаксические конструкции DSL конкретного ИИ-агента должны интерпретироваться в терминах, представленных в настоящей статье.

Например, АГ-память может быть использована воплощенными ИИ-агентами [15]. Под воплощенностью понимается работа ИИ-агентов в реальном окружении — для хранения личной истории (набора данных, которые уже обработал ИИ-агент) и опыта (набора данных, на которых уже обучался ИИ-агент) для быстрого сбора моделей объектов в окружающей их среде для более эффективного взаимодействия с ними. АГ-память может быть использована для обучения роботов моторным навыкам при помощи наблюдения за действиями аналогичной конструкции [16]. При этом АГ-память можно использовать с моделями механической конструкции себя и других подоб-

Перечень утилитарных операций выборки и поиска над элементами АГ-памяти

№ п/п	Сигнатура операции	Описание операции
1	*getAbstractSymbol: S UID s*	Операция получения абстрактного символа по его уникальному идентификатору. Получает на вход множество абстрактных символов и уникальный идентификатор, а возвращает абстрактный символ. Если абстрактный символ по уникальному идентификатору не найден, то операция возвращает пустой объект
2	*findAbstractSymbols: S {p} {s}*	Операция получения множества абстрактных символов по заданному критерию — наличию у абстрактного символа хотя бы одного привязанного сенсорного символа из заданного множества. Операция получает на вход множество абстрактных символов и множество сенсорных символов, а возвращает множество абстрактных символов, каждый элемент которого привязан хотя бы к одному из полученных сенсорных символов. Возвращаемое операцией множество абстрактных символов может быть пустым
3	*getSReference: X UID s**	Операция получения ссылки на абстрактный символ из заданного множества элементов АГ-памяти. Ссылка на абстрактный символ возвращается по ее уникальному идентификатору. Если для заданного уникального идентификатора нет ссылки на абстрактный символ, то операция возвращает пустой объект
4	*findSReferences: X UID {s*}*	Операция получения множества ссылок на абстрактный символ, заданный по своему уникальному идентификатору. Операция получает на вход уникальный идентификатор абстрактного символа, а возвращает множество ссылок на этот абстрактный символ из заданного множества элементов АГ-памяти. Возвращаемое операцией множество ссылок на заданный абстрактный символ может быть пустым
5	*getMReference: X UID m**	Операция получения ссылки на абстрактный символ второго порядка *m** из заданного множества элементов АГ-памяти. Ссылка на абстрактный символ второго порядка возвращается по ее уникальному идентификатору. Если для заданного уникального идентификатора нет ссылки на абстрактный символ второго порядка, то операция возвращает пустой объект
6	*findMReferences: X UID {m*}*	Операция получения множества ссылок на абстрактный символ второго порядка *m**, заданный по своему уникальному идентификатору. Операция получает на вход уникальный идентификатор абстрактного символа второго порядка, а возвращает множество ссылок на этот абстрактный символ второго порядка из заданного множества элементов АГ-памяти. Возвращаемое операцией множество ссылок на заданный абстрактный символ второго порядка может быть пустым
7	*getSymbol: X UID m*	Операция получения абстрактного символа второго порядка по его уникальному идентификатору в рамках заданного множества элементов АГ-памяти. Операция возвращает сам абстрактный символ второго порядка. Если для заданного уникального идентификатора не найдено соответствующего абстрактного символа второго порядка, то возвращается пустой объект
8	*findSymbols: X {p} {m}*	Операция для получения множества абстрактных символов второго порядка из заданного множества элементов АГ-памяти в соответствии с условием, заданным множеством свойств символа. Множество {*p*} представляет собой множество свойств, которые должны быть у абстрактного символа второго порядка, чтобы он был включен в возвращаемый операцией результат. Возвращаемое операцией множество символов второго порядка может быть пустым
9	*getList: X UID k*	Операция получения списка элементов АГ-памяти из заданного множества по заданному уникальному идентификатору. Операция возвращает список элементов АГ-памяти *k*. Если для заданного уникального идентификатора не существует соответствующего списка, то операция возвращает пустой объект
10	*findLists: X e {k}*	Операция получения множества списков элементов АГ-памяти. Получает на вход в качестве операндов множество элементов АГ-памяти, среди которых надо осуществить поиск, а также заданный элемент. Возвращает множество таких списков, элементом которых является второй операнд. Возвращаемое операцией множество списков элементов АГ-памяти может быть пустым
11	*getTemplate: X UID t*	Операция получения шаблона модели управления предикатного символа по его уникальному идентификатору. Получает на вход множество элементов АГ-памяти и уникальный идентификатор, а возвращает найденный шаблон. Если шаблона для заданного идентификатора не существует, то операция возвращает пустой объект
12	*getHypernode: X UID n*	Операция получения гиперсвязи по ее уникальному идентификатору. Получает на вход множество элементов АГ-памяти и уникальный идентификатор, а возвращает найденную гиперсвязь. Если гиперсвязи для заданного идентификатора не существует, то операция возвращает пустой объект

Окончание таблицы

№ п/п	Сигнатура операции	Описание операции
13	*findHypernodes: X* (*s* | *t* | *e*) {*n*}	Операция получения множества гиперсвязей из заданного множества элементов АГ-памяти в соответствии с условием, заданным элементом АГ-памяти одного из трех типов: абстрактный символ, шаблон модели управления предикатного символа или произвольный элемент АГ-памяти. Если вторым операндом операции является абстрактный символ или произвольный элемент АГ-памяти (но не шаблон модели управления предикатного символа), то операция возвращает множество гиперсвязей, в которых хотя бы один актант ссылается на второй операнд. Если вторым операндом является шаблон модели управления предикатного символа, то операция возвращает все гиперсвязи, которые построены по соответствующему шаблону. Возвращаемое операцией множество гиперсвязей может быть пустым
14	*findRoles: X r v {n}*	Операция для получения множества гиперсвязей из заданного множества элементов АГ-памяти, у которых актант с ролью *r* заполнен значением *v*. Эта операция может использоваться, например, для поиска фактов, относящихся к одному действующему лицу (роль SUBJECT) или к одному месту (роль LOCATION) и т. д. Возвращаемое операцией множество гиперсвязей может быть пустым
15	*getLink: X UID l*	Операция для получения ассоциативной ссылки между двумя элементами АГ-памяти по ее уникальному идентификатору. Получает на вход множество элементов АГ-памяти и уникальный идентификатор, а возвращает найденную ассоциативную связь. Если ассоциативной связи для заданного идентификатора не существует, то операция возвращает пустой объект
16	*findLinks: X e {l}*	Операция для получения множества ассоциативных связей из заданного множества элементов АГ-памяти в соответствии с условием, заданным элементом АГ-памяти, который должен принадлежать множеству, передаваемому в первом операнде. Возвращаемое множество ассоциативных связей представляет такие связи, которые исходят из или входят в заданный вторым операндом элемент АГ-памяти. Возвращаемое операцией множество ассоциативных связей может быть пустым

ных роботов. Такие модели позволят сопоставлять части конструкции робота между собой при наблюдении за поведением стороннего агента.

На базе низкоуровневых операций АГ-память может использоваться для самых разнообразных высокоуровневых приложений, например, для задачи привязки символов. Формализм АГ-памяти уже содержит в своем составе множество *S*, которое включает в себя все множество символов, привязанных к результатам сенсорного восприятия ИИ-агента, что автоматически приводит к решению задачи привязки символов. АГ-память может решать задачи персонализации и персонификации. Описанный подход позволяется в полной мере реализовать схему персонализированного общения с пользователем [5]. Наличие в АГ-памяти ИИ-агента позволяет ему создавать и наполнять модель самого себя, что может использоваться для персонификации общения с пользователями. Следует отметить, что использование АГ-памяти не ограничивается приведенными примерами, АГ-память может широко использоваться для решения множества задач.

Представленный формализм может быть использован для решения перечисленных далее задач обработки естественного языка (ЕЯ).

- Информационный поиск, поскольку АГ-память позволяет структурированно описать факты и отношения между ними в тексте, образуя гиперграф из фактов, на который можно отправлять формализованные запросы и получать ответы на базе первоначального текста. Фактически, АГ-память решает задачу информационного поиска с помощью перевода вопросов в структуру, аналогичную АГ-памяти, с дальнейшим выравниванием двух гиперграфов и заполнением вакантных актантов запроса информации.
- Тематический анализ и синтез ЕЯ-фраз из общих и частных знаний ИИ-агента, собранных в множествах *C*, *P* и *H* в АГ-памяти, которые постоянно пополняются в процессе работы АГ-памяти.
- Машинный перевод, при котором при переходе от одного языка к другому сначала формируется внутреннее представление смысла входной фразы с первого языка, которое затем преобразуется в ЕЯ-представление на втором языке.
- Автоматическая классификация и кластеризация текстов — задача обработки ЕЯ-текстов, которая достаточно просто решается при помощи АГ-памяти, так как формальная структура подхода позволяет осуществить подсчет различных метрик текста и получить вывод информации на основании множества полученных метрик с помощью множества полученных значений.
- Автоматическое извлечение фактов (знаний) из текстов. На решении этой задачи основана

АГ-память, т. е. на пополнении основного текста автоматическим извлечением фактов из текста.

- Разработка автоматических вопросо-ответных систем, построенная на основе АГ-памяти при помощи решения задачи информационного поиска. Для заданного текста строится гиперграф АГ-памяти, вопросы преобразуются к виду элемента АГ-памяти, после чего извекается ответ на основе информационного поиска и преобразуется в ЕЯ-текст, возвращающийся пользователю вопрос-ответной системы.

Заключение

Приведены основные операции над АГ-памятью, которые оценивают общую схему использования этого формализма при разработке ИИ-агентов различной природы. Однако в каждом отдельном случае разработчик волен выбирать, какие компоненты приведенной системы ему использовать. При реализации конкретного ИИ-агента разработчик должен учитывать, что используемые операции не ограничены представленными в статье. Он должен выбирать те подмножества операций, которые необходимы для построения собственного ИИ-агента.

Список литературы

1. **Raina V., Krishnamurthy S.** Natural Language Processing // Building an Effective Data Science Practice. Apress, Berkeley, CA. 2022. P. 63—73. DOI: 10.1007/978-1-4842-7419-4_6.
2. **Душкин Р. В.** Обзор подходов и методов искусственного интеллекта // Радиоэлектронные технологии. 2018. № 3. С. 85—89.
3. **Душкин Р. В.** Искусственный интеллект. М.: ДМК-Пресс, 2019. 280 с.
4. **Zadeh L. A.** From computing with numbers to computing with words — From manipulation of measurements to manipulation of perceptions// Int. J. Appl. Math. Comput. Sci. 2001. Vol. 12, No. 3. P. 307—324.
5. **Душкин Р. В.** Развитие методов адаптивного обучения при помощи использования интеллектуальных агентов // Искусственный интеллект и принятие решений. 2019. № 1. С. 87—96.
6. **Xu Yan, Yining Wang, Tianren Liu** et al. An end-to-end system to identify temporalrelation in discharge summaries: 2012 i2b2challenge// Journal of the American Medical Informatics Association. 2012. Vol. 20, No. 5. P. 849—858. DOI: 10.1136/amiajnl-2012-001607.
7. **Sohrab M. G., Khoa D., Makoto M.** et al. BENNERD: A neural namedentity linkingsystem for COVID-19 // Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. 2020. P. 182—188. DOI: 10.18653/v1/2020.emnlp-demos.24.
8. **Kuniyoshi Fusataka, Jun Ozawa, MikiyaFujii** et al. Graph representation forsynthesis process extraction frominorganic material literature // IEICETechnical Report. 2019. Vol. 119 (212). P. 7—12.
9. **Yoshinobu K., Miwa M., Cohen K. B.** et al. U-Compare: A modular NLP workflow construction and evaluationsystem // IBM Journal of Research and Development. 2011. Vol. 55, No. 3. P. 11. DOI: 10.1147/JRD.2011.2105691.
10. **Dushkin R. V., Lelekova V. A., Stepankov V. Y., Fadeeva S.** The structure of associative heterarchical memory // SSRN Electronic Journal. 2022. DOI: 10.2139/ssrn.4196439.
11. **Zhu L., Gao W.** Hypergraph Ontology Sparse Vector Representation and Its Application to Ontology Learning // Data Mining and Big Data. DMBD 2021. Communications in Computer and Information Science / Y. Tan, Y. Shi, A. Zomaya, H. Yan, J. Cai (eds), Springer, Singapore. 2021. Vol. 1454. DOI: 10.1007/978-981-16-7502-7_2.
12. **Harnad S.** The Symbol Grounding Problem // Physica. 1990. D 42. P. 335—346. URL: https://bit.ly/3z5mHcl (дата обращения 29.12.2021).
13. **Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.** Часть VI. Алгоритмы для работы с графами. Алгоритмы: построение и анализ = Introduction to algorithms. 2-е изд. М.: Вильямс, 2006. 1296 с.
14. **Jezequel J.-M.** Domain Specific Languages: From Craft to Engineering // iiWAS '14: Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services, December, 2014. 2 p. DOI: 10.1145/2684200.2684370.
15. **Flemmer R. C.** A scheme for an embodied artificial intelligence // 2009 4th International Conference on Autonomous Robots and Agents. 2009. P. 1—9. DOI: 10.1109/ICARA.2000.4804031.
16. **Makondo Ndivhuwo**. Accelerating robot learning of motor skills with knowledge transfer. Thesis for: Doctor of Philosophy in Computational Intelligence and Systems Science. 2018. DOI: 10.13140/RG.2.2.26694.32329.

System of Operations on Associative Heterarchical Memory

R. V. Dushkin, Chief Science and Technology Officer, drv@aiagency.ru,
V. A. Lelecova, Analyst, lv@aiagency.ru, **K. Yu. Eidemiller,** Academic Chief,
Artificial Intelligence Agency, Moscow, 127473, Russian Federation

Corresponding author:

Vasilisa A. Lelecova, Analyst,
Artificial Intelligence Agency, Moscow, 127473, Russian Federation,
E-mail: lv@aiagency.ru

Received on September 01, 2022
Accepted on November 13, 2022

Text processing in natural language remains an important task for the field of development of artificial intelligence methods and tools. Since the twentieth century, artificial intelligence methods have been divided into two paradigms — top-down and bottom-up. The methods of the ascending paradigm are difficult to interpret in the form of the output

of natural language, and the methods of the descending paradigm are difficult to actualize information. Taking into account the authors' approach to the construction of artificial intelligence agents, the processing of natural language must be performed on two levels: on the lower level, using methods of the bottom-up paradigm, and on the upper level, using symbolic methods of the top-down paradigm. The authors of the article have already introduced a new mathematical formalism based on the notion of a hypergraph — associative heterarchical memory (AH-memory). Such memory should simplify the process of natural language processing with new technologies. Earlier the authors' group has thoroughly analyzed the problem of symbol binding in the application to AH-memory and its structure. In the first paper, abstract symbol binding was performed using multi-serial integration, eventually converting the primary symbols received by the program into integrated abstract symbols. The second paper provided a comprehensive description of the AH-memory in the form of formulas, explanations of them, and their corresponding diagrams. Although there are many possible modules to use, the developer working with AH-memory should choose those parts of AH-memory which are required for successful and efficient functioning of the AI agent. The article will be of interest to developers of artificial intelligence methods and tools, mathematicians and specialists in natural language processing.

Keywords: *associative heterarchical memory, natural language processing, hypergraph, mathematical formalism, artificial intelligence, symbolic method, abstract symbols, operation system, AI agent*

For citation:

Dushkin R. V., Lelecova V. A., Eindemiller K. Yu. System of Operations on Associative Heterarchical Memory, *Programmnaya Ingeneria,* 2023, vol. 14, no. 2, pp. 69—76. DOI: 10.17587/prin.14.69-76 (in Russian).

References

1. **Raina V., Krishnamurthy S.** Natural Language Processing, *Building an Effective Data Science Practice.* Apress, Berkeley, CA. 2022, pp. 63—73. DOI: 10.1007/978-1-4842-7419-4_6.

2. **Dushkin R. V.** Overview of Artificial Intelligence Approaches and Methods, *Radioelectronnye techlonogii,* 2018, no. 3, pp. 85—89 (in Russian).

3. **Dushkin R. V.** *Artificial Intelligence*, Moscow, DMK-Press, 2019, 280 p. (in Russian).

4. **Zadeh L. A.** From computing with numbers to computing with words — From manipulation of measurements to manipulation of perceptions, *Int. J. Appl. Math. Comput.Sci.*, 2001, vol. 12, no. 3, pp. 307—324.

5. **Dushkin R. V.** Development of adaptive learning methods using intelligent agents, *Iskusstvenniy intellect i prinyatie resheniy*, 2019, no. 1, pp. 87—96 (in Russian).

6. **Xu Yan, Yining Wang, Tianren Liu** et al. An end-to-end system to identify temporalrelation in discharge summaries: 2012 i2b-2challenge, *Journal of the American Medical Informatics Association*, 2012, vol. 20, no. 5, pp. 849—858. DOI: 10.1136/amiajnl-2012-001607.

7. **Sohrab M. G., Khoa D., Makoto M.** et al. BENNERD: A neural namedentity linkingsystem for COVID-19, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, pp. 182—188. DOI: 10.18653/v1/2020.emnlp-demos.24.

8. **Kuniyoshi Fusataka, Jun Ozawa, Mikiya Fujii** et al. Graph representation forsynthesis process extraction frominorganic material literature, *IEICETechnical Report*, 2019, vol. 119 (212), pp. 7—12.

9. **Yoshinobu K., Miwa M., Cohen K. B.** et al. U-Compare: A modular NLP workflow construction and evaluationsystem, *IBM Journal of Research and Development*, 2011, vol. 55, no. 3, pp. 11. DOI: 10.1147/JRD.2011.2105691.

10. **Dushkin R. V., Lelekova V. A., Stepankov V. Y., Fadeeva S.** The structure of associative heterarchical memory, *SSRN Electronic Journal*, 2022. DOI: 10.2139/ssrn.4196439/

11. **Zhu L., Gao W.** Hypergraph Ontology Sparse Vector Representation and Its Application to Ontology Learning, *Data Mining and Big Data. DMBD 2021*, *Communications in Computer and Information Science* / Y. Tan, Y. Shi, A. Zomaya, H. Yan, J. Cai (eds), Springer, Singapore, 2021, vol 1454. DOI: 10.1007/978-981-16-7502-7_2.

12. **Harnad S.** The Symbol Grounding Problem, *Physica*,1990. D 42, pp. 335—346, available at: https://bit.ly/3z5mHcl (date of access 29.12.2021).

13. **Cormen T. Lejzerson Ch., Rivest R., Shtain K.** *Introduction to algorithms.* 2nd ed. Moscow, Wiliams, 2006, 1296 p. (in Russian).

14. **Jezequel J.-M.** Domain Specific Languages: From Craft to Engineering, *iiWAS'14: Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*, 2014, 2 p. DOI: 10.1145/2684200.2684370.

15. **Flemmer R. C.** A scheme for an embodied artificial intelligence, *2009 4th International Conference on Autonomous Robots and Agents*, 2009, pp. 1—9. DOI: 10.1109/ICARA.2000.4804031.

16. **Makondo Ndivhuwo.** Accelerating robot learning of motor skills with knowledge transfer, Thesis for: Doctor of Philosophy in Computational Intelligence and Systems Science, 2018. DOI: 10.13140/RG.2.2.26694.32329.

DOI: 10.17587/prin.14.77-81

A Method of Representing Cyclic Program Structures in Artificial Chemistry Model

E. A. Kol'chugina, D. Sci., Professor, kea_sci@list.ru,
Penza State University, Penza, 440026, Russian Federation

Corresponding author:

Elena A. Kol'chugina, D. Sci., Professor,
Penza State University, Penza, 440026, Russian Federation
E-mail: kea_sci@list.ru

Received on November 24, 2022
Accepted on December 15, 2022

The need for automation of software development processes makes it necessary to search for forms of program representation that can undergo automatic transformations without violating the integrity and semantic significance of the results of such transformations. Previously, we have proposed a notation for programs that permits the use of automatic transformation methods, namely methods of evolutionary development used, in particular, in genetic programming. But we have considered only linear and tree-like structures. In this article, we expand the list of available types of structures by adding cyclic structures, as well as complex structural compositions obtained by combining structures of simpler types. We also propose a rule excluding possible anomalies with cyclic structures representation. In general, the proposed methods are based on the concept of artificial chemistry, where programs are considered as analogues of molecules, and program transformations are considered as analogs of reactions. We illustrate the application of proposed notation using examples of the Kekule formula and the cyclic program, automatically obtained in our previous studies. The results obtained demonstrate that the proposed notation and methods make it possible to compose formulas representing computational structures of various and even mixed types.

The results are necessary for software engineering and artificial intelligence to develop methods of automatic synthesis and transformation of programs.

Keywords**:** *artificial chemistry, artificial molecules structure, cyclic and complex structures representation, angle brackets notation*

For citation:

Kol'chugina E. A. A Method of Representing Cyclic Program Structures in Artificial Chemistry Model, *Programmnaya Ingeneria,* 2023, vol. 14, no. 2, pp. 77—81. DOI: 10.17587/prin.14.77-81.

УДК 004.42

Е. А. Кольчугина, д-р техн. наук, доц., проф. кафедры, kea_sci@list.ru,
Пензенский государственный университет

Метод представления циклических программных структур в модели искусственной химии

Поступила в редакцию 24.11.2022
Принята к публикации 15.12.2022

Потребность в автоматизации процессов разработки программного обеспечения обусловливает необходимость поиска форм представления программ, которые могут подвергаться автоматическим преобразованиям без нарушения целостности и семантической значимости результатов таких преобразований. Ранее автором была предложена скобочная нотация для

записи программ, которая допускает использование методов автоматической трансформации, а именно методов эволюционной разработки, используемых, в частности, в генетическом программировании. Но тогда были рассмотрены только линейные и древовидные структуры. В этой статье список доступных типов структур расширен, к нему добавлены циклические структуры, а также сложные структурные композиции, полученные путем объединения структур более простых типов. Также сформулировано правило, исключающее возможные аномалии при представлении циклических структур. В целом, предлагаемые методы основаны на концепции искусственной химии, где программы рассматривают как аналоги молекул, а преобразования программ — как аналоги реакций. Применение предложенной нотации проиллюстрировано на примерах формулы Кекуле и циклической программы, автоматически полученной в ходе предыдущих исследований. Полученные результаты демонстрируют, что предложенные нотация и методы позволяют составлять формулы, представляющие вычислительные структуры различных и даже смешанных типов. Полученные результаты необходимы для развития программной инженерии и искусственного интеллекта в целях разработки методов автоматического синтеза и преобразования программ.

Ключевые слова: *искусственная химия, структура искусственных молекул, представление циклических и сложных структур, скобочная нотация*

Для цитирования:

Kol'chugina E. A. A Method of Representing Cyclic Program Structures in Artificial Chemistry Model // Программная инженерия. 2023. Том 14, № 2. С. 77—81. DOI: 10.17587/prin.14.77-81.

Introduction and background

In fact, this article is devoted to the automation of software design using the methods of evolutionary design and self-organization. To make such automation possible, it is necessary to know how to present any kind of program in a form suitable for automatic transformations, and to offer the means of such transformations.

With the development of parallel and distributed programming technologies, it became obvious that a program is not only a dynamic object unfolding in time, but also a spatial structure. Thus, it becomes important to study the possible types of such structures and their evolution over time, as well as to consider ways of spontaneous formation of these structures as a result of self-organization.

We have devoted many of our previous works to the study of self-organization of programs in computer medium, exploring various models and modeling methods. In some of these works, we have studied the formation of cyclic computing structures [1].

It is stated in origin of life theory that primordial life structures represented primitive molecular organisms were cyclic structures [2]. Some of the self-organizing dynamic life-like structures may possibly be present inside stars [3]. Cyclic structures are robust and capable of error protection, which makes them suitable candidates for creating artificial life forms in a computer medium.

In [1], we have achieved the emergence of cyclic structures from a pool of independent processes performing simple computational functions. The components of these structures were loosely connected, and their connections were indirect and implicit, which made the structures easily destroyed by invading processes. But in the absence of invasions, structures can persist.

In order to achieve a more obvious effect of self-organization and to make interprocess connections explicit and direct, we have suggested in [4] an approach that simulates the force of attraction observed in real-world physics. We have further developed this approach in [5] and proposed a new tool based on Internet sockets for modeling electric charge using "sinks" and "sources".

We have also proposed a notation capable of representing program structures consisting of functions. According to the concept of artificial chemistry [6, 7], we can consider such program structures as artificial molecules, while the functions forming them are considered as atoms. The notation we have given in [5] uses special slot symbols in addition to functions, variables, and parentheses. The slot symbols correspond to "sinks" and "sources" in formulas describing functional program structures. Any slot symbols indicate the position in which the function can be inserted, as well as represent the direction of the flow of the model electric charge.

In [5] we have found that the proposed approach and tools for representing the electric charge are effective and allow to simulate the force of attraction and the formation of complex artificial molecules

(i. e. programs). But the molecules obtained in the experiments described in [5] had a linear structure. The notation given in [5] potentially makes possible the description of molecules with tree-like structure also.

In this paper, we investigate the possibility of representing cyclic and more complex structures using the notation from [5]. The aim of this article is to prove that notation from [5] is capable of representing complex spatial structures with different topologies.

Method

Let's recall the notation we have given earlier in [5]. This notation is able to represent interactions between functions similar to the force of electric attraction. Any function can be an empty function, a function without arguments (i. e. a variable), or have one or more arguments. In turn, any argument of any function can be a variable (i. e. a null-argument function) or another function with a non-empty argument list. Wildcard symbols are also present in the notation. These symbols, or slot symbols, represent the positions in which the new functions will be substituted, and determine the conditions for such substitutions. The wildcard symbols $\{>,<\}$ represent "sinks" (<) and "sources" (>), respectively, and show the direction of the model analogue of the electric charge flow. The number of wildcard symbols represents the number of electrons or vacant positions in the outer orbitals. Being inserted instead of the slot symbol, the function is enclosed in angle brackets of a special kind, also representing the direction of the model charge flow.

Regarding simple functions and variables as atoms, their compositions should be considered as molecules. For example, as it was shown in [5], functions $f(X_1,>,Y_1)$ and $g(X_2,<,Y_2)$ can produce such compositions as $f(X_1,> g(X_2,Y_2)<,Y_1)$ and $g(X_2,< f(X_1,Y_1)>,Y_2)$, since the slot symbols used by these functions are oppositely directed, that is, complementary and compatible. These compositions generally describe the same thing, but considered from different starting points. The angle brackets, as it was mentioned earlier, show a direction of charge flow.

The described notation given in [5] obviously allows representing molecules (i. e. programs, formulas) with linear and tree-like structures, as all function-oriented notations do [8]. But the notation from [5] has a fundamental distinguishing feature: it also allows representing cyclic structures.

To do this, we can simply connect the innermost angle brackets with the complementary external angle brackets of the outermost function. This is possible when the atoms closing the cycles (i. e., the innermost and outermost functions) have both "sinks" and "sources" that can be connected together. For example, for the functions $h(X_1,>,<,Y_1)$ and $j(X_2,<,>,Y_2)$ it is possible to construct a cyclic structure described both as $>h(X_1,> j(X_2,><,Y_2)<,Y_1)<$ and $< j(X_2,< h(X_1,<>,Y_1)>,Y_2)>$.

Let's test our assumption with the example of a benzene ring. Figure 1 illustrates the Kekule formula and describes the transformation of the benzene ring formula to make it possible to represent it using the notation described in this section.

The problem with the representation of a benzene ring using the notation from [5] is that this notation allows representing only the simplest forms of generalization of electrons between atoms. In the given notation, generalization is possible only between two atoms, but not within their group. But in the real world, the generalization of electrons in a benzene ring occurs within a group of atoms. To avoid problems with the representation of such a generalization in the model, it is necessary to invite two forms for "carbon" atoms capable to establish single and double electron connections.

To simplify the model, we sacrifice the representation of the "hydrogen" atom as a function, neglecting its arguments. We reduce representation of atom of "hydrogen" to the level of a variable, considering any compound of model artificial "carbon" and "hydrogen" as a stable group. We also shorten the lists of arguments and results for functions representing artificial "carbon" atoms to make the formulas easier to read.

Hence, we have two functions at our disposal, $C_1(X_1,>,<<)$ and $C_2(X_2,>>,<)$ (Figure 1, *b*), and also a set of instances of variables $\{H_1, \ldots, H_6\}$ of the type *H*. We can also perform instantiating of functions $C_1(X_1,>,<<)$ and $C_2(X_2,>>,<)$, where each of the instances corresponds to a single computation process. Thus, we get two sets of functions $\{C_1^1, C_1^2, C_1^3\}$ and $\{C_2^1, C_2^2, C_2^3\}$. The reorganized structure is represented in Figure 1, *c*.

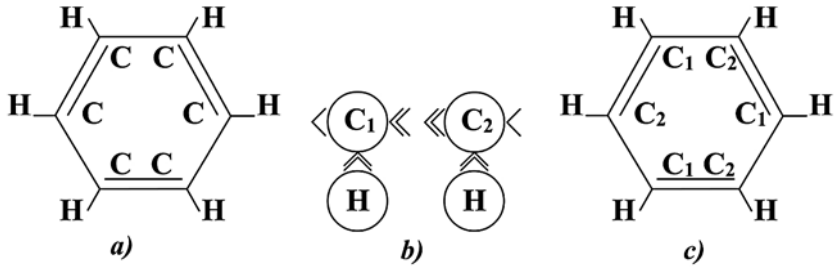

Figure 1. Kekule formula transformation:
a — original formula; *b* — two forms of artificial "carbon" atoms in compounds with artificial "hydrogen" atoms; *c* — formula after the transformation

The resulting form is:

$$> C_2^1\left(H_1,>> C_1^1\left(H_2,> C_2^2\left(H_3,>> C_1^2\left(H_4,> C_2^3\left(H_5,>> C_1^3\left(H_6,><\right)<<\right)<\right)<<\right)<\right)<<\right)< .$$

This form represents a program that can dynamically self-assemble from a set of independent processes performing primitive functions using the means of interaction described in [1, 5].

An additional question that should be answered is the question of the correctness of structures created by the described method. In the plainest and most evident form, the rule excluding possible anomalies with cyclic structures is as follows: the summary number (i. e. power according to [5]) of the outermost angle brackets, like >< or <>, must be equal to the summary number (power) of the complementary innermost angle brackets, like <> or ><, respectively. We defined the term of power for "sinks" and "sources" earlier in [5]. It corresponds to the number of charged particles or possible vacant positions in the outer orbitals of the atom.

In general, pairs of oppositely directed angle brackets should compensate each other, and their balance should coincide. As for our example with the Kekule formula, this rule is followed.

Results and discussion

The resulting representation of the Kekule formula given above illustrates that cyclic structures can also be represented using the notation from [5]. All that remains to be said is to characterize the type of cycles and other structures suitable for representation, and give examples of real program structures that should be represented using this notation.

It is obvious that the cyclic structures of the real world chemistry, which can be represented using this notation, belong to the simplest forms. In chemistry of the real world, there are different types of electron clouds that generalize electrons between atoms in molecules. But in the notation from [5] generalization is possible only within a pair of atoms, and the number of bonds possible for atoms of the same type must be constant and regular.

As for the structure of the benzene molecule and the Kekule formula, which we used as an example, this structure is characterized by the use of π-electrons common to all carbon atoms in the benzene ring. However, it is impossible to imagine electrons shared by several atoms in our model. The reason for this, as indicated in [4, 5], is the simplified physics of the artificial virtual world. As a result, we are forced to introduce into the model different subspecies of atoms of the same chemical element, a kind of isotopes, in order to resolve the contradiction that has arisen.

An important matter is which programs should be presented in the form of cyclic structures of the type in question. In our previous articles [1, 4, 5], we discussed the possibility of the emergence of digital organisms in a computer medium. Digital organisms studied in the field of artificial life, one of the divisions of which is artificial chemistry [6, 7], are self-sustaining and in some cases self-reproducing software and information structures that demonstrate some of the most important properties of living organisms. As it has been shown in [1], such digital organisms, or at least their sketches, can arise spontaneously in experiments under given conditions. After the end of experiments, it is possible to subsequently extract these sketches from the protocol files for using and improving. The sketches can be used immediately as ready programs, or be further developed. But it is important to represent the cyclic nature of the resulting programs, so it is necessary to offer a special notation for this.

The notation given in [5] and further developed here satisfies the requirements of this task, and allows representing cyclic structures from [1] in string form. Let us consider one of the most lengthy program cycle from [1], $f_3g_0j_1f_3i_2f_3i_2$, this structure is represented in Figure 2, *a*. Using notation from [5] and our proposed refinement for representation of the cycles, it will look like

$$> f_3\left(> g_0(> j_1(> f_3(> i_2(> f_3(> i_2\left(><\right)<)<)<)<)<)<\right)< .$$

Again, here we neglect the arguments of the function in favor of readability of resulting formula.

The proposed notation allows creating formulas suitable for evolutionary development methods implementation [9—12]. The methods of evolutionary development, namely genetic algorithms and genetic programming, are known to be implemented to linear and tree-like structures chromosomes representing possible solutions. These methods, specifically crossover methods, in fact are exchange and substitution methods. Two or more participating chromosomes exchange their fragments, and these new fragments replace the previously existing ones.

In case of genetic algorithms and genetic programming, chromosomes can be only of linear or sub-tree structure. But the notation proposed and

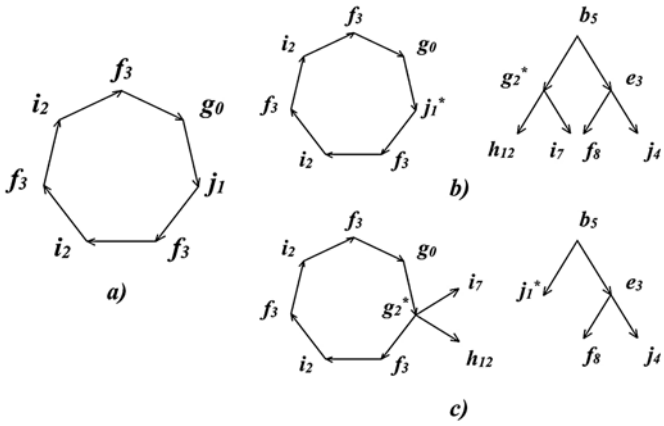

Figure 2. Cycles, trees and example of their transformation:
a — a cycle obtained and observed during the experiments in [5]; *b* — an example of a cycle and a tree-like structure before the crossover; *c* — the resulting structures after the crossover

developed in [5] and here allows more complicated schemes of crossover. To make crossover methods non-destructive, we allow the exchanging fragments to be only sub-formulas given in angle brackets like >< or <>. Therefore, the substituted formula itself can be of linear, tree-like and cyclic structure. An example of such transformation is given in Figure 2, *b*, *c*. The crossover points are marked with asterisks in both structures.

This allows creation of very complex structures containing many tree-like, linear and cyclic groups. The more important is that we can not only create, but subsequently operate these structures keeping their integrity.

Conclusions

In this article, we have enhanced a method for representing compositions of functional structures, which was previously proposed in [5]. This method allows to represent cyclic and cycle-based structures, unlike the λ-calculus and genetic programming methods that allow to represent only linear and tree-like structures.

The resulting formulas representing cyclic structures are not rigidly fixed. They can be changed using the operations of evolutionary development methods, i. e. genetic programming methods and genetic algorithms.

We also proposed a rule that excludes possible anomalies in transformations of cyclic structures. This allows retaining integrity, non-contradiction and semantic significance of structures when performing their transformations.

The achieved results are in demand in software engineering for automating software development in all areas of programming, regardless of the scope of the application. The use of the obtained results in the fields of artificial intelligence and artificial life is also promising.

The results demonstrate that the proposed notation makes it possible to represent not only linear, tree-like and cyclic structures, but also their combinations, opening the way for the representation of complex planar and even non-planar 3D structures. The most interesting would be the representation of regular cellular structures like fullerenes. This can serve as a direction for further research.

The problem is that the possibilities of any artificial chemistry are limited compared to the chemistry of the real world. The source of the problem is the meager set of tools offered by computers with the von Neumann architecture for reproducing real-world physics. That is why it is necessary to introduce extra facilities into the models of artificial chemistry and to resolve arising contradictions.

References

1. **Kol'chugina E. A.** Spontaneous Emergence of Programs from "Primordial Soup" of Functions in Distributed Computer Systems, *Automatic Control and Computer Sciences*, 2018, vol. 52, no. 1, pp. 40—48. DOI: 10.3103/S0146411618010054.
2. **Eigen M., Schuster P.** *The Hypercycle: A Principle of Natural Self-Organization*, Berlin-Heidelberg-New York, Springer-Verlag, 1979, 98 p.
3. **Anchordoqui L. A., Chudnovsky E. M.** Can Self-Replicating Species Flourish in the Interior of a Star? *Letters In High Energy Physics,* 2020, vol. 2020, LHEP-166, pp. 1—4. DOI: 10.31526/LHEP.2020.166.
4. **Kol'chugina E. A.** Model reproduction of non-equilibrium thermodynamics principles as a means to provide software self-development, *IOP Conference Series: Materials Science and Engineering; III International Scientific Conference: Modernization, Innovations, Progress: Advanced Technologies in Material Science, Mechanical and Automation Engineering (MIP-III 2021)*, 29th—30th April 2021, Krasnoyarsk, Russian Federation, 2021, vol. 1155, p. 012054. DOI: 10.1088/1757-899X/1155/1/012054.
5. **Kol'chugina E. A.** Self-Synthesis of Programs Based on Artificial Chemistry Model, *Programmnaya Ingeneria,* 2022, vol. 13, no. 9, pp. 440—448. DOI: 10.17587/prin.13.440-448.
6. **Dittrich P., Ziegler J., Banzhaf W.** Artificial Chemistries — A Review, *Artificial Life*, 2001, vol. 7, no 3, pp. 225—275. DOI: 10.1162/106454601753238636.
7. **Banzhaf W., Yamamoto L.** *Artificial Chemistries,* Cambridge, Massachusetts; London, England, The MIT Press, 2015, 576 p.
8. **Church A.** An Unsolvable Problem of Elementary Number Theory, *American Journal of Mathematics*, Apr. 1936, vol. 58, no. 2, pp. 345—363.
9. **Holland J. H.** Building Blocks, Cohort Genetic Algorithms, and Hyperplane-Defined Functions, *Evolutionary Computations*, 2000, vol. 8, no. 4, pp. 373—391. DOI: 10.1162/106365600568220.
10. **Mitchell M.** *Introduction to Genetic Algorithms*, Cambridge, Massachusetts; London, England, A Bradford Book the MIT Press, 1999, 158 p.
11. **Gladkov L. A., Kurejchik V. V., Kurejchik V. M.** *Geneticheskie algoritmy*`/ Eds V. M. Kurejchik, 2-nd edition, corrected and expanded, Moscow, Fizmatlit, 2006, 320 p. (in Russian).
12. **Koza J. R., Bennett F. H., Andre D., Keane M. A.** Genetic Programming: Biologically Inspired Computation That Creatively Solves Non-trivial Problems, *Evolution as Computation. Natural Computing Series /* Eds L. F. Landweber, E. Winfree, Springer, Berlin, Heidelberg, 2002, pp. 95—124. DOI: 10.1007/978-3-642-55606-7_5.

УДК 004.92
DOI: 10.17587/prin.14.82-92

А. Д. Борисов, аспирант, radiatus@yandex.ru,
С. Д. Махортов, д-р физ.-мат. наук, зав. каф., msd_exp@outlook.com,
Воронежский государственный университет

Нежесткая регистрация человеческого лица по изображениям со стереокамеры

Поступила в редакцию 17.12.2022
Принята к публикации 26.12.2022

Захват мимики актера на съемочной площадке для дальнейшего переноса отыгранной сцены в цифровое пространство стал одной из важнейших задач в области компьютерной графики. Большинство методов решения этой задачи имеет более низкое качество реконструкции в сравнении с захватом, который осуществляется в видеограмметрических установках. Причина — в отсутствии возможности вычисления качественных трехмерных сканированных копий («сканов»). В настоящей статье выделены проблемные вопросы существующих подходов и предложено решение по захвату мимики актера по изображениям со стереокамеры шлема, которое сопоставимо со стационарным захватом. Для получения такого уровня точности предложено использовать локальную модель деформации, основанную на геометриях ключевых выражений лица, а также стереоограничение вместо явного расчета трехмерного скана.

Ключевые слова: *нежесткая регистрация, стереокамера, компьютерная графика, захват движения, цифровые дублеры*

Для цитирования:

Борисов А. Д., **Махортов С. Д.** Нежесткая регистрация человеческого лица по изображениям со стереокамеры // Программная инженерия. 2023. Том 14, № 2. С. 82—92. DOI: 10.17587/prin.82-92.

Введение

Современные стандарты индустрии кино и видеоигр требуют тщательности и точности в воспроизведении внешности людей. Важное место в области компьютерной графики занимает создание правдоподобных трехмерных моделей человеческого лица, так называемых *цифровых дублеров* реальных актеров [1]. Однако интерес представляют не статичные модели, а целиком отыгранная актером сцена. В таких сценах особое внимание уделяется мелким деталям, так как обычный зритель легко может отличить подделку в виде цифрового дублера. В настоящее время существуют несколько способов захвата изображений актерской игры.

Наиболее популярным способом является захват анимации человеческого лица с помощью камер машинного зрения на шлеме. Заранее формируется лицевой скелет, который может относительно точно передать мимику лица реального актера [2]. Для создания анимационной сессии с использованием этого скелета необходимо установить на голове актера шлем с камерой для захвата лица, а также провести аннотирование лица маркерами. Впоследствии в результате решения задачи оптимизации лицевой скелет повторяет движения маркеров с видеопотока камеры. На выходе получается анимационная сессия для данного актера, которую можно использовать в производстве. Этот способ неудачен тем, что в результате не учитываются мелкие движения и изменения лица актера, что очень важно для передачи реалистичной мимики, в противном случае зритель перестает верить увиденной картине.

Существует другой способ, заключающийся в помещении актера в установку, состоящую из большого числа видеокамер и световых панелей, в которой проходит съемка необходимой сцены [3]. В результате получается секвенция трехмерных

сканов, полностью повторяющая отыгранную актером сцену. Такой метод называется видеограмметрией по аналогии с фотограмметрией. Для дальнейшего использования в производстве сканы необходимо преобразовать к корректной топологии, так как «сырые» сканы занимают большой объем памяти компьютера, содержат артефакты. Кроме того, у них отсутствует семантика вершин между кадрами. Эта проблема решается с помощью ретопологии художником каждого кадра из последовательности. Для автоматизации данного процесса используют алгоритм нежесткой регистрации [4], позволяющий деформировать заранее заготовленную модель с корректной топологией (базовая модель) под другую модель с иной топологией (целевая модель). Для получения консистентности топологии во времени используют также текстурную информацию сканов. Такая технология получения анимации лица называется 4D-захватом [5]. Она позволяет относительно автоматизировано, а также максимально достоверно, в отличие от первого упомянутого метода, передать геометрию мимики реального актера для каждого кадра.

В итоге второй метод позволяет достичь наивысшего качества захвата человеческого лица, но является совершенно немобильным и весьма дорогим, поскольку требует большого количества аппаратуры. Первый же способ гораздо мобильнее, однако характеризуется более низким качеством захвата.

Таким образом, возникает потребность в совместном использовании преимуществ обоих подходов, а именно: качества 4D-захвата и мобильности камер машинного зрения на шлеме. Негативная особенность 4D-захвата заключается в том, что для обработки изображений необходимо наличие сканов. Их можно легко зафиксировать в фотограмметрической или видеограмметрической установке, однако проблематично со стереокамеры шлема. Зачастую на шлем устанавливаются лишь две камеры, по которым в лучшем случае можно получить скан низкого качества с большим числом артефактов. Предлагаемая идея заключается в том, чтобы отказаться от явного построения сканов для захвата и использовать изображения напрямую. Это позволит применять алгоритмы 4D-захвата сразу на изображениях, а следовательно, на данных со шлема.

В настоящей работе описан метод нежесткой регистрации человеческого лица со стереокамеры шлема. Цель этого метода — получение качества захвата мимики актера на съемочной площадке, сопоставимого с качеством стационарных систем захвата, основанных на расчете и обработке трехмерных сканов.

Смежные работы

Метод нежесткой регистрации лица актера по структуре представляет своего рода конструктор, который реализуется отдельными заменяемыми модулями. В частности, можно явно выделить *модель деформации* и *функцию энергии*, которая отвечает за то, как модель деформируется.

Существует ряд методов, позволяющих провести нежесткую регистрацию человеческого лица с камер шлема, закрепленного на голове актера [6—8]. Большинство из них использует в качестве модели деформации глобальное линейное смешивание заранее заготовленных обобщенных геометрий ключевых выражений лица с примененными глобальными трансформациями. Этот фактор влечет за собой весьма низкое качество реконструкции лица актера, а также требует большого числа ключевых геометрий, чтобы покрыть все возможные формы лица.

Более гибким способом является использование локальных моделей. В них топология геометрии разбивается на отдельные фрагменты — *патчи* [4, 9]. Каждый патч способен перемещаться в пространстве отдельно от другой части геометрии, но с соблюдением определенной гладкости относительно соседних патчей, чтобы не производить разрывы. Такая модель позволяет получить геометрию высокого качества, однако в силу отсутствия априорной информации о форме человеческого лица в виде ключевых геометрий она способна выдавать большое число артефактов.

В настоящей работе предлагается использование локальной модели деформации, основанной на ключевых геометриях. Этот способ позволит уменьшить число необходимых геометрий и повысить точность относительно моделей, основанных на глобальном смешивании ключевых геометрий. При этом в силу использования априорной информации о форме лица с помощью ключевых геометрий не будет допущено появление артефактов относительно локальных моделей.

В качестве основной функции энергии («движущей силы» для модели деформации) в большинстве решений принимается евклидово расстояние между координатами отслеживаемых маркеров на изображениях и проекциями на геометрии трехмерных точек, соответствующих этим маркерам [7, 8]. Минус такого подхода в том, что реализуется своего рода разреженный захват, поскольку теряется информация между маркерами,

отображающая мельчайшие движения кожи, которые необходимо захватывать для высокого качества реконструкции.

Для более плотного захвата лица используют последовательный расчет оптического потока от первого кадра к последнему [10—12]. Далее выполняется последовательный расчет нежесткой регистрации с учетом оптического потока [13]. Вычисление очередного кадра использует в качестве инициализации результат нежесткой регистрации из предыдущего кадра. К недостаткам данного подхода можно отнести требование наличия уже готового первого кадра секвенции, а также накапливание ошибок при последовательных вычислениях. Кроме того, расчет оптического потока здесь проводится отдельно от общей задачи, и он не учитывает форму человеческого лица, что влечет за собой появление артефактов.

Описываемый в настоящей работе метод предлагает вместо последовательного расчета оптического потока использовать фотометрическое сравнение рендера текущего состояния модели деформации и целевых изображений. При его использовании не требуется заранее готовый первый кадр секвенции, а также уменьшается число артефактов.

Для более точного «попадания» в целевое лицо используется расчет трехмерного скана [9] или карты глубины [7]. Однако изображений с двух камер недостаточно для получения скана или карты глубины хорошего качества. Такой результат имеет большое число артефактов [7]. Поэтому необходимы различные средства отсечения «плохих» данных. В частности, это может быть маскирование плохо видимых, но важных участков лица, чтобы не осуществлять подгонку по глубине для этих регионов. К тому же алгоритмы расчета трехмерного скана или глубины являются отдельными задачами, не имеющими априорной информации о форме лица (аналогично оптическому потоку). Это обстоятельство также является одной из причин большого числа артефактов.

В представленном исследовании для задачи нежесткой регистрации лица использовано стереоограничение, которое более подробно описано в отдельной работе [14]. Оно позволяет осуществлять «подгонку» модели деформации по глубине без отдельного расчета трехмерного скана или карт глубин в рамках общей задачи нежесткой регистрации. В результате минимизируется число артефактов.

Подготовка данных

Для демонстрации работы метода необходим определенный набор данных. Он включает в себя следующие элементы: ключевые геометрии и соответствующие им текстуры конкретного актера, откалиброванные камеры с соответствующими изображениями, а также набор отслеженных маркеров и контуров на изображениях.

Ключевые геометрии представляют собой набор трехмерных моделей выражений лиц данного актера, покрывающий значительную часть того, что может изображать человеческое лицо. Пример таких геометрий представлен на рис. 1 (см. третью сторону обложки).

Для определения внутренних и внешних параметров камеры должны пройти процесс калибровки по изображениям шахматной доски. Далее осуществляется процесс удаления искажения изображений, вызванных используемыми в камерах линзами. Для этого применяют встроенные средства библиотеки OpenCV.

Отслеживание маркеров и контуров на лице актера проводится с помощью стандартных методов, встроенных в библиотеку OpenCV. Однако точность отслеживания такими методами невысока, поэтому дополнительно используется персонализированный детектор [15].

Модель деформации должна получить априорную информацию о том, как может изменяться геометрия лица актера. В этих целях необходимо предоставить алгоритму заранее обработанные ключевые геометрии актера с текстурами. Эти текстуры будут использоваться для генерации динамической текстуры внутри самого алгоритма для осуществления более точного сравнения рендера и целевого изображения со шлема.

Модель деформации

Основная отличительная особенность предлагаемой модели деформации заключается в том, что это локальная модель с априорной информацией о каждом регионе в отдельности.

Исходная трехмерная геометрия лица, представленная набором связанных в общую топологию трехмерных точек, разбивается на ряд ключевых зон, так называемых *патчей.* Это разбиение можно сделать вручную, с учетом анатомии, а можно и автоматически. Для автоматического разбиения связанность вершин геометрии рассматривают как некий граф. В этом случае для выполнения разбиения можно оперировать размерами патчей. Само же разбиение проводится путем нахождения геодезического расстояния между точками [16]. Пример окончательного разбиения отображен на рис. 2 (см. третью сторону обложки).

Затем для каждого патча определяется формула генерации формы на основе обработанных ключевых геометрий:

$$\mathbf{S}_{model} = \overline{\mathbf{S}} + \sum_{i=1}^{m} a_i \mathbf{s}_i,$$

где $\overline{\mathbf{S}}$ — средняя форма региона среди всех ключевых геометрий, она представлена вектором, состоящим из координат вершин; $\mathbf{s}_i$ — вектор координат вершин для i-го ключевого кадра; a_i — вес смешивания для i-й ключевой геометрии; m — число ключевых геометрий. Таким образом получается итоговый вектор координат вершин $\mathbf{S}_{model}$, который определяет форму региона.

Оптимизируя веса смешивания, можно генерировать форму в пространстве ключевых геометрий. К данной форме применяют глобальные трансформации патча, а именно смещение и поворот. Итоговая формула генерации конкретного патча выглядит следующим образом:

$$\mathbf{S} = R(\mathbf{S}_{model}) + \mathbf{T},$$

где R — функция, отвечающая за глобальный поворот трехмерных вершин $\mathbf{S}_{model}$. Она может быть представлена применением матрицы поворота к $\mathbf{S}_{model}$, где поворот кодируется углами Эйлера. А параметр $\mathbf{T}$ представлен вектором смещений по координатам, отвечающим за глобальный перенос патча.

Итоговые параметры генерации патча представляют собой веса смешивания ключевых геометрий и параметры глобальной трансформации. Общее число параметров зависит от количества патчей.

Однако в описанном варианте патчи никак не связаны друг с другом, что влечет за собой разрывы в итоговой геометрии и артефакты. Для устранения этого недостатка в решаемую задачу оптимизации необходимо добавить функцию энергии, которая будет отвечать за связанность патчей. Данная функция для одного конкретного патча выглядит следующим образом:

$$E_{smooth} = \sum_{i=1}^{neighbors} w_i \left[\left\| (\mathbf{R} - \mathbf{R}_i) \right\|_2^2 + \left\| (\mathbf{T} - \mathbf{T}_i) \right\|_2^2 + \left\| (\mathbf{A} - \mathbf{A}_i) \right\|_2^2 \right],$$

где w_i — вес влияния соседнего патча на текущий; $\mathbf{R}$ — матрица поворота для текущего патча; $\mathbf{R}_i$ — матрица поворота соседнего патча; $\mathbf{T}$ — вектор смещений для текущего патча; $\mathbf{T}_i$ — вектор смещений для соседнего патча; $\mathbf{A}$ — вектор весов смешивания ключевых геометрий для текущего патча; $\mathbf{A}_i$ — вектор весов смешивания ключевых геометрий для соседнего патча; *neighbors* — число соседних патчей.

Основная идея связанности заключается в том, чтобы соседние патчи пытались иметь похожие веса формы и трансформации. Вес w_i определяется на этапе разбиения с помощью расчета расстояния до соседних патчей.

Таким образом, реализуется модель деформации, которая способна иметь гибкость локальных моделей, а также содержать априорную информацию о возможных формах лица.

Функция энергии. Точки и контуры

Большинство существующих алгоритмов в качестве основной «движущей силы» используют набор маркеров точек на лице [6, 7]. Рассматриваемый в настоящей публикации алгоритм — не исключение, так как на реальном производстве необходимо иметь гарантированную возможность получения результата даже в плохих условиях. К таковым, в частности, можно отнести ситуацию, когда освещение вышло из строя, а сцена была отыграна.

Для обобщения процедуры обработки точек и контуров необходимо провести дискретизацию контуров на точки. Дальнейшая обработка будет осуществляться однотипно как для оригинальных точек, так и для точек контуров.

Чтобы добавить функцию энергии, отвечающую за точки, необходимо на геометрии определить трехмерные точки, которые будут соответствовать маркерам. Для этого происходит ручное аннотирование указанных точек на трехмерной геометрии с нейтральным выражением лица данного актера. Определение маркеров на геометрии осуществляется с помощью текстуры. Итоговая формула энергии выглядит следующим образом:

$$E_{p2point2D} = \sum_{i=1}^{points} \left[\left\| projection(\mathbf{S}_i) - \mathbf{P}_i \right\|_2^2 \right],$$

где $\mathbf{S}_i$ — трехмерная вершина на геометрии; $\mathbf{P}_i$ — точка на изображении, соответствующая маркеру; *projection* — функция проецирования вершины $\mathbf{S}_i$ через видеокамеру; *points* — общее число точек.

Основная идея такого выбора функции энергии заключается в следующем: упомянутые выше точки геометрии проецируются на нужном кадре в пространство изображения через камеру. После этого результат сравнивается с целевыми точками маркеров на этом изображении.

Функция энергии. Стереоограничение

Чтобы улучшить результат регистрации лица, можно добавить дополнительное ограничение, отвечающее, насколько точно итоговая трехмерная модель соответствует целевой глубине сцены.

При наличии нескольких калиброванных камер очевидным решением этой задачи является получение трехмерного скана из изображений в целях последующей подгонки деформируемой геометрии под этот скан. Однако скан человеческого лица, полученный по стереокамерам, не отличается высоким уровнем качества. Нередко получение скана лица осуществляется теми же алгоритмами, что и получение сканов других объектов. В таких случаях алгоритм не содержит информацию о том, скан какого объекта вычисляется. Так как используется ограниченное число камер, появляется значительное множество артефактов. К тому же расчет полноценного скана с нуля вычислительно затратен. Рассматриваемый метод предлагает меньшее количество вычислений, а также учитывает априорную информацию о форме конкретного человеческого лица.

Метод основан на стереоограничении [14], итоговая формула энергии стереоограничения выглядит следующим образом:

$$E_{stereo} = \\ = \sum_{i=1}^{points} \left[\left\| I_1\left(projection_1\left(\mathbf{S}_i\right)\right) - I_2\left(projection_2\left(\mathbf{S}_i\right)\right) \right\|_2^2 \right],$$

где I_j — функция взятия значения цвета пикселя с изображения камеры j; $protection_j$ — функция проецирования вершины $\mathbf{S}_i$ через камеру j; $points$ — число вершин, наблюдаемых с обеих камер.

Идея выбора такой функции энергии состоит в том, что каждая вершина геометрии, которую видно с обеих камер, проецируется в каждую из камер. Получаются две двумерные точки: одна для первой камеры, другая для второй камеры. Далее происходит извлечение значения цвета пикселя из изображений, соответствующих камерам. На заключительном шаге значения цветов пикселей сравниваются. По сути, данная функция энергии отражает то, как решается задача построения трехмерного скана [17]. Для сравнения результаты нежесткой регистрации со стереоограничением и результаты без него представлены на рис. 3.

Основное преимущество метода дает лежащая в его основе модель деформации человеческого лица, имеющая априорную информацию о возможных изменениях геометрии лица конкретного человека, которая учитывается при решении задачи.

Функция энергии. Изображения

Функция энергии точек и стереоограничения дают возможность проводить нежесткую регистрацию, но они все еще не позволяют получать точность реконструкции лица на уровне пор кожи.

Для решения этой задачи можно использовать поиск оптического потока между рендером ны-

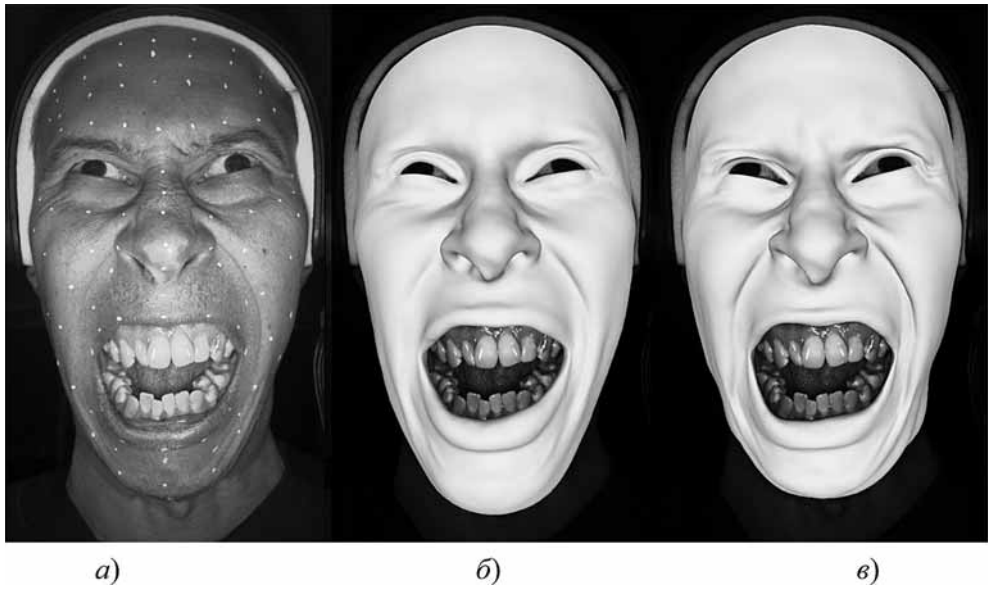

Рис. 3. Результат нежесткой регистрации:
а — целевое изображение; *б* — результат регистрации по точкам; *в* — результат регистрации по точкам и со стереоограничением

нешнего состояния геометрии с текстурой и целевым изображением для каждого ракурса. Такой способ применяется в обработке данных с видеограмметрических установок. Он основывается на использовании нейтральной текстуры для рендера каждого состояния геометрии, и единственная причина, по которой этот подход работает — большое число камер. При поиске потока между рендером и целевым изображением для каждой камеры итоговое решение складывается из всех результатов потока, тем самым получается среднее решение, которое направлено в нужную сторону.

В нашем случае использование оптического потока приводит к артефактам и неправильному решению, так как условия съемки гораздо хуже по сравнению со съемкой в видеограмметрической установке. Также существенный отрицательный эффект оказывает использование всего двух камер машинного зрения, вследствие чего оптический поток из одного ракурса начинает конфликтовать с оптическим потоком другого ракурса. Отмеченная трудность возникает также по причине независимой работы алгоритма поиска оптического потока для каждого ракурса. Он не содержит информации о другом ракурсе и об объектах сцены. В настоящей работе предлагается способ, который позволяет находить соответствия между рендером и целевым изображением каждой камеры согласованно между ракурсами с использованием априорной информации о ключевых геометриях актера.

Для применения этой техники сначала необходимо провести трассировку лучей через каждый пиксель целевого изображения, чтобы определить пары типа (*пиксель; трехмерная точка на геометрии)*.

Далее на этапе оптимизации каждая трехмерная точка актуального состояния геометрии проецируется обратно в пространство изображения. Затем вычисляется значение пикселя рендера в этой точке и сравнивается со значением целевого изображения в этой же точке.

Итоговая формула данной функции энергии выглядит следующим образом:

$$E_{photo} = \sum_{i=1}^{points} \left[\left\| I\left(projection\left(\mathbf{S}_i\right)\right) - \mathbf{P}_i \right\|_2^2 \right],$$

где $\mathbf{P}_i$ — значение цвета пикселя на целевом изображении, которое соответствует вершине $\mathbf{S}_i$; I — функция взятия значения цвета пикселя с рендера для спроецированной вершины геометрии *protection*($\mathbf{S}_i$); *points* — число пар (*пиксель; трехмерная точка на геометрии*).

Динамическая текстура

Для улучшения работы алгоритма при нахождении соответствий между рендером и целевым изображением можно генерировать *динамическую текстуру*. Генерация динамической текстуры требует набора пар вида (*геометрия; текстура*). В качестве таких пар возьмем ключевые геометрии и текстуры актера.

Основная идея метода генерации динамической текстуры состоит в обучении регрессора [18] определению текстуры конкретного региона на основе формы геометрии этого региона. Для этой цели необходимо разбить геометрию на регионы аналогично тому, как это сделано в модели деформации, после чего независимо обучить регрессор для каждого региона. В конечном счете для каждого текущего состояния геометрии в момент оптимизации будет отрабатывать общий регрессор, который будет генерировать для каждого региона соответствующую текстуру. Сравнительный пример рендера с динамической и нейтральной текстурами представлен на рис. 4.

Симуляция освещения

Необходимо также провести симуляцию освещения, так как сгенерированная текстура имеет равномерное освещение, которое получается в момент съемки ключевых выражений лица актера. Для решения этой задачи можно использовать сферические гармоники [8], которые являются стандартом игровой индустрии для быстрой симуляции освещения бесконечно далекого источника света. Освещение получается с помощью расчета специального базиса гармоник из векторов нормалей геометрии. «Золотой серединой» качества и стабильности является использование сферических гармоник третьего порядка [8]. Пример симуляции освещения дан на рис. 5.

Экстраполяция результатов

При осуществлении регистрации актера по данным шлема в силу использования всего двух фронтальных камерев возникает существенное ограничение в информации, однако ее можно реконструировать. В реальном производстве видеоматериала возникает необходимость отобразить человеческую голову целиком, включая шею, даже если захват таких регионов не осуществлялся.

Решением указанной задачи может стать «умная» экстраполяция результатов на основе примеров в виде ключевых геометрий. Основная идея

а) *б*) *в*)

Рис. 4. Целевое изображение (*а*); рендер геометрии с нейтральной текстурой (*б*) и рендер геометрии с динамической текстурой

а) *б*) *в*)

Рис. 5. Рендер геометрии:
а — с нейтральной текстурой; *б* — с динамической текстурой; *в* — с динамической текстурой и освещением

заключается в предсказывании формы невидимых частей головы и шеи на основе видимых регионов. Этот способ позволяет получить полную версию человеческой головы, в которой будут наблюдаться движения шеи и ушей, основанные на форме видимого региона.

Для реализации этого метода геометрия лица делится на две части: та, что видна хорошо, и та, что видна плохо. Необходимо вычислять деформационные градиенты [19] для каждого из участков. Деформационные градиенты используются вместо явных смещений точек геометрии, так как они инвариантны к перемещению.

На основе примеров из ключевых геометрий обучается *регрессор*, который предсказывает деформационные градиенты для невидимого региона. Для использования регрессора достаточно вычислить деформационные градиенты в текущем

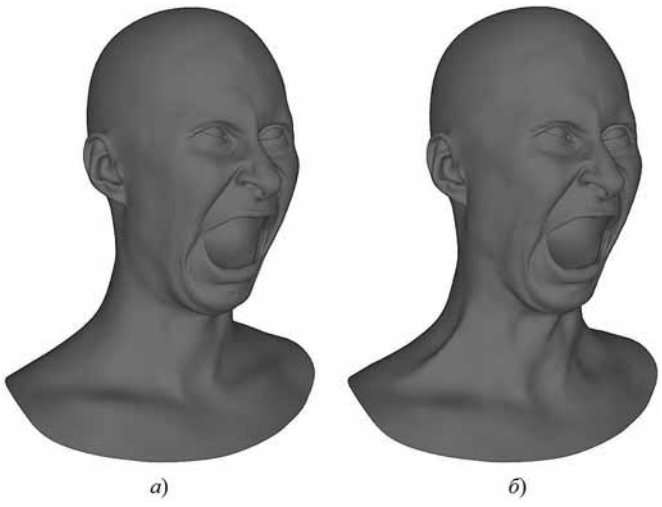

Рис. 6. Оригинальная геометрия после регистрации (*а*) и геометрия после экстраполяции (*б*)

состоянии геометрии, подать их на вход и применить полученные деформационные градиенты к невидимому участку.

Данный способ позволит получить форму невидимых регионов. Их качество будет ниже качества видимых регионов, однако форма будет полностью коррелировать с формами примеров ключевых геометрий — в большинстве случаев этого более чем достаточно. На рис. 6 можно заметить, каким образом добавилось движение шеи после экстраполяции.

Итоговый метод

После определения всех локальных функций энергии осталось сформулировать обобщающий метод. Финальная формула энергии, которая будет оптимизироваться, выглядит следующим образом:

$$E = E_{smooth} + E_{p2point2D} + E_{stereo} + E_{photo}.$$

Для решения общей задачи используется способ обработки, основанный на пирамидах изображений [20]. Он учитывает наличие функций энергии, отвечающих за подгонку рендера под целевое изображение и стереоограничения. Таким образом, обработка изображения будет осуществляться итерационно по уровням пирамид.

Вначале алгоритм принимает на вход набор ключевых геометрий и текстур, затем создает на их основе модель деформации. Модель деформации инициализируется стартовой геометрией. Это может быть как нейтральная модель актера, так и результат регистрации от предыдущего уровня пирамиды. Инициализация модели геометрией представляет собой отдельную задачу оптимизации [4], энергия которой выглядит следующим образом:

$$E_{p2point} = \sum_{i=1}^{points} \left\| v_i^t - v_i \right\|_2^2,$$

где v_i^t — вершина инициализирующей геометрии; v_i — вершина геометрии, полученной моделью деформации; *points* — число вершин, содержащихся в v.

Следует обратить внимание, что инициализирующая геометрия и геометрия модели деформации должны иметь одинаковую топологическую сетку. После решения задачи оптимизации модель деформации будет иметь набор переменных, которые позволяют сгенерировать геометрию, повторяющую геометрию инициализации.

В дальнейшем для текущего состояния геометрии выполняется генерация динамических текстур, которые основываются на ключевом наборе текстур. Генерация осуществляется на основе формы геометрии, после чего происходит симуляция освещения с помощью сферических гармоник.

Получившийся результат используется в задаче оптимизации. Согласно ее постановке необходимо минимизировать функцию энергии E, представленную функциями энергий по соответствию маркерам, контурам, целевому изображению и стереоограничениям. Задача оптимизации решается с помощью нелинейного метода наименьших квадратов, а именно метода Гаусса—Ньютона [21].

После решения задачи оптимизации ее результат может быть использован в качестве инициализации следующего уровня пирамиды либо, если это последний уровень, как итоговый результат.

Результаты тестирования

Представленный в работе метод тестировался на специальном шлеме с шестью видеокамерами. Такого рода шлем непригоден для использования в реальном производстве, однако в рамках исследования он позволяет тестировать нежесткую регистрацию. На основе данных всех камер рассчитывается трехмерный скан и происходит 4D-захват, а с помощью двух камер рассчитывается нежесткая регистрация лица по описанному алгоритму.

Для тестирования записывались две секвенции со скоростью 60 кадров с секунду. Первая — это поочередный набор заранее заготовленных экстремальных выражений. Длина данной секвенции — 3 мин, что эквивалентно 10 800 кадрам. Вторая секвенция — это набор *Harvard*

sentences[1], каждый из которых отыгрывается актером в спокойной, радостной и злой манерах. Суммарная длина второй секвенции составляет 7 мин, что эквивалентно 25 200 кадрам.

В рамках тестовых испытаний проведена обработка секвенций предлагаемым методом с двух камер, а также обработка методом нежесткой регистрации лица на основе сканов, построенных по всем шести камерам. Далее итоговые геометрии сравнивались по евклидову расстоянию. При этом в качестве эталона использовалась обработка на основе сканов, так как на текущий момент это наиболее точный метод нежесткой регистрации.

Суммарная средняя ошибка на всем лице составила 0,7 мм. Максимально она увеличивалась к краям в слабо наблюдаемых областях. В таких регионах ошибка могла доходить до 8 мм, что тем не менее является приемлемым результатом при ограниченном числе камер. Следует отметить, что даже в случае значительной ошибки результат попрежнему является «чистым» от артефактов, так как используется априорная информация о форме лица.

На рис. 7 (см. четвертую сторону обложки) приведен результат расчета по предлагаемому методу. В качестве динамической текстуры использовались цветные текстуры, а не черно-белые, как в момент нежесткой регистрации.

Заключение

Описан авторский метод нежесткой регистрации человеческого лица по изображениям со стереокамеры, позволяющий проводить захват мимики актера на съемочной площадке с использованием только двух камер, закрепленных на шлеме.

Данный метод использует новые техники улучшения качества захвата геометрии лица по изображениям при ограниченном числе данных, таких как локальная модель деформации на основе ключевых геометрий, а также стереоограничения вместо явного расчета трехмерных сканов. Благодаря его применению было достигнуто качество, которое сопоставимо с миллиметровой точностью 4D-захвата.

В качестве продолжения работы планируется масштабирование метода на большее число камер.

[1] Harvard sentences представляют собой набор образцов фраз. Это фонетически сбалансированные предложения, которые также используются для наблюдения за движениями рта актера, когда он говорит.

Список литературы

1. **Borshukov G., Piponi D., Larsen O.** et al. Image-based Facial Animation for “The Matrix Reloaded” // ACM SIGGRAPH computer graphics. N. Y.: ACM, 2003. P. 16. DOI: 10.1145/1198555.1198596.

2. **Garrido P., Valgaert L., Wu C**. et al. Reconstructing detailed dynamic face geometry from monocular video // ACM SIGGRAPH computer graphics. N. Y.: ACM, 2013. P. 1—10. DOI: 10.1145/2508363.2508380.

3. **Debevec P., Hawkins T., Tchou C.** et al. Acquiring the Reflectance Field of a Human Face // ACM SIGGRAPH computer graphics. N. Y.: ACM, 2020. P. 145—156. DOI: 10.1145/344779.344855.

4. **Hao L., Sumner R., Pauly M.** et al. Global correspondence optimization for non-rigid registration of depth scans // Eurographics Symposium on Geometry Processing. N. Y.: ACM, 2008. P. 1421—1430.

5. **Casas D., Tejera M., Guillemaut J.** et al. Interactive Animation of 4D Performance Capture // IEEE Transactions on Visualization and Computer Graphics. IEEE, 2013. P. 762—773. DOI: 10.1109/TVCG.2012.314.

6. **Hongwei X., Leijia D., Jianxing F.** et al. HIGH-QUALITY REAL TIME FACIAL CAPTURE BASED ON SINGLE CAMERA // Creative Commons Attribution 4.0 International. arXiv, 2021. P. 1—9. DOI: 10.48550/arXiv.2111.07556.

7. **Serra J., Moser L., McLean D.** et al. Simplified facial capture with head mounted cameras // ACM SIGGRAPH computer graphics. N. Y.: ACM, 2021. P. 1—2. DOI: 10.1145/3450623.3464637.

8. **Eugen A., Bouaziz S., Pauly M.** Dynamic 3D avatar creation from hand-held video input // ACM SIGGRAPH computer graphics. N. Y.: ACM, 2015. P. 1—14. DOI: 10.1145/2766974.

9. **Fyffe G., Nagano K., Huynh L.** et al. Multi-View Stereo on Consistent Face Topology // ACM SIGGRAPH computer graphics. N. Y.: ACM, 2017. P. 1—2. DOI: 10.1111/cgf.13127.

10. **Weinzaepfel P., Revaud J., Harchaoui Z.** et al. DeepFlow: Large displacement optical flow with deep matching // IEEE International Conference on Computer Vision. IEEE 2013. P. 1385—1392. DOI: 10.1109/ICCV.2013.175.

11. **Brox T., Malik J.** Large displacement optical flow: descriptor matching in variational motion estimation // IEEE Transactions on Pattern Analysis and Machine Intelligence. IEEE, 2011. P. 500—513. DOI: 10.1109/TPAMI.2010.143.

12. **Sun D., Roth S., Black M.** Secrets of optical flow estimation and their principles // IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE, 2010. P. 2432—2439. DOI: 10.1109/CVPR.2010.5539939.

13. **Beeler T., Hahn F., Bradley D.** et al. High-Quality Passive Facial Performance Capture using Anchor Frames // ACM SIGGRAPH computer graphics. N. Y.: ACM, 2011. P. 1—10. DOI: 10.1145/2010324.1964970.

14. **Борисов А. Д., Соломатин Д. И.** Стерео ограничения для задачи не ригидной регистрации лица // Труды молодых ученых факультета компьютерных наук ВГУ, 2022. С. 219—223.

15. **Борисов А. Д., Соломатин Д. И.** Разработка персонализированного детектора ключевых точек лица // Труды молодых ученых факультета компьютерных наук ВГУ, 2020. С. 16—22.

16. **Gabriel P., Cohen L.** Geodesic Methods for Shape and Surface Processing // Advances in Computational Vision and Medical Image Processing: Methods and Applications. Springer Verlag 2009. P. 29—56. DOI: 10.1007/978-1-4020-9086-8.

17. **Beeler T., Bickel B., Beardsley P.** et al. High-quality single-shot capture of facial geometry // ACM SIGGRAPH computer graphics. N. Y.: ACM, 2010. P. 1—9. DOI: 10.1145/1778765.1778777.

18. **Biancolini M.** Fast RBF for Engineering Applications // Fast radial basis functions for engineering applications. Springer International Publishing, 2018. P. 7—78.

19. **Sumner R.** Mesh Modification Using Deformation Gradients // Department of Electrical Engineering and Computer Science. 2005. P. 1—129.

20. **Andelson E, Anderson C., Bergen J.** et al. Pyramid methods in image processing // ACM SIGGRAPH computer graphics. N. Y.: ACM, 1983. P. 1—9.

21. **Madsen K., Nielsen H., Tingleff O.** Methods for Non-Linear Least Square Problems. Cobenhavn, Technical University of Denmark. IMM, 2004. 1—30.

Non-Rigid Registration of a Human Face from Images from a Stereo Camera

A. D. Borisov, Postgraduate Student, radiatus@yandex.ru, **S. D. Makhortov,** Professor, Head of Department of Programming and Information Technologies, msd_exp@outlook.com, Voronezh State University, Voronezh, 394018, Russian Federation

Corresponding author:

Sergey D. Makhortov, Professor, Head of Department of Programming and Information Technologies, Voronezh State University, Voronezh, 394018, Russian Federation
E-mail: msd_exp@outlook.com

Received December 17, 2022
Accepted December 26, 2022

Capturing the facial expressions of an actor on the set to further transfer the scene to the digital space has become one of the most important tasks in the field of computer graphics. Most of the methods for solving this problem have a lower quality of reconstruction in comparison with the capture, which is carried out in videogrammetric installations. The reason is the inability to calculate high-quality three-dimensional scanned copies ("scans"). This article highlights the problematic issues of existing approaches and proposes a solution for capturing an actor's facial expressions from images from a helmet's stereo camera, which is comparable to stationary capture. To obtain this level of accuracy, it is proposed to use a local deformation model based on the geometries of key facial expressions, as well as stereo restriction, instead of explicitly calculating a 3D scan.

Keywords: *soft registration, stereo camera, computer graphics, motion capture, digital doubles*

For citation:

Borisov A. D., Makhortov S. D. Non-Rigid Registration of a Human Face from Images from a Stereo Camera, *Programmnaya Ingeneria,* 2023, vol. 14, no. 2, pp. 82—92. DOI: 10.7587/prin.14.82-92 (in Russian).

References

1. **Borshukov G., Piponi D., Larsen O**. et al. Image-based Facial Animation for «The Matrix Reloaded», *ACM SIGGRAPH computer graphics*, N. Y., *ACM*, 2003, pp. 16. DOI: 10.1145/1198555.1198596.

2. **Garrido P., Valgaert L., Wu C.** et al. Reconstructing detailed dynamic face geometry from monocular video, *ACM SIGGRAPH computer graphics,* N. Y., ACM, 2013, pp. 1—10. DOI: 10.1145/2508363.2508380.

3. **Debevec P., Hawkins T., Tchou C.** et al. Acquiring the Reflectance Field of a Human Face, *ACM SIGGRAPH computer graphics.* N. Y., ACM, 2020, pp. 145—156. DOI: 10.1145/344779.344855.

4. **Hao L., Sumner R., Pauly M.** et al. Global correspondence optimization for non-rigid registration of depth scans, *Eurographics Symposium on Geometry Processing,* N. Y., ACM, 2008, pp. 1421—1430.

5. **Casas D., Tejera M., Guillemaut J.** et al. Interactive Animation of 4D Performance Capture, *IEEE Transactions on Visualization and Computer Graphics,* IEEE 2013, pp. 762—773. DOI: 10.1109/TVCG.2012.314.

6. **Hongwei X., Leijia D., Jianxing F.** et al High-Quality Real TIme Facial Capture Based on Single Camera, *Creative Commons Attribution 4.0 International*, arXiv, 2021, pp. 1—9. DOI: 10.48550/arXiv.2111.07556.

7. **Serra J., Moser L., McLean D.** et al. Simplified facial capture with head mounted cameras, *ACM SIGGRAPH computer graphics*, N. Y., ACM, 2021, pp. 1—2. DOI: 10.1145/3450623.3464637.

8. **Eugen A., Bouaziz S., Pauly M.** Dynamic 3D avatar creation from hand-held video input, *ACM SIGGRAPH computer graphics*, N. Y., ACM, 2015, pp. 1—14. DOI: 10.1145/2766974.

9. **Fyffe G., Nagano K., Huynh L.** et al. Multi-View Stereo on Consistent Face Topology, *ACM SIGGRAPH computer graphics*, N. Y., ACM, 2017, pp. 1—2. DOI: 10.1111/cgf.13127.

10. **Weinzaepfel P., Revaud J., Harchaoui Z.** et al. DeepFlow: Large displacement optical flow with deep matching, *IEEE Inter-*

national Conference on Computer Vision, IEEE 2013, pp. 1385—1392. DOI: 10.1109/ICCV.2013.175.

11. **Brox T., Malik J.** Large displacement optical flow: descriptor matching in variational motion estimation, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* IEEE 2011, pp. 500—513. DOI: 10.1109/TPAMI.2010.143.

12. **Sun D., Roth S., Black M.** Secrets of optical flow estimation and their principles, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, 2010, pp. 2432—2439. DOI: 10.1109/CVPR.2010.5539939.

13. **Beeler T., Hahn F., Bradley D.** et al. High-Quality Passive Facial Performance Capture using Anchor Frames, *ACM SIGGRAPH computer graphics*, N. Y.: ACM, 2011, pp. 1—10. DOI: 10.1145/2010324.1964970.

14. **Borisov A. D., Solomatin D. I.** Stereo Constraints for the Problem of Non-Rigid Face Registration, *Trudy molodyh uchyonyh fakulteta kompyuternyh nauk VGU,* 2022, pp. 219—223 (in Russian).

15. **Borisov A. D., Solomatin D. I.** Razrabotka personalizirovannogo detektora kluchevyh nocher litsa, *Trudy molodyh uchyonyh fakulteta kompyuternyh nauk VGU,* 2020, pp. 16—22 (in Russian).

16. **Gabriel P., Cohen L.** Geodesic Methods for Shape and Surface Processing, *Advances in Computational Vision and Medical Image Processing: Methods and Applications*, Springer Verlag, 2009, pp. 29—56. DOI: 10.1007/978-1-4020-9086-8.

17. **Beeler T., Bickel B., Beardsley P.** et al. High-quality single-shot capture of facial geometry, *ACM SIGGRAPH computer graphics,* N. Y., ACM, 2010, pp. 1—9. DOI: 10.1145/1778765.1778777.

18. **Biancolini M.** Fast RBF for Engineering Applications, *Fast radial basis functions for engineering applications*, Springer International Publishing, 2018, pp. 7—78.

19. **Sumner R.** Mesh Modification Using Deformation Gradients, Department of Electrical Engineering and Computer Science, 2005, pp. 1—29.

20. **Andelson E, Anderson C., Bergen J.** et al. Pyramid methods in image processing, ACM SIGGRAPH computer graphics, N. Y., ACM, 1983, pp. 1—9.

21. **Madsen K., Nielsen H., Tingleff O.** *Methods for Non-Linear Least Square Problems,* Cobenhavn, Technical University of Denmark, 2004, pp. 1—30.

ИНФОРМАЦИЯ

УДК 004.5
DOI: 10.17587/prin.14.93-100

Д. И. Читалов, мл. науч. сотр., cdi9@yandex.ru,
Федеральное государственное бюджетное учреждение науки Южно-Уральский федеральный научный центр минералогии и геоэкологии Уральского отделения Российской академии наук, Миасс, Ильменский заповедник

О разработке модуля для решателя coupledPoroFoam пакета OpenFOAM

Поступила в редакцию 09.11.2022
Принята к публикации 30.11.2022

Описаны основные положения процесса разработки программного модуля с графической составляющей для управления численным моделированием на базе решателя coupledPoroFoam и его подключение к базовой версии графической оболочки для пакета прикладных программ OpenFOAM. Особое внимание уделено определению целей и задач разработки, выбору необходимых технологических средств. С помощью диаграмм описана структура модуля и логика его работы. Приведены результаты исследования, положения, определяющие техническую и практическую значимость работы. На основе созданного модуля проведено численное моделирование одной из базовых задач механики сплошных сред, представлены результаты эксперимента.

Ключевые слова: *численное моделирование, механика сплошных сред, графический интерфейс пользователя, OpenFOAM, язык программирования Python, открытое программное обеспечение, решатель coupledPoroFoam, библиотека PyQt*

Для цитирования:

Читалов Д. И. О разработке модуля для решателя coupledPoroFoam пакета OpenFOAM // Программная инженерия. 2023. Том 14, № 2. С. 93—100. DOI: 10.17587/prin.14.93-100.

Введение

Механика сплошных сред (МСС) — одна из древнейших наук, она предполагает изучение процессов, связанных с агрегатными состояниями веществ, включая жидкости, газы, твердые тела. Для каждого из веществ в МСС выделяется соответствующий раздел. Изучение закономерностей процессов, относящихся к каждому разделу, позволяет прогнозировать поведение веществ при взаимодействии с окружающей средой и объектами. Это актуально при проектировании продукции отраслей тяжелого машиностроения, когда важно прогнозирование поведения изделий в различных условиях эксплуатации. На этапе проектирования осуществляется численное моделирование задач МСС, где каждая задача — это конкретный эксперимент с набором характеристик, когда объектами изучения являются жидкие и газообразные среды, электромагнитные поля, деформируемые твердые тела и т. д. К задачам МСС можно отнести, например, перемещение тела в жидкости и газе, механику деформируемого твердого тела и др.

Одним из популярных программных решений для подготовки численных моделей является пакет прикладных программ OpenFOAM [1]. Это открытая программная платформа, включающая компоненты для моделирования большинства задач МСС. Для генерации численных моделей в OpenFOAM предусмотрены встроенные программы-решатели, каждая из которых ориентирована на определенный блок задач МСС. Помимо набора решателей в дистрибутив OpenFOAM интегрированы служебные утилиты, обеспечивающие пре- и постпроцессинг численного моделирования. Это этапы, на которых, например, формируется и визуализируется расчетная сетка, определяются свойства задачи МСС и т. д.

Востребованность OpenFOAM обусловлена тем обстоятельством, что пакет предоставляется на некоммерческой основе и успешно распространяется для решения задач как на зарубежных, так и на

отечественных предприятиях. В то же время пакет имеет заметные недостатки, в частности, в нем отсутствует встроенный графический интерфейс пользователя, а документация предоставляется на английском языке. Перечисленные недостатки могут влиять на эффективность работы с OpenFOAM, повышают трудоемкость его использования, влияют на качество итоговых численных моделей.

По причине отсутствия встроенной графической оболочки специалист вынужден управлять всеми этапами численного эксперимента вручную. Ему самостоятельно приходится создавать директорию расчетного случая, в ней создавать и заполнять файлы-словари с расчетными параметрами, вручную запускать служебные утилиты и программы-решатели. Для устранения указанных недостатков пакета OpenFOAM международные коллективы исследователей предложили оригинальные программные продукты, такие как Salome [2], Helyx-OS [3] и Visual-CFD [4]. Однако и они оказались не лишены недостатков, например, в ряде случаев необходимо приобретение лицензии и услуг технических специалистов, а интерфейс реализован только на английском языке.

Автором настоящей статьи в 2016 г. представлена собственная версия графической оболочки [5] для OpenFOAM. Это свободно-распространяемый программный продукт с оконным интерфейсом, реализованном в русскоязычной версии. Открытый исходный код данного приложения позволяет создавать и интегрировать дополнительные программные модули, что расширяет перечень возможностей графической оболочки. К настоящему времени было реализовано несколько таких расширений. Полученные результаты описаны и опубликованы в работах [6—8].

В данной статье рассмотрен процесс разработки и подключения к базовой версии графической оболочки нового программного модуля для работы с программой-решателем coupledPoroFoam. Перечень этапов разработки включает исследование особенностей применения указанного решателя, постановку целей и задач, подготовку алгоритмов продукта и перенос их на язык программирования, доработку графической составляющей продукта и подключение программы к базовой версии графической оболочки.

Назначение решателя coupledPoroFoam

Ключевые компоненты дистрибутива OpenFOAM реализованы на базе языка программирования C++. Для построения моделей, соответствующих задачам МСС, применяют многочисленные решатели (солверы), написанные на базе C++. Поскольку OpenFOAM является открытым пакетом, перечень доступных решателей обновляется с выходом очередной версии продукта.

На практике для моделирования задач нелинейного течения жидкости широко используется метод конечных объемов. Он также находит свое применение при решении задач динамики твердого тела в среде. Подобные эксперименты проводят в большинстве отраслей машиностроения, где изучают особенности поведения твердых тел при взаимодействии с окружающими средами. Отмеченный выше решатель coupledPoroFoam ориентирован на численные эксперименты, в которых выявляются закономерности при взаимодействии твердого скелета и поровой жидкости.

Прежде всего речь идет о закономерностях изменения порового давления, т. е. гидродинамического давления, возникающего в водонасыщенных поровых структурах при их уплотнении за счет влияния собственного веса и внешних условий. В представленном решателе реализован новый алгоритм моделирования такого класса задач, который обеспечивает высокую точность соответствия реальному процессу. В данном случае на этапе препроцессинга осуществляется подготовка стандартной блочной сетки с помощью программы-решателя blockMesh.

Кроме того, в экспериментах рассматриваемого класса специалист имеет возможность варьировать граничные условия и выявлять больший спектр закономерностей при исследовании напряжений твердого тела. Область применения указанного решателя — проектирование изделий в области судостроения, сельскохозяйственного и энергетического машиностроения, а также авиастроения. В частности, данный решатель применяется на этапе проектирования продукции авиакосмического строения на предприятии АО «ГРЦ Макеева».

Постановка цели и задач исследования

Цель исследования, результаты которого представлены далее, заключается в изучении процесса взаимодействия пользователя с решателем coupledPoroFoam при проведении численного моделирования задач МСС и в разработке программного средства для эффективного управления этим процессом. Программное средство, реализующее такое управление, должно включать соответствующие графические компоненты. В перечень конкретных шагов, направленных на достижение поставленной цели, входят следующие задачи:

1) исследовать структуру расчетного случая, создаваемого при работе с решателем coupledPoroFoam,

и проанализировать входящие в этот расчетный случай служебные файлы с расчетными параметрами задачи МСС;

2) подготовить и программно реализовать макеты экранных форм, необходимых для указания расчетных параметров для каждого служебного файла-словаря;

3) сформировать алгоритмы, необходимые для реализации логической составляющей программного модуля — автоматического создания и заполнения служебных файлов расчетного случая параметрами, задаваемыми через экранные формы приложения;

4) сформировать алгоритмы, обеспечивающие программный запуск решателя coupledPoroFoam;

5) сформировать алгоритмы валидации расчетных параметров и алгоритмы проверки комплектности расчетного случая перед запуском численного моделирования;

6) сформированные алгоритмы реализовать с помощью выбранного языка программирования и интегрировать в базовую версию графической оболочки с сохранением ее исходных функциональных возможностей.

Решение этих задач должно обеспечить достижение поставленной цели с сохранением текущих функциональных возможностей приложения и с сохранением многослойной архитектуры.

Реализованный программный модуль ориентирован на замену традиционного подхода к постановке эксперимента в OpenFOAM на альтернативный. В рамках замены предполагается полный отказ от использования консоли для управления численным экспериментом и переход к графическому интерфейсу.

Средства разработки

Полный цикл разработки программного приложения включает такие этапы, как создание модели в виде блок-схем, описывающих логику взаимодействия пользователя с приложением, разработку алгоритмов и написание исходного кода, его отладку и тестирование. Для этого применяются инструментальные средства, в том числе дизайнеры блок-схем, интегрированные среды разработки и другие служебные сервисы. Благодаря блок-схемам можно представить взаимосвязь компонентов приложения и логику его функционирования. При разработке программных продуктов главным образом определяются технологии, которые будут положены в основу их реализации, в первую очередь языки программирования и дополнительные библиотеки. Эти технологии приведены далее.

1. Технология реализации модели и алгоритмов представляет собой компонент программы, обеспечивающий взаимодействие пользователя с внутренними данными. Это программно-аппаратная часть, скрытая от пользователя. Реализуется средствами одного из высокоуровневых языков программирования. Так как программно-аппаратная часть базовой версии графической оболочки функционирует на базе языка программирования Python, автором принято решение продолжить работу с данной технологией. Язык Python обеспечивает высокую скорость разработки программных приложений, имеет простой и понятный синтаксис, он гибок и позволяет подключать большой перечень сторонних библиотек и фреймворков [9, 10].

2. Технология реализации графической составляющей, как и для базовой версии графической оболочки, предполагает все оконные формы модуля и их компоненты реализовать средствами библиотеки PyQt [11]. Она совместима с Python-продуктами, обладает интуитивно понятным синтаксисом, позволяет создавать традиционные экранные формы для графических интерфейсов.

3. Технология реализации хранения данных предполагает продолжить использование СУБД SQLite и сохранение вводимых пользователем расчетных параметров в таблицах реляционной базы данных [12]. Такой подход позволяет обеспечить сохранение и восстановление расчетных параметров для дальнейшего их редактирования, т. е. изменения расчетных условий задачи МСС.

Для работы с представленным программным модулем пользователь должен располагать вычислительным устройством под управлением ОС Linux. Кроме того, необходимо наличие самого пакета OpenFOAM. Вычислительное устройство должно иметь достаточно ресурсов для проведения эксперимента с помощью этого пакета.

К дополнительным средствам, обеспечивающим цикл разработки, можно отнести представленные далее компоненты.

- Дизайнер блок-схем. Принято решение использовать онлайн-построитель draw.io, благодаря веб-интерфейсу он функционирует на любой платформе и не требует установки и настройки [13].
- Интегрированная среда разработки. Среди Python-разработчиков распространение и востребованность получила среда PyCharm [14]. Она обладает привлекательным интерфейсом, простотой настройки и необходимыми опциями для набора кода, его отладки, тестирования и запуска.

Важным преимуществом выбранного стека является открытый характер использования перечисленных продуктов. Они предоставляются на некоммерческой основе, не требуют дополнитель-

ных настроек и функционируют под управлением большинства операционных систем. Выбранный стек обеспечивает совместимость программного модуля с базовой версией графической оболочки.

Архитектура модуля и логика его использования

В основе базовой версии графической оболочки лежит модульный принцип организации. Модуль представляет собой набор логически связанных блоков (служебных файлов). Благодаря данному принципу упрощается процесс программирования. Над разработкой модулей могут работать отдельные коллективы специалистов. Каждый модуль в данном случае реализуется в виде блока скриптов на языке программирования Python, служебных файлов с кодом экранных форм, а также дополнительных файлов, обеспечивающих, например, запуск программ-решателей и вспомогательных утилит OpenFOAM. Модульный принцип организации программного приложения соответствует многослойной архитектуре, которая, как правило, представлена в настольных программных продуктах.

Рисунок 1 демонстрирует структурную схему графической оболочки с интегрированным модулем для работы с решателем coupledPoroFoam.

- Директория windows содержит служебные файлы с макетами экранных форм, созданных на базе выражений библиотеки PyQt. По сути, это оболочка каждой формы без учета элементов управления, т. е. конкретных виджетов, через которые пользователь взаимодействует с программой.
- Директория forms содержит служебные файлы, каждый из которых включает набор элементов управления для отдельной формы, в том числе кнопки, поля ввода, выпадающие списки и т. д. При визуализации каждой формы набор элементов управления подгружается в оболочку формы.
- Директория threads содержит дополнительные файлы, обеспечивающие многопоточность работы модуля. За счет возможностей языка Python обеспечивается параллельный запуск нескольких утилит или программ-решателей, что положительно влияет на производительность приложения.
- Директория functions включает Python-файлы с кодом служебных функций, обеспечивающих визуализацию служебных сообщений, подсказок и т. д.

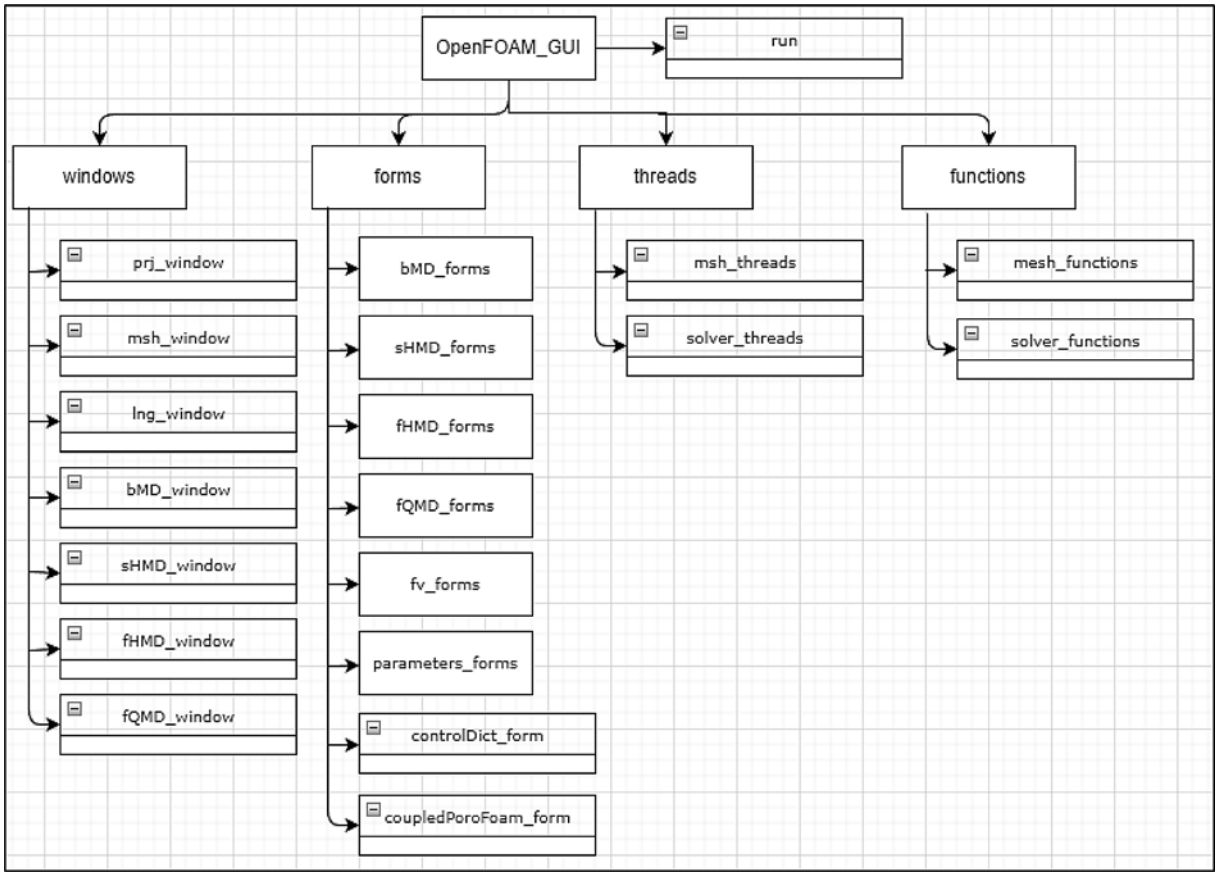

Рис. 1. Структура графической оболочки OpenFOAM_GUI

• Лаунчер run обеспечивает запуск приложения с визуализацией интерфейса.

На рис. 2 представлена блок-схема, описывающая логику работы пользователя с графической оболочкой для OpenFOAM с учетом интеграции модуля для решателя coupledPoroFoam. Благодаря предложенному программному решению пользователь заменяет традиционный подход управления численным экспериментом посредством командной строки на более удобный и эффективный — управление посредством привычного оконного интерфейса. В рамках данного подхода за создание расчетного случая и заполнение входящих в него служебных файлов отвечает программа. Также она обеспечивает запуск необходимых консольных утилит и программ-решателей. Задача пользователя заключается только в том, что он отдает команды программе, а она выполняет соответствующие действия.

Важно отметить, что перед запуском непосредственно численного моделирования на базе решателя coupledPoroFoam пользователь осуществляет этап препроцессинга, на котором генерируется расчетная сетка, определяются граничные условия и задаются ключевые расчетные параметры задачи. Возможности графической оболочки позволяют при необходимости вернуться к любому этапу численного эксперимента, выполнить корректировку параметров и повторно сгенерировать сетку или запустить повторно решение задачи МСС. Процесс численного моделирования завершается постпроцессингом, в рамках которого итоговая численная модель визуализируется для оценки степени ее соответствия заявленным требованиям.

Рис. 2. Логика работы модуля

Результаты исследования

Автором достигнута обозначенная цель исследования. Разработан и интегрирован в базовую версию графической оболочки программный модуль для управления процессом численного моделирования на базе программы-ре-

шателя coupledPoroFoam. Модуль включает графические и программные средства для управления всеми этапами численного моделирования на базе указанного решателя. В рамках работы над достижением поставленной цели автором выполнен весь комплекс обозначенных ранее задач.

- На основе документации и демонстрационных примеров [15, 16] исследован алгоритм работы пользователя с расчетным случаем при проведении численных исследований на базе решателя coupledPoroFoam, проанализированы входящие в него служебные файлы и допустимые расчетные параметры.
- Подготовлены макеты экранных форм в соответствии со структурой служебных файлов, входящих в расчетный случай.
- Подготовлены и перенесены на программный код алгоритмы, отвечающие за процесс автоматического формирования структуры служебных файлов с расчетными параметрами.
- Подготовлены и перенесены на программный код алгоритмы запуска решателя coupledPoroFoam.
- Реализованы алгоритмы валидации расчетных параметров для задачи МСС.

Весь перечень задач выполнен в привязке к базовой версии графической оболочки, которая модифицирована с учетом обозначенных требований. Работоспособность обновленной версии приложения протестирована на примере одной из фундаментальных задач МСС, предполагающей моделирование свойств твердого пористого тела при взаимодействии с жидкой средой. На рис. 3 (см. четвертую сторону обложки) приведены результаты эксперимента на этапе постпроцессинга, т. е. визуализации итоговой численной модели средствами дополнительного служебного пакета ParaView [17].

Заключение

По итогам выполненного исследования автор пришел к выводу, что модульный принцип является весьма эффективным при реализации настольных программных продуктов. Он позволяет подключать к приложению сторонние компоненты и расширять список его возможностей в рамках численного моделирования. В текущей работе речь идет о реализации графических и программных средств для управления численным экспериментом посредством программы-решателя coupledPoroFoam [18].

Благодаря отказу от применения командной строки и переходу к привычному оконному интерфейсу пользователь получает важные преимущества, такие как повышение удобства в процессе управления препроцессингом, решением и постпроцессингом, а также снижение вероятности совершить ошибку на одном из этапов. Последний пункт обусловлен тем, что модуль содержит систему валидации вводимых параметров, которая обеспечивает фильтрацию указываемых значений. Таким образом, все служебные файлы с расчетными параметрами создаются по строго определенному алгоритму. Пользователь не может установить значение неверного типа данных.

Главный вывод заключается в том, что связка технологий Python и PyQt является эффективной при разработке программных приложений с графическим интерфейсом. В связи с регулярным обновлением дистрибутива OpenFOAM могут быть реализованы модули, аналогичные представленному, и для других утилит и решателей, что позволит поддерживать актуальность разработанного продукта, сделать его максимально соответствующим существующим аналогам с сохранением важных преимуществ — русскоязычного интерфейса и открытого исходного кода.

Благодаря сериализации характеристик задачи МСС с помощью реляционных баз данных обеспечивается возможность сохранения и восстановления расчетных параметров для последующего редактирования и запуска численного моделирования уже применительно к новым условиям. Это также повышает удобство пользователя и позволяет оперативно переключаться между различными блоками значений одних и тех же расчетных параметров.

Как и базовая версия графической оболочки, так и реализованный модуль могут применяться в рамках численного моделирования большинства проблем МСС, в которых требуются возможности решателя coupledPoroFoam. Это преимущественно отрасли тяжелого машиностроения, где на этапе проектирования изделий требуется создавать их модели и анализировать изменение характеристик под влиянием внешних условий и среды.

Список литературы

1. **OpenFOAM.** The open source CFD toolbox. URL: https://www.openfoam.com/ (дата обращения 04.11.2022).
2. **Salome.** The Open Source integration Platform for Numerical Simulation. URL: https://www.salome-platform.org/ (дата обращения 04.11.2022).
3. **Helyx-OS.** Open-Source GUI for OpenFOAM. URL: https://engys.com/products/helyx-os (дата обращения 04.11.2022).
4. **Visual-CFD.** URL: https://www.esi-group.com/products/computational-fluid-dynamics (дата обращения 04.11.2022).
5. **Читалов Д. И., Меркулов Е. С., Калашников С. Т.** Разработка графического интерфейса пользователя для про-

граммного комплекса OpenFOAM // Программная инженерия. 2016. Том 7, № 12. С. 568—574. DOI: 10.17587/prin.7.568-574.

6. **Читалов Д. И., Калашников С. Т.** Разработка модуля для реализации зеркального отображения расчетных сеток вокруг заданной плоскости в графическом интерфейсе пользователя платформы OpenFOAM // Программная инженерия. 2019. Том 10, № 7—8. С. 297—304. DOI: 10.17587/prin.10.297-304.

7. **Читалов Д. И.** О разработке модуля для реализации движения и топологического изменения расчетных сеток и его интеграции в графическую оболочку для платформы OpenFOAM // Программная инженерия. 2020. Том 11, № 2. С. 108—114. DOI: 10.17587/prin.11.108—114.

8. **Читалов Д. И.** Разработка модуля для измельчения ячеек расчетных сеток в нескольких направлениях и его интеграция в GUI для программной среды OpenFOAM //Системы и средства информатики. 2020. Т. 30, № 3. С. 133—144. DOI: 10.14357/08696527200312.

9. **Python 3.7** documentation. URL: https://docs.python.org/3.7/ (дата обращения 04.11.2022).

10. **Прохоренок Н. А.** Python 3 и PyQt. Разработка приложений. СПб.: БХВ-Петербург, 2012. 704 с.

11. **PyQt5** Reference Guide. URL: http://pyqt.sourceforge.net/Docs/PyQt5/ (дата обращения 04.11.2022).

12. **SQLite.** URL: https://www.sqlite.org/index.html (дата обращения 04.11.2022).

13. **Free** online diagram software. URL: https://www.draw.io (дата обращения 04.11.2022).

14. **PyCharm**. URL: https://www.jetbrains.com/ru-ru/pycharm/ (дата обращения 04.11.2022).

15. **OpenFOAM.** User Guide. URL: http://foam.sourceforge.net/docs/Guides-a4/OpenFOAMUserGuide-A4.pdf (дата обращения 04.11.2022).

16. **OpenFOAM.** Tutorial Guide. URL: http://openfoam.com/documentation/tutorial-guide/index.php (дата обращения 04.11.2022).

17. **ParaView.** URL: https://www.paraview.org/ (дата обращения 04.11.2022).

18. **OpenFOAM_GUI**. URL: http://github.com/DmitryChitalov/OpenFOAM_GUI (дата обращения 04.11.2022).

On the Development of a Module for the coupledPoroFoam Solver of the OpenFOAM Package

D. I. Chitalov, Junior Researcher, cdi9@yandex.ru,
South Urals Federal Research Centre of Mineralogy and Geoecology of the UB RAS,
Chelyabinsk region, Miass, Ilmen reserve, 456317, Russian Federetion

Corresponding author:

Dmitry I. Chitalov, Junior Researcher, South Urals Federal Research Centre of Mineralogy and Geoecology of the UB RAS, Chelyabinsk region, Miass, Ilmen reserve, 456317, Russian Federation
E-mail: cdi9@yandex.ru

Received on November 09, 2022
Accepted on November 30, 2022

The main provisions of the process of developing a software module with a graphical component to control numerical simulation based on the coupledPoroFoam solver and its connection to the basic version of the graphical shell for the OpenFOAM application package are described. Similar applications are analyzed, the relevance of the problem is formulated. A description of the coupledPoroFoam solver and a list of its application areas are given. The process of numerical simulation of problems of continuum mechanics based on the presented solver is studied. Particular attention is paid to the definition of the goals and objectives of development, the choice of the necessary technological means. Using diagrams, the architecture of the module and the logic of its operation are described. The purpose of each component of the application and the step-by-step process of the user working with the module are given. The results of the study, provisions that determine the technical and practical significance of the work are described. Based on the created module, a numerical simulation of one of the basic problems of continuum mechanics was carried out, and the results of the experiment were presented. The achieved tasks are defined, the final conclusions about the possibility of using the selected technologies for the development of software applications are formulated.

Keywords: *numerical simulation, continuum mechanics, graphical user interface, OpenFOAM, Python, open source software, coupledPoroFoam solver, PyQt*

For citation:

Chitalov D. I. On the Development of a Module for the coupledPoroFoam Solver of the OpenFOAM Package, *Programmnaya Ingeneria,* 2023, vol. 14, no. 2, pp. 93—100. DOI: 10.17587/prin.14.93-100 (in Russian).

References

1. **OpenFOAM.** The open source CFD toolbox, available at: https://www.openfoam.com/ (date of access 04.11.2022).
2. **Salome.** The Open Source Integration Platform for Numerical Simulation, available at: http://www.salome-platform.org (date of access 04.11.2022).
3. **Helyx-OS.** Open Source GUI for OpenFOAM, available at: http://engys.com/products/helyx-os (date of access 04.11.2022).
4. **Visual-CFD** for OpenFOAM. CFD simulation software aimed at solving complex flow applications, available at: http://www.esi-group.com/software-solutions/virtual-environment/cfd-multiphysics/visual-cfd-openfoam (date of access 04.11.2022).
5. **Chitalov D. I., Merkulov E. S., Kalashnikov S. T.** Development of a graphical user interface for the OpenFOAM toolbox, *Programmnaya ingeneria*, 2016, no. 12, pp. 568—574. DOI: 10.17587/prin.7.568-574 (in Russian).
6. **Chitalov D. I., Kalashnikov S. T.** Development of a module for the implementation of mirroring of computational meshes around a given plane in the graphical user interface of the openfoam platform, *Programmnaya ingeneria*, 2019, no. 7-8, pp. 297—304. DOI: 10.17587/prin.10.297-304 (in Russian).
7. **Chitalov D. I.** On the development of a module for the implementation of motion and topological changes in computational meshes and its integration into the graphical shell for the openfoam platform, *Programmnaya ingeneria*, 2020, no. 2, pp. 108—114. DOI: 10.17587/prin.11.108-114 (in Russian).
8. **Chitalov D. I.** Development of a module for grinding cells of computational meshes in several directions and its integration into gui for the openfoam software environment, *Sistemy i sredstva informatiki*, 2020, no 3, pp. 133—144. DOI: 10.14357/08696527200312 (in Russian).
9. **Python 3.7** documentation, available at: http://docs.python.org/3.7/ (date of access 04.11.2022).
10. **Prohorenok N. A.** Python 3 and PyQt. Application Development. St. Petersburg: BHV-Petersburg, 2012. 704 p. (in Russian).
11. **PyQt5** Reference Guide, available at: http://pyqt.sourceforge.net/Docs/PyQt5/ (date of access 04.11.2022).
12. **SQLite**, available at: https://www.sqlite.org/index.html (date of access 04.11.2022).
13. **Free** online diagram software, available at: https://www.draw.io(date of access 04.11.2022).
14. **PyCharm,** available at: https://www.jetbrains.com/ru-ru/pycharm/ (date of access 04.11.2022).
15. **The OpenFOAM** Foundation. User Guide, available at: http://foam.sourceforge.net/docs/Guides-a4/OpenFOAMUserGuide-A4.pdf (date of access 04.11.2022).
16. **OpenFOAM.** Tutorial Guide, available at: http://openfoam.com/documentation/tutorial-guide/index.php (date of access 04.11.2022).
17. **ParaView.** Open-source, multi-platform data analysis and visualization application, available at: http://www.paraview.org/ (date of access 04.11.2022).
18. **OpenFOAM_GUI,** available at: http://github.com/Dmitry-Chitalov/OpenFOAM_GUI (date of access 04.11.2022).

КОНФЕРЕНЦИЯ

ТРЕНДЫ И ПОТЕНЦИАЛ РОССИЙСКОГО СЕКТОРА КИБЕРБЕЗОПАСНОСТИ-2023

25 мая 2023 г., Москва, Swissotel Красные Холмы

В последнее время становится популярным комплексный подход к безопасности, который затрагивает как программный уровень, так и уровень взаимодействий и регламентов. О том, какой подход будет доминирующим в ближайшем будущем, 25 мая 2023 г. расскажут эксперты в сфере отечественного сектора кибербезопасности.

Подробности: https://eventtoday.biz/cybersecurity

ООО "Издательство "Новые технологии". 107076, Москва, ул. Матросская Тишина, д. 23, стр. 2
Технический редактор *Е. В. Конова*. Корректор *А. В. Чугунова*.

Сдано в набор 27.12.2022 г. Подписано в печать 27.01.2023 г. Формат 60×88 1/8. Заказ PI223
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 **А. Д. Борисова, С. Д. Махортова**
«НЕЖЕСТКАЯ РЕГИСТРАЦИЯ ЧЕЛОВЕЧЕСКОГО ЛИЦА
ПО ИЗОБРАЖЕНИЯМ СО СТЕРЕОКАМЕРЫ»

Рис. 1. Пример ключевых геометрий с текстурами

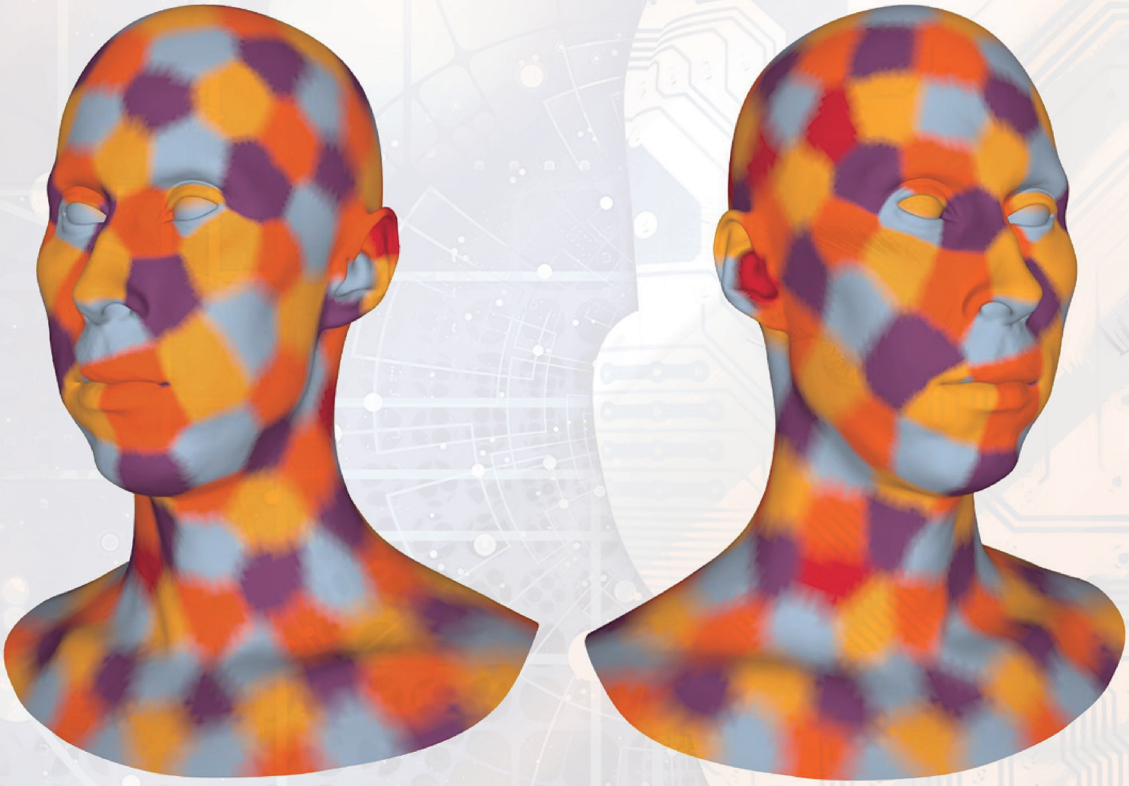

Рис. 2. Пример разбиения геометрии на патчи

Рисунок к статье
А. Д. Борисова, С. Д. Махортова
«НЕЖЕСТКАЯ РЕГИСТРАЦИЯ ЧЕЛОВЕЧЕСКОГО ЛИЦА ПО ИЗОБРАЖЕНИЯМ СО СТЕРЕОКАМЕРЫ»

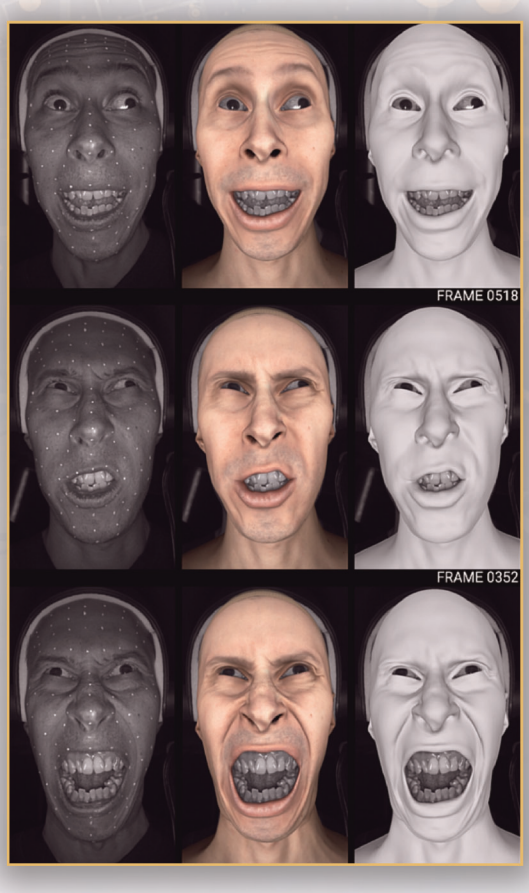

Рис. 7. Результат нежесткой регистрации

Рисунок к статье **Д. И. Читалова**
«О РАЗРАБОТКЕ МОДУЛЯ ДЛЯ РЕШАТЕЛЯ coupledPoroFoam ПАКЕТА OpenFOAM»

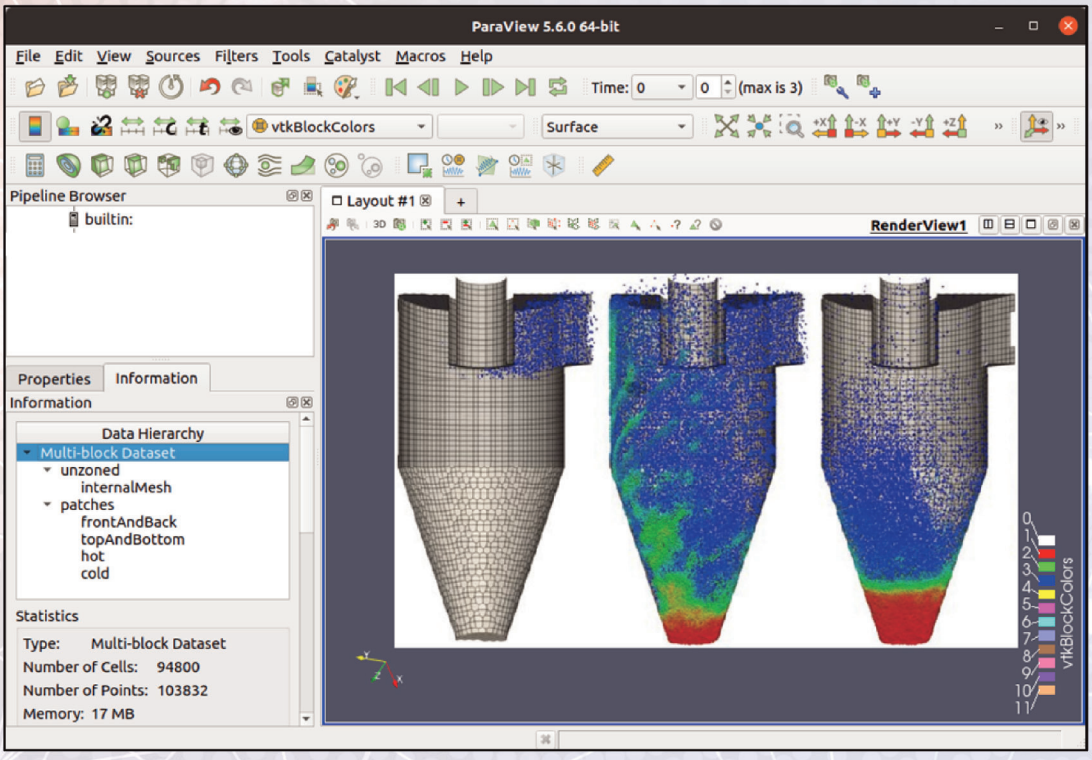

Рис. 3. Результаты численного моделирования задачи МСС на базе решателя coupledPoroFoam пакета OpenFOAM

ISSN 2220-3397. Программная инженерия. 2023. Том 14. № 2. 53–100. Подписной индекс по Объединенному каталогу «Пресса России» – 22765