ISSN 2220-3397

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

ПрИн
1
2022
Том 13

Рисунки к статье **Е. В. Авдеева**

«О НЕКОТОРЫХ АСПЕКТАХ УТОЧНЕНИЯ ТЕМПЕРАТУРНОГО РАСПРЕДЕЛЕНИЯ В РАЗРАБОТАННОМ ПРОГРАММНОМ ИНСТРУМЕНТАРИИ МОДЕЛИРОВАНИЯ НАГРЕВА ПОРОШКА ЛАЗЕРОМ»

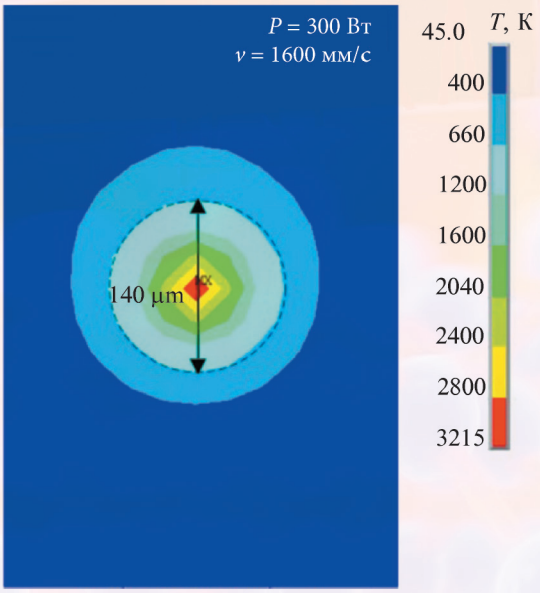

Рис. 8. Распределение температуры [14], диаметр ванны расплава 140 мкм

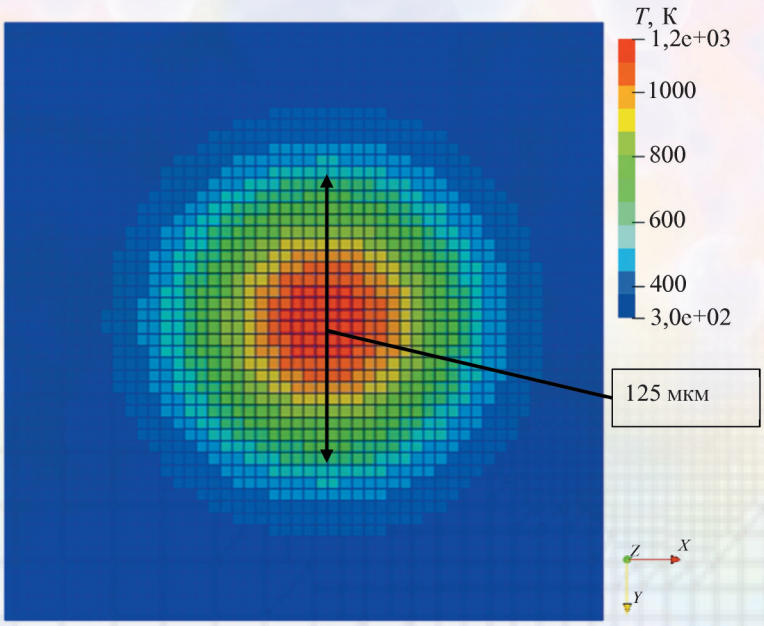

Рис. 9. Распределение температуры, диаметр ванны расплава 125 мкм, расчетный случай 1 (исходная грубая сетка)

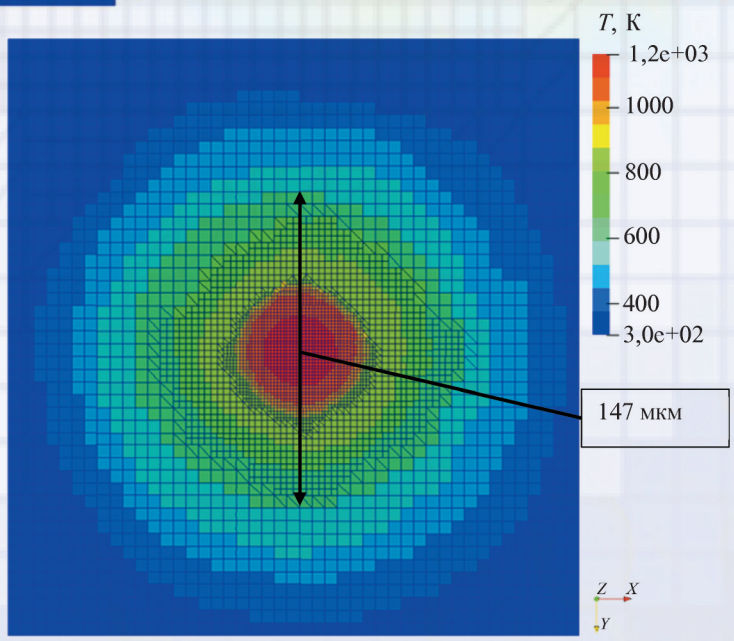

Рис. 10. Распределение температуры, диаметр ванны расплава 147 мкм, расчетный случай 2 (адаптированная сетка)

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

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.
Тел.: (499) 270-16-52.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования и базу данных RSCI на платформе Web of Science.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 13
N 1
2022

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 004.2+004.4 DOI: 10.17587/prin.13.3-16

В. В. Корнеев, д-р техн. наук, проф. гл. науч. сотр., korv@rdi-kvant.ru,
ФГУП "Научно-исследовательский институт "Квант", Москва

Параллельное программирование

Рассмотрен генезис моделей параллельного программирования. Показано, что параллелизм и аппаратная поддержка синхронизации, присущие архитектуре, обусловливают модель параллельного программирования. Современная элементная база требует перехода на модель программирования на базе разделяемой памяти.

Ключевые слова: *СБИС, архитектура, модель параллельного программирования, потоковая модель вычислений, граф межмашинных связей*

Введение

Несмотря на то что параллельные вычисления и модели параллельного программирования в настоящее время используются во всех сферах деятельности человека, принятие той или иной модели не осознается как технологический сдвиг, обусловливающий изменение архитектуры вычислительных систем, и, как следствие, модель параллельного программирования. Архитектура систем определяет возможности параллельного функционирования вычислительных ресурсов, а модель программирования — возможность эффективного отображения программы на аппаратные ресурсы.

В настоящей статье предпринята попытка рассмотреть генезис моделей параллельного программирования и обосновать необходимость перехода к модели с разделяемой памятью и синхронизацией на базе дополнительного бита слов памяти.

Однородные универсальные вычислительные системы высокой производительности

Параллельная обработка информации, как средство получения производительности, недостижимой для одиночного компьютера, возникла при решении актуальных вычислительно трудных задач на создаваемых для этого специализированных вычислительных системах. В начале 1960-х гг. в Институте математики имени С. Л. Соболева СО РАН (ИМ СОАН СССР) были начаты исследования по архитектуре вычислительных систем и вычислительных сред, параллельным алгоритмам и средствам параллельного программирования, машинной графике и распознаванию образов, что сейчас именуется искусственным интеллектом. Начальный этап исследований проблем построения и применения высокопроизводительных вычислительных систем завершился выходом в 1966 г. монографии [1]. Рассмотрим результаты, имеющие фундаментальное значение для появления и развития параллельного программирования, сведенные в монографии в единую концепцию на основе модели "коллектива вычислителей".

Впервые на основе анализа физических ограничений развития микроэлектроники были сформулированы подходы к повышению производительности вычислительных систем, включающие:

- увеличение тактовой частоты;
- рост числа одновременно (параллельно) функционирующих обрабатывающих устройств;
- программируемость структуры, как программную настройку связей между обрабатывающими устройствами.

Программируемость структуры при этом рассматривалась как средство достижения универсальности вычислительной установки, позволяющее получить высокую производительность при исполнении разных алгоритмов за счет программной реализации для исполнения этих алгоритмов специализированных вычислительных систем.

Чтобы вычислитель функционировал на тактовой частоте ν наибольшая длина связей (проводников) между устройствами не должна превышать $d_{\text{пр}} = c/2\xi\nu$, где c — скорость света; ν — тактовая частота единой сетки тактовых сигналов вычислителя; c/ν — длина волны, соответствующая тактовой частоте ν; ξ — коэффициент превышения длины волны над длиной связи (константа в пределах от 10 до 100). Пусть ρ — максимально допустимое число устройств в единице объема, определяемое технологией производства оборудования и возможностями теплоэнергообмена. Тогда число устройств, которое можно разместить в шаре диаметром $2d_{\text{пр}}$, не будет превышать $n_{\text{пр}} = 2\pi\rho c^3/3\xi^3\nu^3$. Производительность вычислителя прямо пропорциональна произведению числа устройств на тактовую частоту их функционирования. Поэтому предельная производительность вычислительных устройств с единой сеткой тактовых сигналов имеет принципиальное ограничение.

Следствие приведенной выше модели производительности заключается в том, что для достижения сколь угодно большой производительности нужно отказаться от единой тактовой сети и строить систему как совокупность синхронных вычислителей, объединенных асинхронными каналами. В этом случае ограничение на предельное число устройств действует только в пределах каждого отдельного вы-

числителя, а число таких вычислителей в системе не ограничено. Это обстоятельство позволяется достигать требуемой производительности, что и произошло в XXI веке. В настоящее время при построении систем их относят к архитектурам класса GALS (*Globally Asynchronous, Locally Synchronous*), в которых реализована глобальная асинхронность при передаче между синхронными вычислительными устройствами.

Доказанная возможность построения вычислительных систем с требуемым большим числом вычислительных машин, в терминологии авторов [1] — элементарных машин (ЭМ), еще не означала возможности решения задач с производительностью, пропорциональной числу использованных ЭМ. Для обоснования ограниченности роста производительности за счет параллельных вычислений использовались рассуждения, известные впоследствии как закон Амдаля: если в программе есть не распараллеливаемая часть σ и идеально распараллеливаемая часть $(1 - \sigma)$, то при исполнении этой программы на m процессорах достижимое за счет параллелизма ускорение $S = 1/(\sigma + (1 - \sigma)/m)$. В предельном случае при m, стремящемся к бесконечности, ускорение $S = 1/\sigma$.

Таким образом, с одной стороны, отсутствуют алгоритмизированные подходы к построению параллельных вычислительных систем и к разработке параллельных программ с доказанной эффективностью их исполнения. С другой стороны, есть прецеденты построения параллельных систем на базе параллельных алгоритмов решения вычислительно трудных задач. В этой ситуации было решено создать экспериментальную параллельную вычислительную систему "Минск-222" и начать исследовать на ее основе параллельные программы для актуальных вычислительных задач и наиболее часто встречающихся их фрагментов, реализующих вычислительные методы, например, решение систем линейных уравнений. В ходе этой работы предполагалось создать эффективные алгоритмы распараллеливания программ, в том числе — автоматизированного и автоматического.

Вычислительная система "Минск-222" была построена на базе серийной ЭВМ "Минск-22" путем подсоединения к каждой ЭВМ системного устройства (СУ), имеющего по два управляющих и информационных канала для соединения с соответствующими каналами двух соседних ЭВМ. В систему "Минск-222" могло быть объединено до 16 ЭМ, каждая имела номер i, $0 \leqslant i \leqslant N - 1$, N — число ЭМ в системе. Элементарная машина с номером i соединялась двунаправленными управляющим и информационным каналами с ЭМ с номерами k и j, $k = (i - 1) \bmod N$ и $j = (i + 1) \bmod N$. Таким образом, формировалась структура межмашинных связей с топологией "кольцо".

Каждое СУ имело в своем составе регистр настройки (РН), состоящий из триггеров: TR, TQ, TΩ, а также блок операций системы (БОС), расширяющий систему команд базовой ЭВМ так называемыми системными командами. Последние разделялись на четыре типа:

- команды настройки, управляющие установкой регистров настройки и регистров используемых признаков ЭМ;
- команды обобщенного безусловного перехода (ОБП), служащие для начальной загрузки программ и данных в ЭМ с установленным в "1" триггером TQ и принудительного управления ходом вычислений в этих ЭВМ (состояние "1" или "0" триггера TQ в ЭМ определяет соответственно выполнение или пропуск команды, выдаваемой из ЭМ, выполняющей команду ОБП);
- команды обобщенного условного перехода (ОУП), выполняющей переход по указанному в ней адресу, при условии, что

$$\Omega_k = \mathop{\&}_{i \in E} \omega_{ki}, \quad k = 1, 2, 3,$$

где E — подмножество номеров машин с предварительно установленными в "1" триггерами TΩ и выбранным признаком ω_k, $k \in \{1, 2, 3\}$, использованным для выработки обобщенного признака Ω_k, ω_{ki} — признак (результат меньше нуля, переполнение, результат равен нулю), вырабатываемый ЭМ с номером i;
- команды передачи (П) z слов памяти, начиная с заданного в команде адреса, и команды приема (ПР) w слов памяти с размещением их в собственной памяти, начиная с заданного в команде адреса, с выполнением их приема вплоть до поступления всех w слов и только после этого перехода к выполнению команды, следующей за командой приема.

Установка в "1" триггеров TR в ЭМ с номерами i и $i + n(\bmod N)$, а также установка в "0" триггеров TR в ЭМ с номерами $i + 1(\bmod N)$, ..., $i + n - 1(\bmod N)$, приводила к образованию подсистемы из n ЭМ с номерами i, $i + 1(\bmod N)$, ..., $i + n - 1(\bmod N)$. Таким образом, система могла быть разделена на совокупность подсистем, функционирующих как самостоятельные системы.

Архитектура системы "Минск-222" определила модель параллельного программирования на базе передачи сообщений и распределенной памяти. Усилиями специалистов по алгоритмам и программистов ИМ СОАН СССР были разработаны параллельные программы, настраиваемые на предоставленное число ЭМ, как на параметр. Такие программы удалось создавать для практически важных вычислительных задач и получать при их исполнении ускорение, прямо пропорциональное числу используемых ЭВМ с некоторым повышающим коэффициентом. Большая емкость оперативной памяти в системе "Минск-222" по сравнению с одной ЭВМ "Минск-22" с медленной внешней памятью на магнитной ленте и быстродействие каналов связи, сравнимое с быстродействием оперативной памяти, обеспечивали производительность, значительно превосходящую "механическую" сумму производительностей машин системы.

Сформировалось представление параллельной программы как совокупности ветвей с операторами взаимодействия между ними по данным и управлению. Фундаментальным результатом, полученным на основе опыта создания практически востребованных параллельных программ, было выявление типовых схем обмена данными между ветвями параллельной программы. Эти схемы сводятся к пяти типам обмена: трансляционный; трансляционно-циклический;

конвейерно-параллельный; коллекторный; дифференцированный. При трансляционном обмене один и тот же блок данных передается из одной (любой) ветви программы одновременно во все остальные ее ветви. Трансляционно-циклический обмен транслирует блок данных из каждой ветви во все остальные. Конвейерно-параллельный обмен обеспечивает передачу блока данных из каждой ветви i, выполняемой на ЭМ с номером i, в соседнюю ветвь k, $k = (i + 1) \bmod B$, где B — число ветвей параллельной программы. Коллекторный обмен реализует сбор блоков данных из других ветвей программы в одну ветвь. Наконец, дифференцированный обмен выполняет передачу блоков данных между задаваемыми парами ветвей или из одной ветви в несколько.

В середине 1990-х гг. модель программирования на базе передачи сообщений стала общепризнанной и оформилась как *Message Passing Interface* (MPI) — библиотека функций для создания и исполнения параллельных программ. При этом имеет место следующее соответствие между функциями системы параллельного программирования "Минск-222" и MPI:

- трансляционный обмен — MPI_Bcast;
- трансляционно-циклический обмен — MPI_Alltoall или MPI_Allscatter;
- коллекторный обмен — MPI_Gather;
- обобщенный условный переход — MPI_Barrier;
- дифференцированный обмен — MPI_Send, MPI_Recv.

Таким образом, можно констатировать, что в монографии [1] была предложена перспективная, со временем ставшая основополагающей, концепция архитектуры параллельных вычислительных систем с распределенной памятью и модель параллельного программирования на базе обмена сообщениями.

Начиная с 1970-х гг. предпринимались попытки создавать параллельные системы, базирующиеся на этой концепции. Так был проект объединения машин старших моделей ЕС ЭВМ, но в одной, даже крупной организации, была только одна ЭВМ, а объединение ЭВМ в пределах страны представляло собой не только техническую проблему. Кроме того, используемая элементная база позволяла создавать более производительные ЭВМ с единой сетью синхронизации с повышением тактовой частоты, так как число элементов в них было далеко до предела, определяемого плотностью упаковки элементов и тактовой частотой.

Положение изменилось с появлением микропроцессоров. Была создана вычислительная система МВС-1000М [2] с производительностью на уровне 10^{12} флопс. Это была первая система отечественной разработки, попавшая в верхние 50 позиций списка TOP500 самых производительных вычислительных установок в мире. Архитектура системы МВС-1000М предоставляет только системные операции передачи и приема блока данных. Как интерфейс прикладного программирования пользователям предоставляются вызовы программ библиотеки MPI. В определенной мере это является следствием отсутствия возможности добавить системные команды, как это сделано в "Минск-222", в микропроцессор. Однако программная реализация барьерной синхронизации в MPI, в отличие от системной команды ОУП "Минск-222", требует значительных временных затрат и служит одним из основных источников потери производительности при параллельных вычислениях. Поэтому, например, в IBM Blue Gene\L введена специальная коммуникационная сеть для реализации аналога ОУП "Минск-222".

Вычислительные системы с программируемой структурой

В начале 1970-х гг. появились теоретические вопросы по коренному противоречию между реализацией архитектуры "Минск-222" и моделью параллельных систем из монографии [1]. Вопросы заключались в том, что реализация предполагала "длинные" связи, например, при выполнении системной команды ОУП, что при увеличении числа ЭВМ системы должно было вести к снижению тактовой частоты. Кроме того, команды П и ПР предполагали наличие связей между любыми ЭМ системы, сколь большой бы она не была, т. е. связи должны быть "длинными". А архитектуры класса GALS, локально синхронные и глобально асинхронные, не должны иметь "длинных" связей. Каждая ЭМ должна иметь связи только с ограниченным числом соседних ЭМ, а для пересылки данных в не соседнюю ЭМ должна использоваться последовательность пересылок между соседними ЭМ. Поэтому требовалось понять, как строить вычислительные системы, имеющие только "короткие" локальные связи между ЭМ и как программировать и эффективно исполнять параллельные программы с производительностью, прямо пропорциональной числу используемых ЭМ. Результаты исследований по этим и смежным вопросам опубликованы в вышедшей в 1985 г. монографии [3]. Ниже будут представлены основные положения предложенной архитектуры и подхода к параллельному программированию с локальными связями.

Потенциальная неограниченность числа ЭМ, объединяемых в систему, может быть достигнута только при использовании пространственного распределенного коммутатора межмашинных связей с децентрализованным устройством управления. Во всяком другом случае структуру системы можно представить, как совокупность машин, связанных проводниками с одним коммутатором, что противоречит потенциальной неограниченности числа машин при неизменной тактовой частоте.

Пространственный коммутатор $N \times N$ с децентрализованным управлением с N входами и N выходами строится из N неординарных коммутаторов с v + 1 входом $\mathbf{вх}_i$ и выходом $\mathbf{вых}_i$, $i = 0, 1, ..., v$. Пример построения такого коммутатора с восемью входами и выходами, созданного на базе коммутаторов с v = 3, показан на рис. 1. При этом вход $\mathbf{вх}_0$ и выход $\mathbf{вых}_0$ каждого из N неординарных коммутаторов j, $j = 1, ..., N$, служат $Вх_j$ и $Вых_j$ пространственного коммутатора.

При создании вычислительной системы из N машин и имеющихся коммутаторов с v + 1 входом $\mathbf{вх}_i$

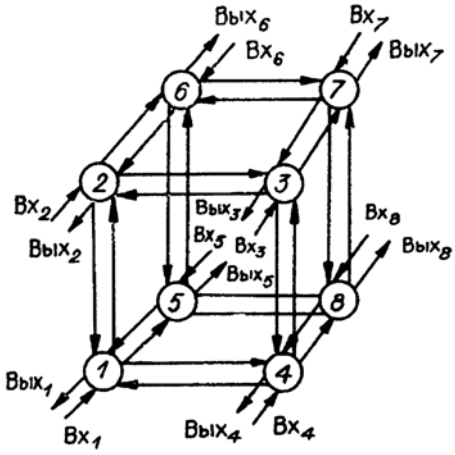

Рис. 1. Пространственный коммутатор 8×8

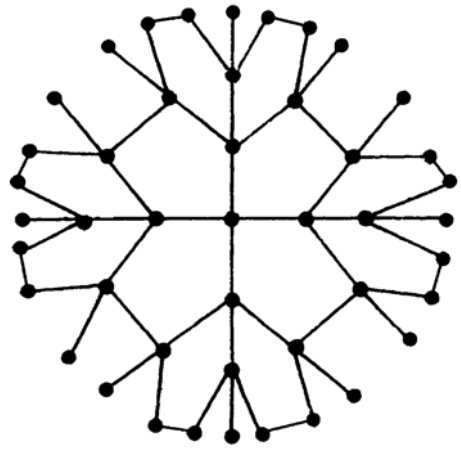

Рис. 2. Первые три яруса бесконечного планарного графа *L*(4, 5)

и выходом **вых**$_i$, i = 0, 1, ..., v, необходимо выбрать граф межмашинных связей. Поскольку существуют различные неизоморфные однородные графы с числом вершин N и степенью вершин v, возникает проблема выбора из них графа, оптимального по критериям, предъявляемым к графам межмашинных связей. Была выдвинута гипотеза, подтвержденная статистическими экспериментами, что оптимальный граф обладает минимальными диаметром и средним диаметром среди графов с числом вершин N и степенью вершин v.

В качестве графов межмашинных связей предложено параметрическое семейство однородных графов Л(N, v, g) с количеством вершин N, степенями вершин v, обхватом g, каждая вершина которых входит в v циклов длины g.

Граф Л(N, v, g) строится на основе N-вершинного подграфа бесконечного планарного графа L(v, g) путем выбора соответствующего подграфа из всех существующих и формирования ребер с образованием графа Л(N, v, g). При этом N-вершинный подграф включает:

- все вершины d ярусов графа L(v, g), если $\sum_{i=0}^{d} Ni = N$, где Ni — число вершин на ярусе i, i = 0, ..., d, N_0 = 1, N_1 = v, d — диаметр строящегося графа Л(N, v, g);
- все вершины d − 1 ярусов графа L(v, g), если $\sum_{i=0}^{d-1} Ni < N$ и $N - \sum_{i=0}^{d-1} Ni$ вершин яруса d.

Алгоритм перебирает возможные варианты и либо находит требуемый граф, либо определяет отсутствие возможности его построения. На рис. 2 приведены три яруса бесконечного планарного графа L(4, 5).

Графы Л(N, v, g) существуют не для всех комбинаций значений параметров N, v и g. С помощью переборного алгоритма удалось, например, построить графы Л(N, 4, 5) для значений N = 19, ..., 45.

На рис. 3 приведены графы Л(10, 3, 5) и Л(24, 4, 5).

Граф Л(10, 3, 5) имеет диаметр 2 и включает все вершины ярусов 0, 1, 2 графа L(3, 5), Граф Л(24, 4, 5) имеет диаметр 3 и включает все вершины ярусов 0, 1, 2 графа L(4, 5), а также 7 вершин яруса 3 этого графа.

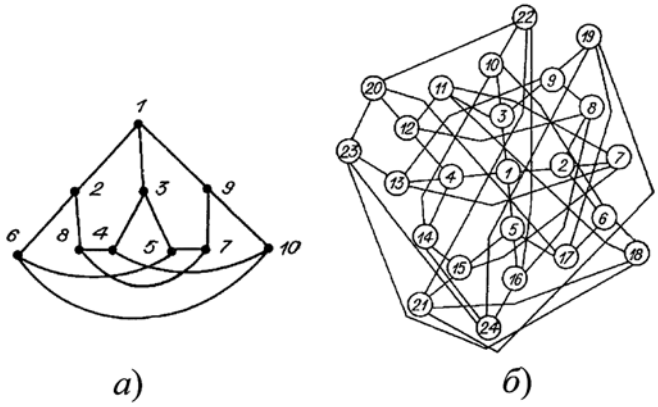

Рис. 3. Графы Л(10, 3, 5) (*а*) и Л(24, 4, 5) (*б*)

Графы Л(N, 4, 4) являются групп-графами конечных абелевых групп порядка N с двумя образующими элементами. Они существуют для всех значений N.

У графа Л(N, v, g) будут минимальные диаметр d и средний диаметр $d_{ср}$, если $g = 2d + 1$ или $g = 2d$ среди всех графов с числом вершин N, степенями вершин v.

Возможной постановкой задачи поиска оптимального графа межмашинных связей служит ограничение диаметра (числа хопов — промежуточных передач пакетов). В этом случае для построения используются коммутаторы с v ⩾ 64. В работе [4] представлены алгоритмы синтеза графов, близких к Л(N, v, 5) и Л(N, v, 7), с диаметрами 2 и 3, соответственно.

В работе [5] представлены результаты статистического моделирования, доказывающие оптимальность использования Л(N, v, g) с минимальными диаметром d и средним диаметром $d_{ср}$, включая эффективность реализации функций библиотеки MPI на вычислительных системах с графами межмашинных связей Dragonfly и Л(N, v, 7) с одинаковыми числом вершин и степенями вершин. С помощью собственного переборного алгоритма синтезированы графы Л(20, 4, 7), Л(30, 5, 7), Л(36, 5, 7), Л(252, 11, 7), Л(264, 11, 7) с диаметром 3.

Важными прикладными аспектами использования Л(N, v, g) с минимальными диаметром d и средним диаметром $d_{ср}$ в качестве графа межмашинных связей служат:

- максимизация вероятности образования связного подграфа из исправных машин при заданных вероятностях отказа машин и линий связи среди всех графов с числом вершин N и степенями вершин v (структурная живучесть);
- максимизация вероятности числа одновременно устанавливаемых соединений между отдельными машинами и/или группами машин (структурная коммутируемость).

Архитектура с локальными связями представляет возможность непосредственной передачи блока данных только между соседними машинами, соединенными линией связи. Структура ЭМ вычислительной системы с графом Л(N, 4, 4) показана на рис. 4.

В передающей машине блок данных помещается в выходную очередь линии связи. Если во входной очереди этой линии связи имеется место для приема блока данных, то он поступает во входную очередь. Если во входной очереди нет свободного места, то передача задерживается вплоть до момента освобождения места во входной очереди.

Передача блока данных между машинами, не соединенными линией связи, реализация ОУП, а также ОБП выполняется через машины, служащие соседними друг другу. Выделение подсистем (программирование структуры системы) выполняется программно средствами системного программного обеспечивания. С точки зрения программиста, структура, для которой создается параллельная программа, выбирается из числа представленных на рис. 5 или какая-либо другая. Важно чтобы эта структура имела параметрическое описание и могла быть выделена с любым требуемым числом машин. Следует отметить, что "корневое дерево" может быть создано на связном подграфе при любом Л(N, v, g) графе межмашинных связей. Структуры "линейка" и "кольцо" при этом имеют ограничения на возможности их построения, а "решетка" строится без использования транзитных машин только на Л(N, v, 4) графе.

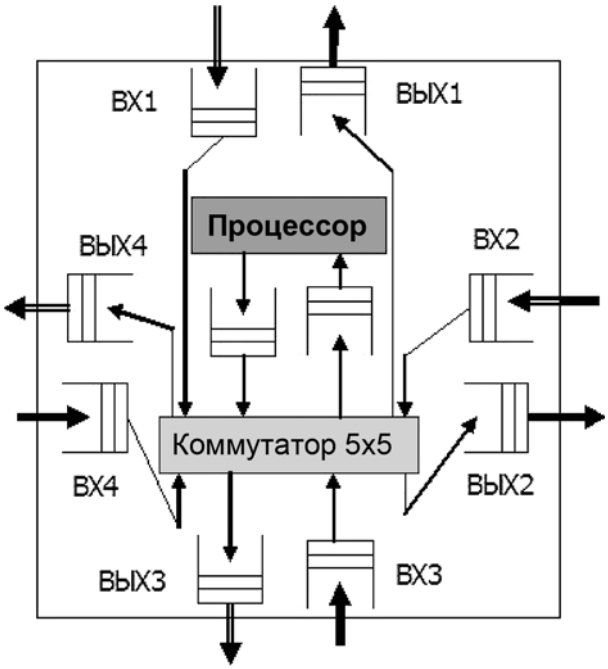

Рис. 4. Структура ЭМ вычислительной системы с графом Л(N, 4, 4)

Рис. 5. Структуры подсистем:
а — "линейка"; *б* — "кольцо"; *в* — "корневое дерево"; *г* — "решетка"

Для возможности программирования необходимо определить нумерацию машин. Будем полагать, что при построении подсистемы выбирается корневая машина, которой присваивается номер 0. Нумерация машин для подсистем "линейка" и "кольцо" естественна, для подсистемы "решетка" могут быть варианты по строкам или столбцам. Примем, что нумерация машин подсистемы "корневое дерево" выполняется по алгоритму "поиск в глубину" с учетом одного и того же порядка номеров полюсов для всех коммутаторов, образующих пространственный коммутатор системы. Примеры задания такой нумерации приведены на рис. 6.

Параллельная программа должна задавать поток данных между локально взаимодействующими машинами, которые исполняют локально свои параллельные ветви. Каждый обмен данными между ветвями параллельной программы выполняется по одной из ранее рассмотренных схем: трансляционной, трансляционно-циклической, конвейерно-параллельной, коллекторной или дифференцированной. По этой причине параллельная программа должна состоять из двух взаимодействующих программ — вычислительной и путевой процедур.

Рассмотрим суть этих процедур. Пусть требуется выполнить трансляционно-циклическую схему обмена. На рис. 7 показана подсистема из шести машин со структурой "корневое дерево", а также сформированные для реализации этой схемы обмена "восходящая" и "нисходящая" сети подсистемы. Будем обозначать τ_i вес вершины i, равный числу вершин поддерева, корнем которого она служит, для подсистемы рис. 7, $\tau_0 = 6$, $\tau_1 = 1$, $\tau_2 = 4$.

Путевая процедура, реализующая трансляционно-циклический обмен, задает функционирование восходящей и нисходящей сетей подсистемы, на $n + 1$ машинах которой исполняется параллельная программа.

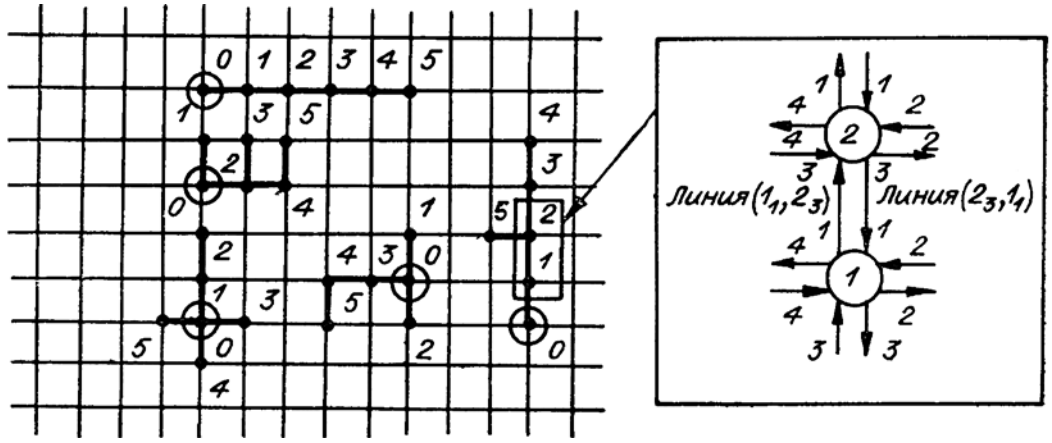

Рис. 6. Подсистемы из шести машин с заданными номерами (обведены корни деревьев)

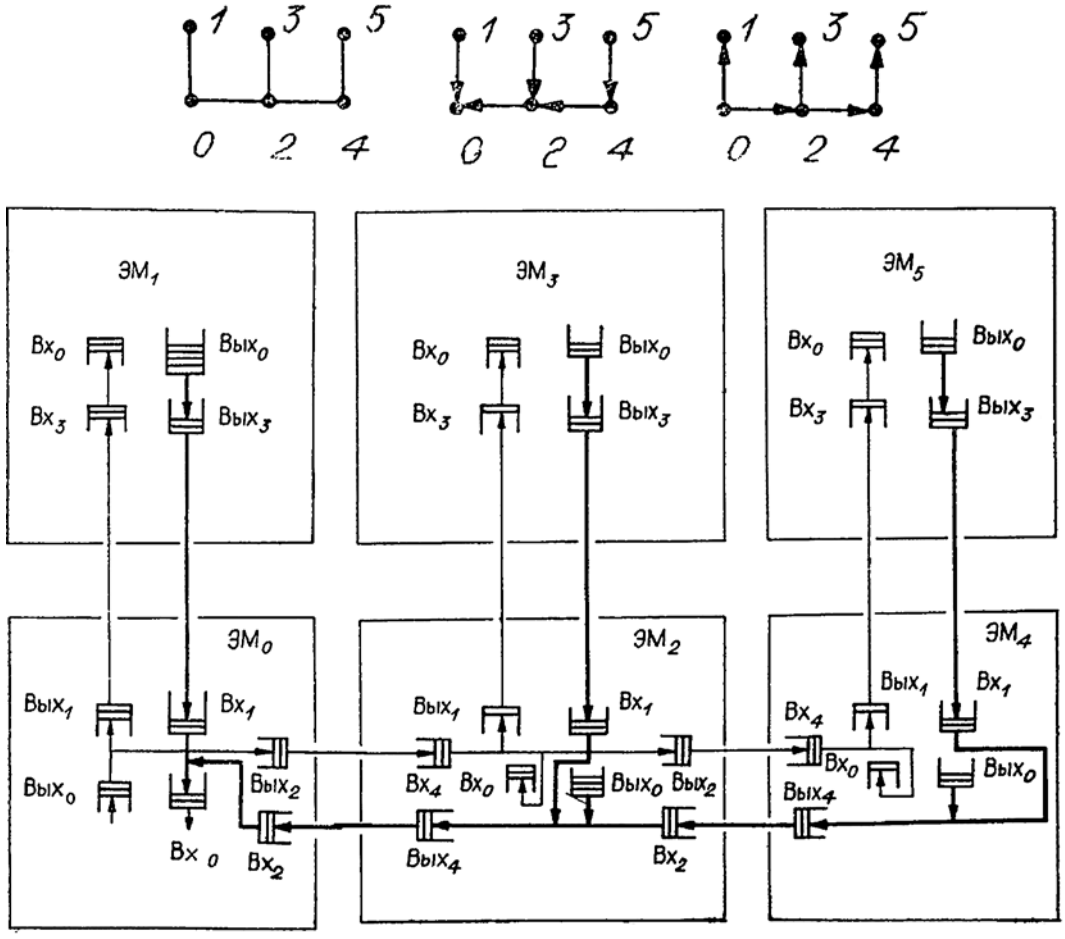

Рис. 7. Подсистема из шести машин со структурой "корневое дерево"

В восходящей сети каждая машина $ЭМ_i$, $i = 1, ..., n$, загружает по следующей циклической дисциплине очередь $Вых_{e(j)}$ линии (i, j), передающую элемент данных в $ЭМ_j$, служащую для нее машиной-преемником в восходящей сети, $e(j)$ — номер полюса коммутатора, $e(j) \in \{1, ..., v\}$ линии (i, j). Первым передается один элемент данных, выбираемый из начала очереди $Вых_0$, вырабатываемый в самой $ЭМ_i$, $i = 1, ..., n$. Следом за ним в $Вых_{e(j)}$ заносятся τ_{i1} элементов данных, каждый из которых выбирается из начала очереди $Вх_{i1}$, содержащей элементы данных, поступающие в $ЭМ_i$ из $ЭМ_{i1}$. $ЭМ_{i1}$ служит машиной-предшественником $ЭМ_i$ и имеет наименьший номер $i1$ среди всех машин $ЭМ_{i1}$, ..., $ЭМ_{ip}$, являющихся

машинами-предшественниками $ЭМ_i$ в восходящей сети $i_1 < ... < i_p$. После τ_{i1} элементов данных из очереди $Вх_{i1}$ в $Вых_{e(j)}$ заносятся τ_{i2} элементов данных из очереди $Вх_{i2}$ и далее, пока не будет занесено τ_{ip} из очереди $Вх_{ip}$. Затем последовательность действий повторяется, начиная с передачи одного элемента данных, выбираемого из начала очереди $Вых_0$.

Корневая машина $ЭМ_0$ загружает очередь $Вх_0$, циклически выбирая по описанной выше схеме элементы данных из очередей $Вх_{01}$, $Вх_{02}$, ..., $Вх_{0s}$, поступающие в $ЭМ_0$ из машин-предшественников $ЭМ_{01}$, ..., $ЭМ_{0s}$.

Функционирование нисходящей сети инициируется $ЭМ_0$. Из начала очереди $Вых_0$ выбирается элемент данных, выработанный вычислительной процедурой в $ЭМ_0$, и загружается в конец очередей $Вых_{01}$, $Вых_{02}$, ..., $Вых_{0s}$ для передачи по нисходящей сети из $ЭМ_0$ в машины-преемники $ЭМ_{01}$, ..., $ЭМ_{0s}$. Каждая $ЭМ_i$ в нисходящей сети заносит выбранный из начала очереди $Вх_{e(j)}$ элемент данных, поступивший из $ЭМ_j$ машины-предшественника, в свою очередь $Вх_0$ и выходные очереди, пересылающие элемент данных в машины-преемники.

Вычислительная процедура, исполняемая каждой машиной, берет элементы данных из очереди $Вх_0$ и после обработки помещает результат в очередь $Вых_0$. Если какая-либо ЭМ не завершила вычислительную процедуру к моменту, когда она может загружать элемент данных, то передача задерживается, равно как если в какой-либо очереди нет места для приема элемента данных.

Совместное функционирование восходящей и нисходящей сетей приводит к загрузке в очередь $Вх_0$ в каждой $n + 1$ машине подсистемы последовательности $x_1^0, x_2^0, ..., x_n^0, x_1^1, ..., x_n^1, x_1^2, ...$.

Используя приведенную выше реализацию трансляционно-циклического обмена, можно, слегка ее модифицировав и создав соответствующую вычислительную процедуру, строить параллельные программы, применяющие другие схемы обменов. Реализация дифференцированных обменов возможна, например, с использованием подобно трансляционно-циклической схемы модифицированной восходящей и нисходящей сетей. Модификация восходящей сети состоит в том, что циклически опрашиваются все очереди, поставляющие элементы данных, пропуская пустые. Сами элементы данных снабжаются тэгом, состоящим из номера ЭМ, их выработавшей, и номеров ЭМ, назначенных получателями.

Другая реализация межмашинных обменов основана на адресации машин подсистемы, исполняющей ветви параллельной программы.

Для реализации дифференцированных обменов может использоваться задание адресов машин и передача элементов данных по кратчайшим путям адресатам. Известна координатная адресация, которая применяется в вычислительных системах с Л(N, v, 4) графом или многомерными кубическими структурами в качестве графов межмашинных связей, в которых адрес ЭМ задается координатами по каждому направлению. При передаче элемента данных сравниваются значения координат адреса назначения и ЭМ, в которой выполняется сравнение. Если значения всех координат равны, то адресат достигнут. Среди направлений передачи с не совпавшими координатами выбирается ведущее к уменьшению рассогласования. Это эффективная адресация. Она используется в ряде суперкомпьютеров, например, фирмы CRAY.

Для вычислительных систем с Л(N, v, g) — графом межмашинных связей предложена Д(z, m)-адресация, при которой адрес $ЭМ_i$, $i \in \{0, ..., n-1\}$, также задается вектором $\mathbf{A}_i = A_{i0}, ..., A_{im-1}$. Каждый A_{ij}, $j = 0, ..., n-1$, принимает значение из множества $\{0, 1, ..., 2^z - 1\}$, $z \in \{1, 2, ...\}$, n — количество машин подсистемы, $mz \geqslant -\lceil \log_2 n \rceil$, $\lceil x \rceil$ — наименьшее целое такое, что $x \geqslant \lceil x \rceil$. $ЭМ_i$ и $ЭМ_k$, $i, k \in \{0, 1, ..., n-1\}$, принадлежат подсистеме $R_r(A_{i0}, ..., A_{ir-1})$ яруса r, если $A_{ir} \neq A_{kr}$, и $A_{ij} = A_{kj}$ для всех $j < r$, $r = 1, 2, ..., m$; $R_0()$ — подсистема яруса 0 (вся подсистема из n машин), $R_m(A_{i0}, ..., A_{im-1})$ — подсистема яруса m, состоящая из одной $ЭМ_i$. Адресация машин должна удовлетворять следующим свойствам:

- машины каждой подсистемы $R_r(A_{i0}, ..., A_{ir-1})$ яруса r, $r = 1, 2, ..., m$, должны вместе с линиями связи между ними отображаться в связный подграф графа межмашинных связей;
- $R_0 \supseteq R_1(A_{i0}) \supseteq R_2(A_{i0}, A_{i1}) \supseteq ... \supseteq R_m(A_{i0}, ..., A_{im-1})$;
- $R_r(A_{i0}, ..., A_{ir-1}) = \bigcup_{j=0}^{d} R_{r+1}(A_{i0}, ..., A_{ir-1}, j)$, $d = 2^z - 1$;
- $R_{r+1}(A_{i0}, ..., A_{ir-1}, j) \cap R_{r+1}(A_{i0}, ..., A_{ir-1}, k) = \varnothing$, $j \neq k$.

Подсистема из 15 машин с заданной Д(2, 3)-адресацией показана на рис. 8.

При Д(z, m)-адресации в каждой машине $ЭМ_i$, $i \in \{0, 1, ..., n-1\}$, должно храниться только $m2^z$ номеров $p_1(0)$, $p_1(1)$, ..., $p_1(2^z - 1)$, $p_2(0)$, $p_2(1)$, ..., $p_2(2^z - 1)$, ..., $p_m(0)$, $p_m(1)$, ..., $p_m(2^z - 1)$ выходных полюсов $ЭМ_i$, принадлежащих кратчайшим путям из $ЭМ_i$ соответственно в подсистемы $R_1(0)$, $R_1(1)$, ..., $R_1(2^z - 1)$, $R_2(A_{i0}, 0)$, ..., $R_2(A_{i0}, 2^z - 1)$, ..., $R_m(A_{i0}, ..., A_{im-2}, 2^z - 1)$. Кратчайший путь до подсистемы — это путь до ближайшей машины этой подсистемы. Минимальный

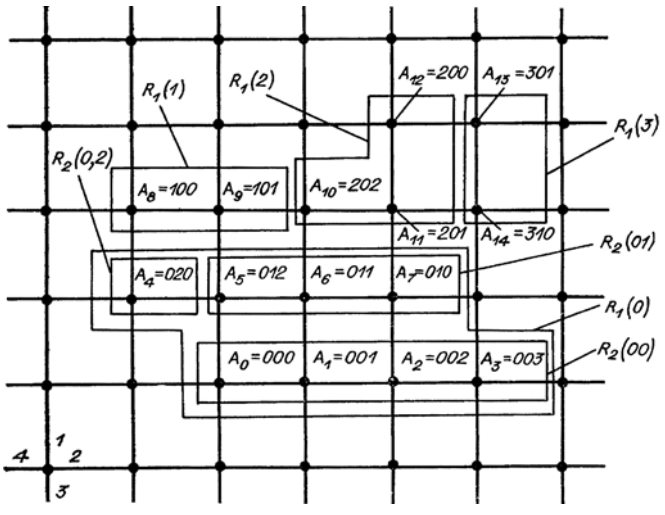

Рис. 8. Подсистема из 15 машин с Д(2, 3)-адресацией

объем требуемой памяти достигается при z = 1 и $m = \lceil \log_2 n \rceil$.

Одним из вариантов реализации Д(z, m) — адресации служит следующий. Полагаем m равным расстоянию от корневой до наиболее удаленной от нее машины подсистемы. Корневая машина $ЭМ_0$ получает адрес $A_0 = A_{00}$, ..., $A_{0m-1} = 0$, ..., 0. Остальные машины получают адреса, состоящие из последовательности номеров выходных полюсов ЭМ, принадлежащих пути между корневой и адресуемой машинами. Если длина этого пути r, $r = 1, 2, ..., m$, меньше m, то компоненты адреса, начиная с r-й, равны 0.

При Д(z, m)-адресации определение достижения адресата и выбор направления дальнейшей передачи выполняются по тому же алгоритму как при координатной адресации, за исключением того, что при выборе направления передачи учитывается рассогласование координаты с наименьшим номером. Передачи выполняются по близким к кратчайшим путям, так как хранятся направления к подсистемам, а не к требуемому адресату.

На базе Д(z, m)-адресации могут быть построены путевые процедуры для эффективной реализации трансляционной, трансляционно-циклической, конвейерно-параллельной, коллекторной схем обмена.

С использованием рассмотренных путевых и вычислительных процедур для широкого круга вычислительных методов были созданы параллельные программы с ускорением вычислений, пропорциональным числу используемых машин вычислительной системы с архитектурой с локальными связями. Этот результат достигается за счет программирования структуры, необходимой для реализации схем обмена исполняемого параллельного алгоритма. Поэтому вычислительные системы с архитектурой с локальными связями логично называть "вычислительные системы с программируемой структурой".

Для проверки всего комплекса идей по построению программно-аппаратных средств объединения машин в систему, децентрализованной распределенной операционной системы, средств программирования структуры и разработки параллельных программ был реализован макет вычислительной системы с программируемой структурой МИКРОС. Система построена на базе серийных микроЭВМ "Электроника-60". Общий вид системы показан на рис. 9.

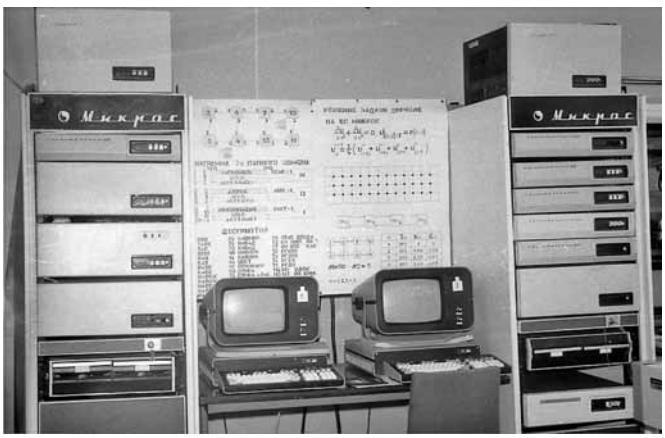

Рис. 9. Вычислительная система с программируемой структурой МИКРОС, 1986 г.

Экспериментальные исследования, проведенные на системе МИКРОС, продемонстрировали реализацию основных положений концепции вычислительных систем с программируемой структурой, что было подтверждено актом комиссии СО АН СССР, утвержденным председателем Президиума СО АН СССР.

К настоящему времени параллельные вычисления с локальными связями между машинами реализуются в основном в спецвычислителях с фиксированной структурой и известны как систолические и волновые. Программирование современных суперкомпьютеров, построенных на СБИС с небольшим (в пределах сотен) числом процессорных элементов, выполняется, как правило, с использованием MPI. Однако представляется, что при использовании СБИС с 10^3 и более процессорных элементов программируемость структуры станет практически востребованной как средство достижения универсальности параллельного программирования. Возможно, что это будет внутри MPI, а прикладной программист будет пользоваться привычным интерфейсом.

Потоковые вычислительные системы с разделяемой памятью

Исходя из современных представлений о развитии элементной базы и архитектур, суперкомпьютер экзафлопсного уровня производительности должен обрабатывать порядка 10^9 и более потоков при производительности одного потока порядка 10^9флопс. Однако практика использования современных компьютеров списка TOP500 показывает, что уже на компьютерах с миллионами потоков проявляются препятствия их эффективному применению и дальнейшему повышению быстродействия в силу простоев процессоров в ожидании завершения чтения/записи данных (так называемая "стена памяти").

В концептуальном аспекте, преодоление разрыва между временем выполнения команд в процессоре и временем доступа к памяти возможно при создании программ, использующих весь присущий алгоритму параллелизм до мельчайших гранул. В этом случае появляется возможность при ожидании одним потоком завершения обращения к памяти выполнять другие потоки, параллельные с ним [6], т. е. работать с памятью в темпе обслуживания потока запросов. Для снижения потерь производительности при переключении процессора с исполнения одного потока на выбор и начало исполнения другого предложены "легкие" потоки — q-треды, имеющие минимальный контекст. При таком подходе программа для суперкомпьютера экзафлопсного уровня производительности должна представлять собой описание порождения миллиардов потоков, межпотоковых коммуникаций и синхронизации этих потоков. Эту программу компилятор (статически) и библиотеки (динамически в ходе вычислений) преобразуют в совокупность потоков, определяя ресурсы, на которых будут протекать эти потоки или оставляя отображение потоков на ресурсы Runtime системы времени исполнения.

Могут использоваться отображения как в асинхронные потоки, протекающие в универсальных процессорах и имеющие собственный отдельный счетчик команд, так и в синхронные потоки, выполняемые по одному счетчику команд в разных арифметическо-логических устройствах (АЛУ), например, в GPGPU NVidia Fermi синхронно в каждом такте может протекать до 512 потоков. Синхронные потоки составляют аппаратно экономную альтернативу асинхронным, но более сложны в программировании [7]. Таким образом, программа будет исполняться на гетерогенных ресурсах. Необходимо для доступных заданию гетерогенных ресурсов обеспечить оптимальное назначение на них тредов задания. Конечной целью служит автоматическое назначение, но начинать следует с создания API библиотеки формирования требуемой совокупности ресурсов и назначения тредов на эти ресурсы. Надо уметь определять гетерогенные ресурсы, например, устройства выполнения пучков синхронных тредов, и выделять в программе треды, которые должны выполняться в синхронном режиме на этих устройствах. В ходе отображения тредов на ресурсы должны формироваться межтредовые коммуникации с использованием наиболее подходящего коммуникационного ресурса и режимов использования этих ресурсов (организация потоков сообщений, формирование сообщений заданной длины путем сборки из нескольких и т. д.).

Существующая модель программирования на базе передачи сообщений не позволяет эффективно использовать потенциальный параллелизм алгоритмов обработки. Создание других моделей экзафлопсного программирования необходимо чтобы:

— эффективно использовать многократно возросший параллелизм обработки (10^9 и более потоков);

— при обеспечении высокой степени параллелизма не вносить дополнительных трудностей при разработке параллельных программ, т. е. не снижать продуктивности их разработки.

Более того, новые модели должны многократно в сравнении с существующим уровнем повышать эту продуктивность. Этому должно способствовать применение глобально адресуемой памяти, а также повышение уровня и непроцедурности средств параллельного программирования, которые характерны для новых моделей.

В качестве основы модели экзафлопсного программирования, удовлетворяющей вышеуказанным требованиям, может быть выбрана модель, известная как *Parallel Random Access Model* (PRAM) [8]. Эта модель базируется на архитектуре мультипроцессора с общей разделяемой памятью. Процессоры P_i, $i = 1, ..., n$, имеют доступ к общей памяти (*distributed shared memory* — DSM) или секционированной общей памяти (*partitioned global address space* — PGAS), которая физически распределена по вычислительным модулям (процессор + блок памяти), однако имеет общее адресное пространство и логически доступна всем процессорам. Создавая программу, пользователь предполагает, что команды потоков, протекающие в каждом из процессоров, выполняются синхронно (время выполнения всех команд одно и то же), а межпроцессовые коммуникации и синхронизация осуществляются через ячейки разделяемой памяти.

Возможны различные подходы к разрешению конфликтов доступа разных процессоров к одной ячейке разделяемой памяти [8]. В контексте настоящего рассмотрения будем полагать приемлемым подход *concurrent-read exclusive-write* (CREW) с возможностью чтения одной и той же ячейки совокупностью процессоров, а записи только одним из процессоров. При этом все одновременные доступы к памяти по чтению предшествуют доступу по записи.

В этой модели параллельного программирования пользователь должен указывать, какие вычисления можно проводить параллельно, сообразуясь только с выбранным алгоритмом. Это способствует выявлению всего параллелизма, присущего алгоритму, и повышает продуктивность программирования [8].

Механизм разрешения конфликта доступа к одной ячейке разделяемой памяти CREW реализуется аппаратными средствами управления доступом к памяти. В свое время именно отсутствие эффективного масштабируемого аппаратного решения для разрешения конфликтов доступа к памяти привело к отказу от PRAM и переходу к модели на базе передачи сообщений. Однако представляется, что такое решение в настоящее время существует.

Рассмотрим по порядку отдельно языковые средства порождения и завершения процессов и средства межпроцессовых коммуникаций и синхронизации в рамках предлагаемого расширения PRAM-модели [8].

Языком параллельного программирования, реализующим модель PRAM, может служить расширение языка C [8]. Для порождения и завершения асинхронных тредов в него введены три дополнительные функции: spawn (*a*, *n*), join, fetch_and_add(*e*, *x*). Эти функции позволяют, соответственно:

- породить заданное параметром *n* число тредов с последовательными номерами, начиная с номера *a*;
- завершить тред, исполняющий join, и перейти к следующему оператору, если завершены все порожденные соответствующей spawn треды;
- выполнить как неделимую атомарную операцию присвоения переменной, например, номеру треда, значения параметра *x* и увеличить значение *x*, прибавив к *x* значение *e*.

Пример параллельного программирования задачи формирования массива *B*, состоящего из ненулевых элементов массива *A*, представлен ниже [8]. Символ $ используется для обозначения номера текущего процесса.

```
intx;
x = 0;
spawn(1, n) {
int e;
e = 1;
if (A[$]! = 0) {
a = fetch_and_add(e, x)
B[a] = A[$];
}
}
```

Исходя из необходимости реализации PRAM, ЭМ должна выглядеть как "обычный" многопотоковый процессор, функционирующий под управлением операционной системы (ОС) семейства Unix.

В ЭМ под управлением ОС Minix3 могут быть "нативно" реализованы алгоритмы распараллеливания программы на множество потоков, распределения потоков, контроля загрузки ядер и т. д. При "нативном" распараллеливании фрагменты кода единой программы назначаются с очень низкими накладными расходами на множество исполнительных ядер [9]. Ядра при этом получают указатель на фрагмент кода, передаваемого им для исполнения. Результаты сохраняются в общей памяти. В относительно устоявшейся терминологии это называется подход на базе "легких потоков". Такой подход реализован, например, в библиотеке "легких тредов" Qthreads [6], созданной в Sandia National Laboraroties. Его основная идея — обеспечить механизм порождения параллельно исполняемых потоков на уровне программы, а не на уровне ОС. В идеале накладные расходы на вызов такого потока сравнимы с затратами при вызове обычной функции. В рамках формализма Qthread порождение потока описывается вызовом вида

```
void start_thread(int(*)(int), int, ...)
```

Вызов может быть из любого ядра. Исходной функции нужно передать указатель на запускаемую функцию, число аргументов и собственно аргументы, если таковые имеются. Такой вызов совместно с атомарными, не блокируемыми операциями инкремента и некоторыми другими стандартными операциями создает возможность порождать (spawn) и завершать (join) группы параллельно выполняемых потоков, создавая последовательно-параллельную программу.

Синхронизация, которая выполняется на базе примитивов ОС, как в Unix-процессах и p-тредах [10], вызывает большие задержки. В рамках предлагаемой модели вычисления, проводимые каждым процессором, представляются легким потоком — тредом. Для синхронизации тредов при этом предлагается использовать механизм межтредовой синхронизации и коммуникации на базе разделяемой памяти. Его суть в добавлении к каждому слову памяти дополнительного бита, принимающего значение full/empty (FE-бита) и использовании наряду с традиционными синхронизирующих команд обращения к памяти [11]. Команды writeef, readfe, readff и writeff, обращающиеся к ячейке памяти, могут выполняться только при определенном в них в первом компоненте суффикса значении FE-бита и оставляют после выполнения значение этого бита, заданное программистом во втором компоненте суффикса команды. Выполнение команды задерживается, если FE-бит не имеет требуемого значения. Например, команда writeff требует, чтобы перед ее выполнением значение FE-бита слова памяти, в которое будет запись, было full и оставляет после выполнения это же значение. Значение FE-бита устанавливает обычно, что ячейка памяти имеет содержимое (full) или нет (empty).

Синхронизация на базе FE-бита не требует специальных разделяемых переменных, таких как замки, семафоры и др., а также механизмов синхронизации весьма затратных по времени их определения и исполнения при миллионах тредов [6, 11]. Кроме того, применение разделяемых переменных и неделимых (атомарных) последовательностей команд, определяющих значение условия ветвления и выполняющих переход в соответствии с полученным значением, существенно сложнее и более длительно, чем выполнение команд доступа к памяти при синхронизации динамически порождаемых легких тредов. Поэтому синхронизация на базе FE-бита позволит выдавать максимально возможное в исполняемой программе число обращений к памяти, генерируемых легкими тредами, и параллельно выполнять эти доступы в расслоенной памяти.

Следует отметить, что в CrayXMT [11] расширение языка C для использования синхронизации на базе FE-битов реализовано как введение синхронизирующих переменных *x*$ и аппаратных функций (*generic functions*) для выполнения операций чтения и записи. Среди них отметим:

- purge *x*$ — присвоение FE-биту *x*$ значения empty;
- writeqr *x*$, g — запись в *x*$ значения *g*, если значение FE-бита *x*$ равно *q* или ожидание записи, пока значение FE-бита не станет *q*; после записи значение FE-бита становится равным *r*;
- readqr *x*$ — чтение значения *x*$, если значение FE-бита *x*$ равно *q* или ожидание чтения пока значение FE-бита не станет *q*; после чтения значение FE-бита становится равным *r*.

Например, фрагмент программы [11], записывающий по адресу *i* значение 2 в случае, если FE-бит равен full, и оставляющий значение этого бита не измененным, приведен ниже:

```
int i;
writeff(&i, 2);
```

Использование синхронизирующих переменных имеет свои особенности. Так, в работе [11] приводится пример условного оператора if ((*x*$ >= 10)&&(*x*$ <= 100)){}, который корректно выполняется, если программист задумал использовать два разных значения переменной *x*$ при условии, что есть другой процесс, записывающий новое значение в *x*$ после выполнения проверки *x*$ >= 10.

Если же программист хотел только проверить принадлежность *x*$ интервалу [10, 100], то это будет дедлок, избавить от которого может правильное программирование:

```
tmpx = x$;
if ((tmpx > = 10) && (tmpx < = 100)){}
```

Синхронизирующие переменные могут использоваться как семафоры для создания барьеров и критических интервалов, организующих исключительный доступ одного процесса из множества процессов к разделяемым этим множеством процессов переменным. Кроме этого, синхронизирующие переменные могут использоваться для задания потоковых (*dataflow*) вычислений. Например, вычисление $F(i, j) = 0{,}25(F(i - 1, j) + F(i - 1, j - 1) +$

+ $F(i, j-1)$ + $F(i, j)$ в области $0 < i < n + 1$, $0 < j < n + 1$ при заданных значениях $F(0, 0)$, $F(0, j)$, $F(i, 0)$ и установленных у $F(0, 0)$, $F(0, j)$, $F(i, 0)$ значениях FE-битов, равных full, может быть представлено следующим образом (символ $ используется для обозначения номера текущего процесса [8]):

```
spawn (1, n) {
int i;
i = $;
spawn (1, n) {
int j;
j = $;
purge (&(F[i, j]); // установка FE-битов, равных empty,
// в области 0<i<n + 1, 0<j<n + 1
}
}
spawn (1, n) {
int i;
i = $;
spawn (1, n) {
int j;
j = $;
double Left = readff(&(F[i, j-1]);
double BottomLeft = readff(&(F[i-1, j-1]);
double Bottom = readff(&(F[i-1, j]);
double t = readee(&(F[i, j]);
t = 0.25*(t + Left + BottomLeft + Bottom);
writeef(&(F[i, j]), t);
}
}
```

Программа порождает n тредов, каждый из которых порождает по n тредов. На первом шаге у ячеек памяти, хранящих $F(i, j)$ в области $0 < i < n + 1$, $0 < j < n + 1$, устанавливаются FE-биты, равные empty. Далее, аналогично предыдущему шагу, проводится порождение $n \times n$ тредов. В каждом треде вычисляется соответствующая функция. Сначала вычисляется $F[1, 1]$, так как исходно $F(0, 0)$, $F(0, 1)$, $F(1, 0)$ имеют FE-биты, равные full. Затем параллельно могут быть вычислены $F[2, 1]$ и $F[1, 2]$, так как появилось необходимое для их вычисления значение $F[1, 1]$ с FE-битом, равным full, затем параллельно могут быть вычислены $F[3, 1]$, $F[2, 2]$, $F[1, 3]$ и так далее.

Представляется, что рассмотренная модель параллельного программирования на базе PRAM, реализованная путем расширения языка C [8] функциями порождения, завершения асинхронных легких тредов, а также синхронизации атомарными операциями и доступа к памяти с использованием FE-битов, в достаточной мере удовлетворяет требованию "пользователь должен только указывать, какие вычисления можно проводить параллельно, сообразуясь только с выбранным алгоритмом".

Рассмотрим, как возможно реализовать функционирование в режиме потока запросов к памяти, обеспечиваемое рассмотренной моделью программирования, в условиях ограничений, обусловленных современной элементной базой.

Согласно представлению об архитектуре экзафлопсного компьютера (рис. 10), он строится из вычислительных модулей (ВМ), каждый из которых имеет один или несколько процессорных кристаллов с подсоединенными к ним блоками памяти и интерфейсами коммуникационной среды, объединяющей все вычислительные модули [7, 9, 12, 13].

Процессорный кристалл содержит накристальную коммуникационную сеть (*interconnect*), объединяющую процессорные элементы (ПЭ), состоящие из вычислительных ядер с подсоединенными к ним блоками локальной памяти, в том числе кеш-памяти, специальные вычислительные устройства, один или несколько блоков управления памятью (*Memory Management Unit* — MMU). Необходимость введения блоков памяти, создающих эффект большого процента не переключающихся в каждом такте транзисторов в занимаемой ПЭ локальной области кристалла, создает дополнительные эффекты.

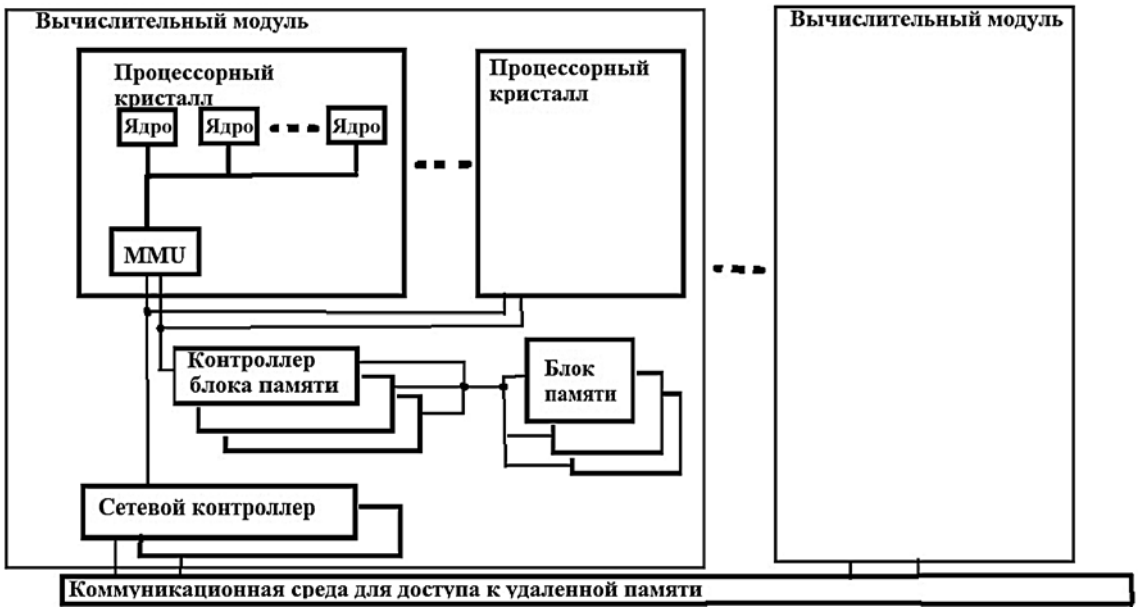

Рис. 10. Архитектура экзафлопсного суперкомпьютера

Короткие проводники передачи команд и данных между ядрами и блоками локальной памяти, к которым они обращаются, обеспечивают высокие энергоэффективности и тактовую частоту. На кристалле может разместиться много (тысячи) указанных ПЭ, функционирование которых требует загрузки и выгрузки их блоков памяти для смены программ и данных, а также инициализации выполнения в них программ. Такие архитектурные решения использованы в кристаллах разной специализации, например, PEZY-SC2 [14], ориентированном на численные методы решения дифференциальных уравнений, и Graphcore Colossus IPU [15], созданном для решения задач искусственного интеллекта. Специализация архитектуры кристалла определяет, служит ли он только полем однородных ПЭ, загружаемых и управляемых из внешнего по отношению к кристаллу хост-компьютера, как Graphcore Colossus, либо реализуется иерархическая гетерогенная кластерная архитектура, в которой на каждом уровне иерархии вычислительные устройства имеют локальные блоки памяти, как в PEZY-SC2. В этом варианте архитектуры связь между хост-компьютером и специализированным кристаллом частично перекладывается на управляющие процессоры кластеров, что снижает требования к производительности хост-компьютера и пропускной способности канала связи между ним и специализированным кристаллом или множества объединенных специализированных кристаллов.

Вычислительный модуль имеет также необходимое для обеспечения требуемой пропускной способности число смарт-контроллеров блоков распределенной разделяемой памяти, собственно блоки этой памяти и один или несколько сетевых контроллеров.

Распределенная разделяемая память суперкомпьютера состоит из совокупности блоков памяти, размещенных в каждом ВМ. Число блоков памяти в каждом ВМ, с одной стороны, должно быть как можно больше, чем обеспечивается потенциальная возможность обслуживания большего числа запросов к памяти. С другой стороны, следует иметь в виду, что их число ограничивается сложностью управления.

При инициации суперкомпьютера выполняется конфигурация глобального адресного пространства путем распределения адресного пространства по блокам памяти. Это распределение фиксируется в блоках управления памятью MMU. Каждое ядро из числа размещаемых на кристалле при выполнении текущим тредом команды доступа к разделяемой памяти по чтению или по записи после выдачи обращения к памяти приостанавливает этот тред до получения ответа от памяти. Обращение треда к памяти помещается в очередь необслуженных запросов к памяти в накристальном блоке управления памятью MMU.

На основе задаваемого при инициализации распределения глобального адресного пространства по блокам памяти блок управления памятью определяет, куда идет обращение — к локальному или удаленному блоку памяти другого ВМ. В последнем случае формируется обращение к сетевому контроллеру, который должен переслать в другой ВМ обращение соответствующему удаленному блоку памяти, поставив этот запрос в свою очередь необслуженных запросов. По получении ответа от удаленного блока памяти сетевой контроллер извлекает соответствующий запрос из очереди необслуженных и пересылает полученный ответ удаленного блока памяти в соответствующий блок управления памятью MMU.

По получении ответа от удаленного или локального блока памяти, блок управления памятью MMU извлекает соответствующий запрос из очереди необслуженных запросов к памяти и передает ядру, выдавшему запрос, результат обращения к памяти, делая возможным продолжение исполнения приостановленного треда.

Контроллер памяти для каждого запроса независимо может работать в обычном режиме (запросы на чтение и запись выполняются в порядке поступления, не используют дополнительные признаки) или в расширенном режиме с учетом механизма FE-битов. Такой бит сопровождает в памяти каждое слово.

Таким образом, выполнение в ядре команды обращения к памяти задерживается в ожидании ответа о завершении обращения к памяти из MMU. Смарт-контроллер блока памяти, получив обращение из MMU непосредственно, либо через сетевой контроллер, выполняет работу в обычном или в расширенном режиме с FE-битом указанной ячейки памяти. Смарт-контроллер блока памяти содержит память FE-битов слов памяти блоков памяти, которыми управляет этот контроллер. Если FE-бит не имеет требуемого значения, то обращение к памяти помещается в таблицу ожидания. Строки этой таблицы содержат собственно обращение к памяти (команду чтения/записи, адрес ячейки, значения FE-битов до выполнения обращения и по его завершении, служебную информацию для работы в расширенном режиме, а также указатель на незавершенную команду MMU). Изменение состояния FE-бита слова инициирует обработку таблицы ожидания с реализацией принятого подхода к разрешению конфликтов доступа к одной ячейке разделяемой памяти CREW с возможностью чтения одной и той же ячейки совокупностью тредов, а записи только одним из тредов. Причем число чтений передается из операции порождения тредов в служебной информации, и только при его достижении меняется состояние FE-бита слова. После обработки таблицы ожидания из нее выбирается обращение к памяти, которое удаляется из таблицы и запускается на выполнение.

Если FE-бит имеет требуемое значение, то контроллер блока памяти выполняет требуемое чтение или запись и формирует заданное значение FE-бита, если достигнуто заданное в служебной информации число чтений. Не вдаваясь в детали реализации контроллера памяти отметим, что после завершения обращения к ячейке памяти должен выполняться поиск в таблице ожидания строк, соответствующих этой ячейке памяти, и проверка возможности выполнения соответствующего обращения к памяти. Если таковое возможно, то обращение пускается на выполнение и соответствующая ему строка удаляется из таблицы. Естественно, описанные действия требуют ассоциативного поиска в таблице ожидания.

Распределенность обращений в память служит гарантией высокой производительности, пропорцио-

нальной числу используемых смарт-контроллеров блоков памяти вычислительных модулей.

Отметим отличие рассмотренного подхода от традиционной организации потоковых (*dataflow*) архитектур [16]. В традиционных архитектурах основу составляет ассоциативная память, в которую помещаются команды, ждущие готовности операндов. При готовности всех операндов команда извлекается из ассоциативной памяти и исполняется. Ясно, что число команд, одновременно извлекаемых из ассоциативной памяти, определяет производительность машины. Поэтому объем ассоциативной памяти должен быть большим. Но созданию такой ассоциативной памяти препятствуют затраты энергии и падение быстродействия при росте объема памяти.

Попытки программной реализации ассоциативной памяти на базе адресуемой памяти с использованием хеш-функций, разбиение общей ассоциативной памяти на локальные блоки и определение готовности только начальных команд целых фрагментов команд кардинально не изменили ситуацию [17, 18].

Однако подход на уровне контроллеров блоков памяти, выполняющих работу с FE-битами слов, представляется вполне реализуемым. Во-первых, блоков памяти может быть много, и во-вторых, выполняется только действительно необходимая синхронизация без какого-либо предварительного разбиения программ на фрагменты команд.

Реализация в процессоре внеочередного исполнения команд посредством резервирующей станции и использование контроллеров блоков памяти, выполняющих работу с FE-битами слов, покрывают всю функциональность традиционных dataflow-архитектур.

Заключение

Исследование и разработка языков и средств параллельного программирования требуют анализа существующих архитектурных идей, направленных на повышение производительности, создания экспериментальных суперкомпьютеров и программирования для них актуальных задач, исполнение которых на современных средствах неудовлетворительно по производительности и масштабируемости. Решение следует искать с разных сторон, меняя представление параллельных программ, способы отображения программных компонентов на ресурсы, вводя реализации синхронных и асинхронных тредов, программно-аппаратную поддержку синхронизации потоков и межпотоковых коммуникаций в целях определения удовлетворительного варианта архитектуры, ОС, Runtime-системы, средств подготовки программ и их компиляции. Иными словами, совместной разработки аппаратных и программных средств.

Развитие параллельного программирования возможно только в рамках конкретной модели, учитывающей "родовые" особенности (*intrisic*) архитектуры, обусловившей появление этой модели. Так для модели с передачей сообщений такой особенностью служит атомарность передач сообщений с возможностью продолжения исполнения только по завершении приема всего сообщения. В случае модели с разделяемой памятью, особенность заключается в реализации управления доступом к разделяемой ячейке памяти. Поэтому параллельное программирование для этих моделей строится на разных постулатах.

Наступил момент смены парадигмы суперкомпьютерных вычислений вообще и парадигмы программирования в частности. В этом отношении налицо сходство с аналогичной ситуацией начала 1990 гг., приведшей к замене векторно-конвейерных вычислений на базе парадигмы последовательного программирования, расширенной векторными операциями, на парадигму массово-параллельных вычислений на базе передачи сообщений.

Новая парадигма заключается в представлении всего возможного параллелизма обработки. Пользователь должен только указывать, какие вычисления можно проводить параллельными потоками над общей разделяемой памятью, сообразуясь только с выбранным алгоритмом. При необходимости прочитать одно и то же значение многими потоками, а потом записать вместо него в соответствующую ячейку общей памяти новое значение пользователь полагает, что механизм разрешения конфликта реализуется аппаратными средствами управления доступом к памяти.

Список литературы

1. **Евреинов Э. В., Косарев Ю. Г.** Однородные универсальные вычислительные системы высокой производительности. Новосибирск: Наука, 1966. 308 с.
2. **Фортов В. Е., Левин В. К., Савин Г. И.** и др. Суперкомпьютер МВС-1000М и перспективы его применения // Наука и промышленность России. 2001. № 11. С. 49—52.
3. **Корнеев В. В.** Архитектура вычислительных систем с программируемой структурой. Новосибирск: Наука, 1985. 166 с.
4. **Besta M., Hoefler T.** Slim Fly: A Cost Effective Low-Diameter Network Topology. USA // SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New Orleans, Louisiana, 16-21 Nov. 2014. P. 348—359.
5. **Deng Y., Guo M., Ramos A. F.** et al. Optimal Low-Latency Network Topologies for Cluster Performance Enhancement. arXiv:1904.00513v1 [cs.NI].
6. **Wheeler K., Murphy R., Thain D.** Qthreads: An API for Programming with Millions of Lightweight Threads // 2008 IEEE International Symposium on Parallel and Distributed Processing, 14-18 April 2008. URL: https://ieeexplore.ieee.org/document/4536359
7. **Корнеев В. В.** Подход к программированию суперкомпьютеров на базе многоядерных мультитредовых кристаллов // Вычислительные методы и программирование. 2009. Том 10. С. 123—128.
8. **Wen X., Vishkin U.** FPGA-Based Prototype of a PRAM-On-Chip Processor // CF '08 Proceedings of the 5th conference on Computing frontiers ACM. New York, 2008. P. 55—66.
9. **Елизаров С. Г., Лукьянченко Г. А., Корнеев В. В.** Технология параллельного программирования экзафлопсных компьютеров // Программная инженерия. 2015. № 7. С. 3—10.
10. **Institute** of Electrical and Electronics Engineers. IEEE Std 1003.1-1990: Portable Operating Systems Interface (POSIX.1), 1990. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub151-2.pdf
11. **Kopser A., Vollrath D.** Overview of the Next Generation Cray XMT // 53rd Cray User Group meeting, CUG 2011. Fairbanks, Alaska. 2011. P. 1-10. URL: https://cug.org/5-publications/proceedings_attendee_lists/CUG11CD/pages/1-program/final_program/Monday/03C-Kopser-Paper.pdf
12. **Корнеев В. В.** Модель программирования — смена парадигмы // Открытые системы. 2010. № 3. С. 29—31.
13. **Корнеев В. В.** Модель программирования и архитектура экзафлопсного суперкомпьютера // Открытые системы. 2014. № 10. С. 20—22.
14. **Torii S., Ishikawa H.** ZettaScaler: Liquid immersion cooling Manycore based Supercomputer // ISC 2017. Frankfurt am Main,

Germany, June 18—22, 2017. URL: https://www.pezy.co.jp/wp-content/uploads/2019/02/Keynote_candar2017.pdf
15. **Jia Zh., Tillman B., Maggioni M., Scarpazza D.** Dissecting the Graphcore IPU Architecture via Microbenchmarking. Technical Report. December 7, 2019. arXiv:1912.03413v1 [cs.DC] 7 Dec 2019.
16. **Gurd J., Bohm W., Teo Y.** Performance Issues in Dataflow Machines // Elsevier Science Publishers (North-Holland) Future Generations Computer Systems 3. 1987. P. 285—297.
17. **Бурцев В. С.** Выбор новой системы организации выполнения высокопараллельных вычислительных процессов, примеры возможных архитектурных решений построения суперЭВМ // Параллелизм вычислительных процессов и развитие архитектуры СуперЭВМ. М.: ИВВС РАН, 1997. С.41—78
18. **Климов А. В., Левченко Н. Н., Окунев А. С.** Преимущества потоковой модели вычислений в условиях неоднородных сетей. // Информационные технологии и вычислительные системы. 2012. № 2. С. 36—45.

Parallel Programming

V. V. Korneev, korv@rdi-kvant.ru, Research and Development Institute "Kvant", Moscow, 125438, Russian Federation

Corresponding author:

Korneev Victor V., Principal Researcher, Research and Development Institute "Kvant", Moscow, 125438, Russian Federation
E-mail: korv@rdi-kvant.ru

Received on November 11, 2021
Accepted on November 22, 2021

As a source for the introduction of parallel programming models, the article substantiates the development of the technology of VLSI and the corresponding change in the architecture of computing systems in the direction of increasing the parallelism of processing. That is, if the architecture changes significantly, then the parallel programming model should change in order to reduce the difficulty of effectively mapping program to hardware resources. A model of parallel programming on shared memory and synchronization based on FE-bits of shared memory words is considered. The architecture of the computing system of the exaflops performance level is also proposed, for which the considered programming model is adequate. The article tries to convey to the reader that this architecture originated as a synthesis of the development of VLSI technology, architecture of parallel systems and high-production parallel programming. It is indicated that when implementing the proposed architecture and programming model, an architecture with data flow processing is implemented. A parallel programming model for computing systems with local connections and optimal graphs of machine-to-machine connections are presented.

Keywords: *VLSI, architecture, parallel programming model, data flow processing, interconnection graph.*

For citation:

Korneev V. V. Parallel Programming**,** *Programmnaya Ingeneria*, 2022, vol. 13, no. 1, pp. 3—16.

DOI: 10.17587/prin.13.3-16

References

1. **Evreinov E. V., Kosarev Y. G.** *Homogeneous universal high-performance computing systems*, Novosibirsk, Nauka, 1966, 308 p. (in Russian).
2. **Fortov V. E., Levin V. K., Savin G. I., Zabrodin A. V., Karatanov V. V., Elizarov G. S., Korneev V. V., Shabanov B. M.** The MVS-1000M supercomputer and its application prospects. *Nauka i promyshlennost Rossii*, 2001, vol. 55, no. 11, pp. 49—52 (in Russian).
3. **Korneev V. V.** *Architecture of computer systems with a programmable structure*, Novosibirsk, Nauka, 1985, 166 p. (in Russian).
4. **Besta M., Hoefler T.** Slim Fly: A Cost Effective Low-Diameter Network Topology, *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis,* New Orleans, Louisiana, 16—21 Nov. 2014, pp. 348—359.
5. **Deng Y., Guo M., Ramos A. F., Huang X., Xu Zh., Liu W.** Optimal Low-Latency Network Topologies for Cluster Performance Enhancement. arXiv:1904.00513v1 [cs.NI].
6. **Wheeler K., Murphy R., Thain D.** Qthreads: An API for Programming with Millions of Lightweight Threads, *Workshop on Multithreaded Architectures and Applications at IEEE IPDPS*, April, 2008, available at: https://ieeexplore.ieee.org/document/4536359
7. **Korneev V. V.** An approach to programming supercomputers based on multicore multithreaded VLSI chips, *Computational Methods and Programming*, 2009, vol. 10, pp. 123—128 (in Russian).
8. **Wen X., Vishkin U.** FPGA-Based Prototype of a PRAM-On-Chip Processor, *CF '08 Proceedings of the 5th conference on Computing frontier, s* ACM New York, 2008, pp. 55—66.
9. **Elizarov S. G., Lukyanchenko G. A., Korneev V. V.** Technology of parallel programming of exaflops computers, *Programmnaya ingeneria,* 2015, no. 7, pp. 3—10 (in Russian).
10. **Institute** of Electrical and Electronics Engineers. IEEE Std 1003.1-1990: Portable Operating Systems Interface (POSIX.1), 1990, available at: https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub151-2.pdf
11. **Kopser A., Vollrath D.** Overview of the Next Generation Cray XMT, *53rd Cray User Group meeting, CUG 2011*, Fairbanks, Alaska, 2011, pp. 1—10, available at: https://cug.org/5-publications/proceedings_attendee_lists/CUG11CD/pages/1-program/final_program/Monday/03C-Kopser-Paper.pdf
12. **Korneev V. V.** Programming model — paradigm shift, *Otrrytye sistemy,* 2010, no. 3, pp. 29—31 (in Russian).
13. **Korneev V. V.** Programming model and architecture of an exaflop supercomputer, *Otrrytye sistemy,* 2014, no. 10, pp. 20—22 (in Russian).
14. **Torii S., Ishikawa H.** ZettaScaler: Liquid immersion cooling Manycore based Supercomputer, *ISC 2017*, Frankfurt am Main, Germany, June 18—22, 2017, available at: https://www.pezy.co.jp/wp-content/uploads/2019/02/Keynote_candar2017.pdf
15. **Jia Zh., Tillman B., Maggioni M., Scarpazza D.** Dissecting the Graphcore IPU Architecture via Microbenchmarking. Technical Report. December 7, 2019. arXiv:1912.03413v1 [cs.DC] 7 Dec 2019.
16. **Gurd J., Bohm W., Teo Y.** Performance Issues in Dataflow Machines, *Elsevier Science Publishers (North-Holland) Future Generations Computer Systems 3,* 1987, pp. 285—297.
17. **Burtsev V. S.** The choice of a new system for organizing the execution of highly parallel computing processes, examples of possible architectural solutions for building supercomputers, *Parallelism of computing processes and the development of the architecture of supercomputers*, Moscow, IVVS RAN, 1997, pp. 41—78. (in Russian).
18. **Klimov A. V., Levchenko N. N., Okunev A. S.** Advantages of a data flow computing model in heterogeneous networks, *Information technologies and computing systems*, 2012, no. 2, pp. 36—45 (in Russian).

УДК 004.414.38 DOI:10.17587/prin.13.17-31

А. А. Кодубец, аспирант, alexey@kodubets.ru, Дальневосточный федеральный университет, Владивосток

Обзор инструментальных средств поддержки в области инженерии требований для программных систем

Представлен краткий обзор инструментальных средств поддержки в области инженерии требований для программных систем. Проведена классификация и дано описание каждого класса инструментальных средств, включая краткое описание соответствующих наиболее функциональных программных решений. Сформулированы достоинства, недостатки и возможные пути улучшения существующих инструментальных средств.

Ключевые слова: *инженерия требований, системы управления требованиями, промышленно создаваемые программные системы*

Введение

Инженерия требований (*Requirements Engineering*) — это дисциплина, которая занимается систематизацией методов и подходов в области выявления, анализа, разработки, документирования и поддержания требований на протяжении всего жизненного цикла программного обеспечения (ПО). Эта дисциплина является самостоятельным компонентом не только программной инженерии, но и системной инженерии [1, с. 143].

Требования — это утверждения относительно атрибутов, свойств (функций) программной системы, которые необходимы пользователю (заказчику) для решения проблемы или достижения целей [2]. Для описания будущей системы инженерами по требованиям ПО эти утверждения структурируются и документируется в виде текста на естественном языке. Для промышленно создаваемых программных систем процесс разработки требований (*Requirements Development*) является сложным и трудоемким, поскольку такие системы имеют тысячи и более требований [3, с. 179, 4, с. 126, 5, с. 20, 6, с. 5].

Чем сложнее и масштабнее программная система, тем выше риски совершить ошибки в требованиях. Влияние таких ошибок на проект значительное. Например, сравнение исследовательских отчетов группы Стендиша с разницей в 20 лет (1994 и 2014 гг.) [3, с. 3, 7, с. 8] показало, что одной из основных причин успеха проектов было наличие явно задокументированных требований (*Clear Statement of Requirements* — 3-е место среди всех причин, 13 %), а незавершенность требований и отсутствие вовлеченности конечных пользователей стали причиной провала проектов (*Incomplete Requirements, Lack of User involvement*, 1-е и 2-е места среди всех причин, 13,1 и 12,4 % соответственно). Таким образом, требования являются одним из ключевых факторов успеха проектов. В дальнейшем по тексту термины "требования" и "инженерия требований" используются в смысловых значениях требований к программным системам.

С ранних 90-х гг. прошлого века работы в области инженерии требований привели к осознанию необходимости увеличения эффективности разработки требований посредством создания инструментальных средств (*Requirements Management Tools*) для автоматизации рутинных операций, выполняемых инженерами [8]. Как следствие, в 1993 г. вышла первая коммерчески успешная система управления требованиями Rational Doors (сейчас IBM Doors) [8], и затем было создано большое число инструментальных средств в области инженерии требований [9, 11].

Цель и проблема

С момента выхода первых систем управления требованиями сложность программных систем значительно выросла, и вместе с этим возникали новые преграды и проблемы на пути к дальнейшему увеличению эффективности работы инженеров по требованиям ПО. Одна из таких проблем разработки требований для промышленно создаваемых программных систем подробно изложена в работе [12, с. 339—341]: *проблема коммуникации, координации и контроля*. Перечислим далее ключевые факторы, вызывающие данную проблему [12].

1. Для разработки требований необходимо взаимодействие заказчика и исполнителя в целях выяснения деталей и ограничений для создаваемой системы, а также согласование задокументированных требований.

2. Для упрощения разработки программной системы осуществляется архитектурное разбиение целевой системы на подсистемы и компоненты, в том числе проводится иерархическое структурирование работ согласно созданной архитектуре.

3. Структура разделения труда и ответственности может быть достаточно сложной как *вертикально* — заказчик, подрядчики, внутренние или внешние субподрядчики, так и *горизонтально*, когда инфраструктурные компоненты пронизывают различные уровни системы, затрагивая многих участников процесса.

4. С увеличением сложности системы и числа требований увеличивается число представителей заказчика и вовлекаемых внутренних экспертов исполнителя и субподрядчиков.

5. Собирать всех участников процесса вместе по каждому вопросу неэффективно. Собрания и взаимодействие структурируются через специализацию и разделение труда с учетом структуры ответственности и решаемых вопросов.

При таком увеличении числа участников трудозатраты на коммуникацию, координацию и контроль значительно увеличиваются [12, с. 339—341]. Разработка требований начинает зависеть в основном от управления коллективом участников процесса и их эффективного взаимодействия, и таким образом, коммуникация между участниками встает на критический путь процесса разработки требований. Иногда эту проблему называют "*организационные и коммуникационные проблемы инженерии требований*" [13]. Далее для краткости будем использовать сочетание *"проблема коммуникации, координации и контроля"*.

Многие инструментальные средства в области инженерии требований *прямо* или *опосредовано* созданы для уменьшения влияния от описанной выше проблемы. Например, инструментарий автоматической проверки правил изложения единичных требований влияет на эффективность и повышает их качество, и, как следствие, более качественные спецификации улучшают коммуникацию между участниками процесса разработки. Однако в данном примере вклад в разрешение проблемы *опосредованный* или *косвенный,* поскольку речь идет именно о качестве требований, а не напрямую об улучшении эффективности *коммуникации*. Другой пример, допустим, спецификация подготовлена для формального рецензирования большим числом экспертов (т. е. координация), и инструментальное средство помогает организовать такое рецензирование с автоматическим напоминанием экспертам о необходимости рецензирования и автору о необходимости исправления замечаний, тем самым упрощая *координацию* и *коммуникацию* между участниками этого процесса. В данном примере вклад в разрешение *проблемы коммуникации, координации и контроля прямой*, поскольку речь идет именно об упрощении координации и коммуникации с экспертами во время рецензирования.

Цель обзора — рассмотрение существующих инструментальных средств поддержки в области инженерии требований для программных систем, а также анализ того, насколько они дают *прямой* вклад в уменьшение или разрешение обозначенной выше проблемы.

Структура обзора

Для поиска инструментальных средств использовались ключевые слова "Система управления требованиями" и "Requirements Management Tool", их аббревиатуры, сокращения и производные в поисковых системах Google и Яндекс, а также поиск по базам данных статей РИНЦ и Scopus. На основе множества источников (в том числе [9—11, 14—19]) был составлен начальный список, содержащий более 150 продуктов. Анализ показал, что даже широко известные инструментальные средства уже заменены на более новые, в том числе, под новыми брендами, поэтому устаревшие продукты убраны из списка. Дополнительно проверено, что поддержка и обновление каждого инструментального средства были не более чем 10 лет назад. После проверок осталось более 80 инструментальных средств в списке. Дальнейшая фильтрация списка осуществлялась по следующим критериям: 1) инструментальное средство должно содержать существенную функциональность, связанную с инженерией требований; 2) имеется поддержка английского либо русского языков; 3) инструментальное средство не имеет значительных дефектов в основных сценариях использования; 4) инструментальное средство предназначено только для гражданского применения. Это позволило сократить список до более 60 поддерживаемых инструментальных средств. Большинство из них коммерческого и иностранного происхождения, а также несколько средств с открытым исходным кодом. На следующем этапе выполнялась разработка классификации через более глубокий анализ информации на сайтах и в документации от поставщиков, а также через дополнительное тестирование (если были доступны пробные версии), и в некоторых случаях через удаленные демонстрации продуктов от поставщиков. Затем для классов с большим числом представителей выполнялась группировка на следующие подклассы: 1) наиболее функциональные продукты, имеющие наибольшее число функций и средств, чем в других продуктах, относительно инженерии требований; 2) другие продукты, предоставляющие только подмножество функций из наиболее функциональных продуктов. На основе выработанной классификации и наиболее функциональных решений каждого класса был подготовлен данный обзор. Итак, выявлены следующие классы:

1) системы управления требованиями (*Requirements Management Tool*);

2) системы управления жизненным циклом продукта и системы управления жизненным циклом программного обеспечения (*Application lifecycle management* — ALM [20] Tool; *Product Lifecycle Management* — PLM Tool; *Software Development Life Cycle* — SDLC Tool);

3) инструментальные средства моделирования и поддержки управления требованиями (*Model-based System Engineering tools* [14]);

4) интеллектуальные системы анализа и повышения качества требований (*Natural Language Processing-based tools for Requirements Analysis*);

5) системы прототипирования пользовательского интерфейса и поддержки управления требованиями (*WYSIWYG-prototyping & Requirements platform*).

В отдельные разделы статьи выделены отечественные инструментальные средства и инструменты с открытым исходным кодом.

Системы управления требованиями

Процесс разработки требований для промышленно создаваемых программных систем включает

в себя ряд типовых активностей, выполняемых инженером по требованиям ПО:

- сбор и выявление ранних требований (пожеланий заказчика), выявление целей и проблем, решаемых задач, определение окружения и ограничений через взаимодействие с представителями заказчика (интервью, опросы, анкетирование), включая импорт пожеланий из документов различных форматов (Excel, Word и т. п.);
- документирование требований, в том числе со структурированием спецификации через использование стандартных шаблонов;
- поиск источников требований, включая анализ предметной области, анализ окружения, аналитика существующих решений, стандартов, государственных регуляций в целях выявления скрытых и производных функциональных и нефункциональных требований;
- идентификация заинтересованных сторон: ответственных лиц, принимающих решение, конечных пользователей и их типов;
- обсуждение пожеланий и требований, согласование их с релевантными внутренними и внешними доменными экспертами, а также экспертами заказчика и координация в случае междисциплинарных вопросов;
- выявление зависимостей от внутренних и внешних субподрядчиков и поставщиков, координация релевантных требований с ними и подготовка компонентных требований для них;
- трассировка системных и компонентных требований, трассировка требований на ранние пожелания заказчика, бизнес-требования, пользовательские требования;
- структурирование работ над требованиями через архитектурное разбиение системы на компоненты, в том числе по необходимости эскизное проектирование и моделирование, и создание предварительной архитектуры в целях выяснения архитектурных и компонентных ограничений, влияющих на требования;
- проверка технической исполнимости требований и верификация характеристик качества единичных требований;
- контроль изменений и версионирование требований;
- задание атрибутов единичных требований (идентификатор, приоритет, источник и т. п.) и приоритизация требований на основе ограничений, в том числе бюджетов и сроков;
- в конечном итоге, валидация и согласование с заказчиком формально задокументированных требований в виде спецификаций.

Данный список может изменяться и дополняться в зависимости от специфики проекта и структуры ответственности. Многие из этих активностей включают в себя рутинные составляющие, которые возможно автоматизировать. Для определения принадлежности к классу "Система управления требованиями" использовался следующий критерий: инструмент должен предназначаться, в первую очередь, для помощи по этим типовым активностям. Итак, в результате исследования, анализа и тестирования инструментальных средств в области инженерии требований, а также изучения существующих обзоров и рейтингов [9—11, 14—19] были найдены следующее продукты.

Наиболее функциональные:

- Jama Connect [21], выпускается с 2007 г. компанией Jama Software, США;
- Requirements Management Doors [22], выпускается с 1993 г. компанией Rational (сейчас часть IBM), США;
- Dimensions RM [23], выпускается с 2009 г. компанией Serena Software (сейчас часть Micro Focus), США;
- Modern Requirements [24] (ранее inteGREAT), выпускается с 2006 г. компанией Modern Requirements, Канада;
- Visure Requirements, выпускается с 2007 г. компанией Visure Solution [25], США.

Другие системы управления требованиями:

- Accompa Requirements Management Tool [26], выпускается с 2007 г. компанией Accompa, США;
- Reqsuite RM [27], выпускается с 2015 г. компанией OSSENO Software GmbH, Германия;
- ReqView [28], выпускается с 2013 г. компанией Eccam s.r.o., Чехия;
- RequirementOne [29], выпускается компанией RequirementsOne, основанной в 2015 г., Великобритания;
- TraceCloud [30] компании TraceCloud, основанной в 2008 г., США;
- Pearls Pro [31], выпускается с 2017 г. компанией Pearl Inc., Канада.

Наиболее функциональные системы управления требованиями [21—25] помогают по типовым активностям и задачам, упомянутым выше, благодаря наличию следующих средств в них:

- редактора требований с возможностью использования шаблонов, а также задания структуры документа;
- возможности копирования и повторного использования требований;
- импорта пожеланий заказчика из MS Word, MS Excel или IBM Doors, а также экспорта спецификаций требований в MS Word и MS Excel;
- редактора иерархии документов (системные требования, компонентные требования и т. п.);
- редактора атрибутов единичных требований;
- редактора диаграмм для построения моделей (подобно MS Visio);
- версионирования изменений, включая возможность задания публичных и рабочих версий спецификаций;
- инструмента комментирования требований;
- средства построения таблиц трассировки между различными спецификациями требований (бизнес-требования, системные требования, компонентные требования и т. п.), в том числе в целях выяснения влияния изменений на проект (*Impact Analysis*);
- средства подтверждения и валидации требований (соответствия пожеланиям заказчика или бизнес-требованиям);

- инструмента согласования созданных спецификаций требований (Jama [21], Modern Requirements [24]);
- e-mail-нотификации об изменениях;
- наличия поиска и смены представления (View) на основе фильтрации по атрибутам или ключевым словам;
- средства управления рисками;
- редактора жизненного цикла для документов и проектов.

Как видно из списка, наиболее функциональные инструменты имеют значительный объем средств, прежде всего связанных с редактированием и структурированием требований. Эти средства облегчают работу инженеру по требованиям ПО. Однако средства для *прямого* уменьшения или разрешения *проблемы коммуникации, координации и контроля* либо отсутствуют полностью, либо имеют ограниченную функциональность. Например, Jama Connect [21] и Modern Requirements [24] имеют в составе средства комментирования требований, проведения рецензирования, согласования спецификаций и e-mail-нотификации. Однако отсутствуют автоматические средства проведения анкетирования заказчика, его представителей или доменных экспертов, создания фокус-групп для обсуждения подмножеств требований с владельцами (*Requirements owner*) и ответственными командами (*Responsible team*), нет средств ведения структурированной координации, нет средств коммуникации для разрешения выявленных проблем в требованиях, нет средств проверки и контроля зависимостей, и в целом каких-либо средств оперативного контроля. На практике все эти отсутствующие средства компенсируются ручной работой.

Системы управления жизненным циклом продукта и системы управления жизненным циклом программного обеспечения

Разработка требований является одной из стадий жизненного цикла ПО. Если рассматривать жизненный цикл широко, то во время разработки программных систем создается большое число различных сущностей проекта: спецификация требований; архитектура; дизайн компонентов; тесты; программный код; отгружаемые заказчику релизы и т. п. Эти сущности имеют влияние и зависимость друг на друга, поскольку одни создаются на ранних этапах жизненного цикла, другие позднее, но должны полностью согласовываться с предыдущими. Проверка такой согласованности является одним из методов обеспечения качества ПО. Таблицы трассировки между компонентами программной инженерии позволяют выполнить такую проверку. Например, можно протрассировать от теста к требованиям, от дизайна компонента к требованиям и т. п. Есть индустрии, в которых такая трассировка является обязательной, например, автомобильная [32]. Особенно важно использовать таблицы трассировки при изменении требований на поздних фазах, потому что идет каскадное влияние на все сущности проекта. Для усиления контроля между компонентами программной инженерии в целом на проекте и обеспечения такой трассировки существует класс инструментальных средств, называемый системы управления жизненным циклом ПО (*Software Development Life Cycle* — SDLC Tool) или системы управления жизненным циклом приложений для программных систем (*Application lifecycle management* — ALM Tool) и системы управления жизненным циклом продукта (*Product Lifecycle Management* — PLM Tool) для аппаратных и программно-аппаратных систем. Как правило, ALM- и PLM-продукты являются интеграцией набора самодостаточных модулей, где каждый модуль добавляет поддержку отдельных компонентов программной инженерии или системной инженерии. Для определения принадлежности к данному классу использовался следующий критерий: в инструменте должны быть как минимум средства, связывающие инженерию требований, программирование (его отслеживание) и тестирование. Значимость инструмента определялась по объему именно средств в модуле для инженерии требований. Итак, были найдены следующие программные продукты.

С точки зрения инженерии требований наиболее функциональные ALM-продукты:

- Helix ALM [33] (ранее TestTrack Pro), выпускаемый с 1996 г. компанией Perforce (ранее Seapine Software), США;
- Kovair ALM studio компании Kovair Software [34], основанной в 2008 г., США;
- codeBeamer ALM [35], выпускается с 1999 г. компанией Intland Software GmbH, Германия;
- DevSpec [36], выпускается компанией TechExcel, Inc., основанной в 1995 г., США;
- Valispace [37], выпускается компанией Valispace GmbH, основанной в 2016 г., Португалия.

ALM-продукты для медицинской индустрии:

- Orcanos ALM [38], выпускается компанией Orcanos Ltd., основанной в 2005 г., Израиль;
- Cockpit Enterprise [39], выпускается компаний Cognition Corporation, Великобритания, более 20 лет на рынке;
- Aligned Elements [40], выпускается компанией Aligned AG, основанной в 2006 г., Швейцария;
- MatrixReq ALM [41], выпускается Matrix Requirements GmbH, Германия.

Другие ALM- и PLM-продукты:

- ALM Octane [42], выпускается с 2010 г. (на базе HP Quality Central) компанией Micro Focus (ранее HP), Великобритания;
- Aqua ALM [43], выпускается с 2012 г. компанией Aqua cloud GmbH, Германия;
- ENOVIA PLM [44], выпускается компанией Dassault Systemes (ранее MatrixOne и SmartTeam с 1998 г.), Франция;
- SpiraTeam [45] компании Inflectra, основанной в 2006 г., США;
- Polarion PLM [48] компании Siemens, выпускается с 2015 г., Германия (ранее Polarion Software, США);
- ReqTest [47] выпускается с 2009 г. компанией ReQtest AB, Швеция;
- Rommana — [48], выпускается с 2011 г. компанией Rommana Software, США;

- Sox [49], выпускается компанией EnCo Software GmbH, основанной в 2005 г., Германия;
- Xebrio [50], выпускается компанией Xebrium Software, основанной в 2013 г., США;
- PTC PLM [51], выпускается с 1998 г. компанией PTC, Inc., США.

Наиболее функциональные PLM- и ALM-продукты имеют модуль для разработки и управления требованиями, представляющий собой подмножество средств из решений класса "Системы управления требованиями". Поставщики осознают ограниченность своих ALM- и PLM-продуктов относительно систем управления требованиями, и в качестве компромисса предоставляют средства интеграции с системами управления требованиями [33—35]. Преимущество ALM- и PLM-продуктов прежде всего заключается в интеграции между компонентами программной инженерии и системной инженерии и наличию функциональности шире инженерии требований, перечисленной кратко далее:

- средства управления портфолио проектов и дорожной картой организации;
- средства управления проектом, включая график проекта и его бюджет;
- средства отслеживания дефектов и задач;
- средства управления рисками проекта (*Risk Management*);
- средства управления тестированием (*Test Management*);
- средства трассировки между компонентами программной инженерии (трассировка требований на тесты, требований на дизайн и т. п.);
- функциональность взаимодействия со средствами непрерывной интеграции (*Continuous Integration*), управление релизами и отгрузками;
- интеграция с системами управления версиями и IDE (интегрированная среда разработки — *Integrated Development Environment*);
- интеграция с системами моделирования, импортирование и экспортирование данных из/в форматы других систем;
- средства создания отчетов и просмотра метрик проекта и ПО;
- средства задания модели жизненного цикла ПО и контроля методологии разработки (*Workflow management*);
- средства сопровождения ПО (*Maintenance, Service & Support*).

Более 100 различных интеграций с внешними программными системами имеет Kovair ALM [34], в том числе выходящие за рамки программной и системной инженерий, в частности, интеграция с ERP (*Enterprise Resource Planning*) и CRM-системами (*Customer Relationship Management*).

В медицинской индустрии есть множество сертификаций для программно-аппаратных систем, например, стандарты ISO 14971 [52] и ISO 13485 [53], в целях отслеживания рисков и требований к безопасности для человека. В связи с этим существует подкласс ALM-продуктов для медицинской индустрии [38—41], имеющие средства (встроенные базы знаний) для отслеживания соответствия нормативным требованиям.

Программно-аппаратные системы имеют множество ограничений. Например, программные ограничения на используемый размер памяти, время исполнения алгоритмов; аппаратные ограничения по энергопотреблению, размеру и массе изделия; а также экономические ограничения по себестоимости изделия. Эти ограничения могут быть предоставлены заказчиком и прописаны в требованиях, но могут быть и следствием других факторов. Они могут вызываться окружением, спецификой решаемой проблемы, спецификой создаваемого технического решения, или быть связанными с экономическими факторами. Подсистемы и компоненты, в свою очередь, имеют внутренние ограничения. Продукт Valispace [37] позволяет задавать такие ограничения параметров системы и компонентов в виде требований (значений параметров, минимум, максимум) и автоматически вычисляемых формул (подобно PTC MathCAD) на основе параметров дочерних и зависимых компонентов, а сами ограничения автоматически (каскадно) отслеживать как на уровне всей системы в целом, так и на уровне конкретных компонентов. Это важно, поскольку даже небольшие изменения в требованиях могут сильно повлиять на систему в целом. Например, требуется увеличить время работы системы от батареи в 1,5 раза, не меняя емкость батареи и стоимость изделия. Это возможно сделать как за счет программной части, оптимизируя скорость работы алгоритмов, так и за счет аппаратной части, оптимизируя схему питания и т. п., либо комбинируя решения. Таким образом, поддержка в Valispace отслеживания ограничений позволяет повысить эффективность работы инженера по требованиям ПО.

Интеграция различных компонентов программной и системной инженерии в ALM- и PLM-продуктах дает *автоматическую* возможность понять влияние изменений требований на сущности проекта за счет средств трассировки, и как следствие, это уменьшает влияние от *проблемы коммуникации, координации и контроля* на поздних этапах жизненного цикла ПО. Интеграция с внешними системами шире программной инженерии дает возможность *опосредовано* уменьшить проблему, упрощая приоритизацию требований через средства управления бюджетом и графиком проекта. Также, если новая программная система создается как замена существующей системы, которая разрабатывалась с использованием ALM- и PLM-продуктов, то можно использовать аналитику сущностей проекта, т. е. избежать некоторых рисков предыдущего проекта за счет повторного использования знаний и опыта из прошлого проекта. Таким образом, ALM- и PLM-продукты увеличивают эффективность разработки проектов, однако с точки зрения инженерии требований помогают в основном на более поздних этапах жизненного цикла ПО, а на ранних этапах, когда происходит основная работа по разработке требований ПО, эти продукты либо не дают вклада вообще, либо дают низкий вклад в уменьшение основной *проблемы,* описанной в статье.

Инструментальные средства моделирования и поддержки управления требованиями

Требования, моделирование и архитектура системы тесно связаны, особенно на ранних этапах разработки промышленно создаваемой программной системы [54]. Во время построения моделей могут выявляться дополнительные детали и ограничения. На основе требований, цели проекта и решаемой проблемы, а также моделей предметной области, ограничений и других аспектов создается архитектура программной системы. Она влияет на техническую исполнимость требований, а требования влияют на нее [54]. В одной архитектуре возможно выполнить одно подмножество требований, а в другой — другое подмножество. В реализованной и внедренной системе возможны случаи, когда новую функциональность невозможно или слишком затратно реализовывать в силу архитектурных ограничений. Моделирование и создание архитектуры являются важными шагами на ранних этапах жизненного цикла.

Существуют инструментальные средства для моделирования и документирования архитектуры, например, MS Visio и IBM Rational Architect [55]. Эти инструменты поддерживают стандартные языки моделирования UML, IDEF и SysML. Инструментальные средства моделирования и поддержки управления требованиями объединяют в себе функциональные возможности систем управления требованиями и инструментальных средств для моделирования, разработки и документирования архитектуры. В англоязычной литературе такие инструменты называют инструментами системного инжиниринга на основе моделе-ориентированного подхода или инструментами программной инженерии для разработки, управляемой моделями (*Model-based Systems Engineering tool* [14], *Model-driven development* — MOD [52]). Этот класс инструментов существенно пересекается с UML-инструментами, которых на рынке представлено достаточно много [56]. Поэтому для определения принадлежности инструмента к классу "Инструментальные средства моделирования и поддержки управления требованиями" использовался следующий критерий: в инструменте должны быть как минимум средства, связывающие инженерию требований и моделирование, при этом инструмент должен поддерживать широкий набор стандартов моделирования. Итак, были найдены следующие программные продукты.

Наиболее функциональные:

- Cradle-RM [57], выпускается с 1987 г. компанией 3SL, Великобритания;
- objectiF RM [58], выпускается с 1992 г. компанией microTOOL GmbH, Германия;
- Enterprise Architect [59], выпускается с 2000 г. компанией Sparx Systems Pty Ltd, Австралия;
- Rhapsody [60], выпускается с 1996 г. компанией IBM, США (ранее Rational, США, а еще ранее Telelogic, Швеция и I-Logix Inc, Израиль);
- Cameo System Modeler (ранее MagicDraw) [61], выпускается с 2010 г. компанией Dassault Systemes, Франция (ранее Nomagic, Inc., США).

Другие:

- PREEvision [62], выпускается компанией Vector Informatik GmbH, основанной в 1988 г., Германия;
- MacA&D and WinA&D [63], выпускается компанией Excel Software, основанной в 1986 г., США.

Продукты данного класса имеют значительную функциональность, связанную с моделированием и системным инжинирингом для *программно-аппаратных систем*. Ярким представителем этого класса является Cradle-RM [57]. Он не уступает IBM Rational Architect по функциональности, но при этом имеет дополнительные средства для инженерии требований:

- трассировку от конкретных объектов на моделях (диаграммах) до конкретных требований, тестов или других диаграмм;
- средства представления трассировки в виде диаграмм;
- редактор Requirements Diagram из SysML — т. е. поддержку представления спецификации требований в виде диаграмм;
- поддержку вычисляемых значений в спецификациях, задаваемых формулами, и вывод вычисляемых значений на этапе отображения требований, если заданы зависимые значения (подобно MS Excel);
- средства представления нескольких спецификаций требований в виде единой иерархической диаграммы при наличии связей между ними;
- средства управления рисками — связь рисков и требований;
- средства управления тестированием.

Есть инструментальные средства данного класса, которые специализируются только на автомобильной индустрии и помогают следовать обязательным стандартам и регламентам. Например, решение PREEvision [62], имеющее в одном инструменте функциональность по разработке требований и созданию программных, аппаратных и гибридных программно-аппаратных моделей и архитектур согласно стандарту ISO 26262 [32].

Как было сказано ранее, промышленно создаваемые программные системы имеют большие спецификации требований, множество подсистем и компонентов, которые в свою очередь имеют отдельные спецификации, дизайн и свою архитектуру. Системы данного класса позволяют эффективно связать все эти сущности проекта вместе. Однако данный класс инструментальных средств только *опосредованно* уменьшает влияние от *проблемы коммуникации, координации и контроля* за счет улучшения структурирования информации о создаваемой системе. В большинстве инструментов *прямые* средства уменьшения влияния от *проблемы* либо отсутствуют полностью, либо минимальны и ограничены следующим: возможностью добавлять комментарии к сущностям и e-mail-нотификациям об изменениях. Исключением является Enterprise Architect [59], в котором есть возможность создавать чаты и привязывать их к любым сущностям проектируемой системы, а также проводить рецензирование этих сущностей. Это улучшает сферу коммуникации, но не охватывает все типовые активности, перечисленные ранее, и, как следствие, инженер по требованиям ПО продолжает работать над ними в ручном режиме.

Интеллектуальные системы анализа и повышения качества требований

Разрабатывая требования, инженер анализирует решаемую задачу проекта, проблему и предметную область, все поступающие вводные от заказчика, экспертов для выявления пожеланий и требований и их документирования в виде спецификации на естественном языке, а также согласования их с целями заказчика и исполнителя. Это высокоинтеллектуальная работа. Как показал обзор исследований в области инженерии требований [12, с. 342], существует перспективное направление "Обработка требований как текста на естественном языке" (*Natural Language Processing For Requirements Engineering* [6] — NLP4RE) и соответствующий класс инструментов "Интеллектуальные системы анализа и повышения качества требований", которые решают некоторые задачи инженерии требований с использованием методов машинного обучения, искусственного интеллекта и методов обработки естественного языка (подобно системам проверки правописания). Например, в этих исследованиях и инструментах решаются следующие задачи: проверка изложения единичных требований, поиск дублирования в требованиях, оценка сложности единичных требований и т. п. Итак, были найдены следующие программные продукты данного класса.

Наиболее функциональные:

- QVscribe [64], выпускается компанией QRA Corp, основанной в 2012 г., США;
- Raven for Microsoft Office [65], выпускается компанией Ravenflow, основанная в 2000 г., США;
- VT Docs [65], выпускается компанией Visible Thread, основанной в 2008 г., Ирландия;
- RQA Quality Studio [67], выпускается компанией Knowledge Centric Solutions, S. L., основанной в 1999 г., Испания.

Другие инструменты:

- Reqsuite QC [68], выпускается с 2014 г. компаний OSSENO Software GmbH, Германия;
- Requirements Quality Assistant [69], является частью IBM Doors Next Generation.

Одним из наиболее функциональных представителей данного класса инструментальных средств является QVscribe. Он дополняет ранее описанные классы инструментальных средств и может быть использован в комбинации с ними: существуют интеграции с MS Word, MS Excel, Jama и др. QVscribe предоставляет следующие *автоматические* средства:

- проверку соответствия изложения требований принятым в индустрии практикам, например, INCOSE [70], EARS [71];
- поиск похожих требований в спецификациях, чтобы избегать дублирования, на основе вычисления дистанции сходства;
- проверку единиц измерения величин, например, чтобы не было указания в одном месте метров, а в другом сантиметров в одном и том же контексте;
- выявление в тексте просторечий, негативного изложения, сложного изложения в предложениях с указанием времени (когда, до, после);
- выявление в тексте степени незавершенности единичных предложений;
- проверку корректности изложения неявных ссылок ("сделать также как там", когда не ясно где это "там");
- проверку в целом степени ясности изложения на основе нахождения неоднозначных слов, выражений и нахождения неясной терминологии;
- возможность предоставления отчетов о качестве требований.

Во время данного исследования автором были привлечены опытные инженеры по требованиям ПО для тестирования QVscribe. В эксперименте QVscribe анализировал несколько десятков страниц текста требований разных проектов. Все автоматически найденные дефекты (46 шт.) были проанализированы инженерами. Результат анализа показал, что QVscribe находит много ошибок первого рода (ложные срабатывания), но также находит около 30 % настоящих ошибок в изложении. Результаты данного эксперимента показывают, что часть интеллектуальных задач инженера по требованиям ПО может быть автоматизирована.

Другим оригинальным представителем данного класса является Raven for Microsoft Office [65]. Он имеет следующие функциональные средства: автоматический поиск ошибок последовательности изложения и пропусков в требованиях с анализом нарратива (повествования) текста; автоматическое выявление акторов, объектов и зависимостей между ними; автоматическое создание словарей терминов из текста требований; автоматическое создание кросс-функциональных блок-схем из текста требований.

Большинство инструментов данного класса работает с английским языком, и поставщики этих средств позиционируют их на рынке как инструментальные средства для работы со спецификациями требований. Однако RQA Quality Studio [67] может работать со спецификациями, написанными на французском, немецком, японском, испанском, шведском, итальянском и нидерландском языках. А VT Docs [66] позиционируется на рынке шире, чем только анализ спецификаций требований. Например, VT Docs может анализировать запросы на предложения (*Request for Proposal*) и улучшать качество текста контрактов.

Инструментальные средства данного класса помогают справляться с рутинными задачами инженера по требованиям ПО, но дают лишь *опосредованный* вклад в уменьшение *проблемы коммуникации, координации и контроля* за счет улучшения качества изложения требований. Однако методы NLP4RE могут использоваться как компонент или зависимость для разрешения данной *проблемы*. Например, при наличии средств автоматического выявления релевантных экспертов (владельцев) на основе классификации требований методами машинного обучения [72] становится возможным упрощение координации работ, т. е. автоматическое извещение экспертов о требованиях, относящихся к их экспертизе, с запросом обратной связи и т. п.

Системы прототипирования пользовательского интерфейса и поддержки управления требованиями

Промышленно создаваемые системы могут быть без пользовательского интерфейса или с ограниченным пользовательским интерфейсом (UI), например, роботы, автономные системы, где взаимодействие — это конфигурация, которая задается один раз на этапе производства или внедрения. Для такого типа систем нет смысла детально прорабатывать UI. Однако в большинстве своем программные и программно-аппаратные системы имеют интерфейс человеко-машинного взаимодействия. Например, веб-интерфейс, десктоп UI, UI мобильного приложения, дисплей с сенсорным экраном, распознавание жестов и т. п. Сразу несколько вариантов UI из этого списка может поддерживаться в одной системе, включая множество различных экранных форм, которые должны быть реализованы в единообразном стиле на разных платформах, но каждая из платформ может иметь свои ограничения. Разработка эстетической составляющей UI — это ответственность дизайнеров (промышленных дизайнеров, дизайнеров пользовательского интерфейса). Задачи инженера по требованиям ПО, специфичные для области UI, состоят в следующем:

- документирование и анализ пожеланий заказчика относительно UI с проактивным предложением подходов, включая эргономику и учитывая принципы, технологии и знания из инженерной психологии и компьютерной графики;
- оформление концепции и архитектуры UI в виде UML-диаграмм вариантов использования, деятельности и машин состояний и т. п.;
- обсуждение и согласование архитектуры и концепций UI с экспертами и заказчиком;
- прототипирование UI (экранных форм) на основе концепции, уточняя требования, выявляя дополнительные ограничения к создаваемой системе и при необходимости привлекая эстетического дизайнера;
- документирование требований к UI в виде спецификаций, включая экранные формы, либо выделяя их в отдельный документ, и в итоге, согласование этих документов с заказчиком.

Для того чтобы запрототипировать экранные формы существует много инструментальных средств, начиная от средства широкого профиля MS Visio и заканчивая множеством специализированных средств [73] под специфичные платформы или инструменты, совмещенные с IDE, в том числе с поддержкой принципа WYSIWYG. Некоторые из этих инструментов имеют средства поддержки инженерии требований. Для определения принадлежности инструмента к классу "Системы прототипирования пользовательского интерфейса и поддержки управления требованиями" использовался следующий критерий: в инструменте должны быть как минимум средства, связывающие инженерию требований и прототипирование UI. Итак, были найдены следующие программные продукты данного класса:

- Irise [74], выпускается компанией Irise, основанной в 1997 г., США;
- Justinmind [75], выпускается компанией Justinmind, основанная в 2005 г., США.

Irise имеет широкую функциональность для прототипирования UI для веб-приложений, мобильных платформ и десктоп-платформ. Непосредственно для инженерии требований в данном инструменте предусмотрена следующая функциональность:

- редактор требований в контексте WYSIWYG-редактора прототипирования экранных форм с возможностью привязки единичных требований к конкретным элементам на экранной форме;
- интеграция и синхронизация с системами управления требованиями и ALM-системами (Jama, Dimensions RM);
- комментирование требований;
- приоритизация требований;
- поддержка UML и BPMN (*Business Process Model and Notation*);
- контроль версий, включая средства просмотра изменений в требованиях и экранных формах;
- поддержка отправки нотификаций в Skype и Slack;
- импорт требований из MS Excel и экспорт требований в MS Word;
- генератор отчетов по статусу разработки требований (если есть интеграция с ALM-системами);
- средства задания контроля методологии разработки.

Итак, с точки зрения инженерии требований инструменты данного класса имеют малое подмножество функциональности из решений класса "Система управления требованиями", поэтому они и имеют интеграцию с ними. Наличие средств коммуникации — комментирования, веб-интерфейса нотификаций через службы обмена мгновенными сообщениями, e-mail-нотификации — все это улучшает *сферу коммуникации*, но в основе инструментов данного класса находится редактор экранных форм, и в нем отсутствуют механизмы оперативного *контроля* и *координации* групповой деятельности, которые необходимо выполнять вручную.

Отечественные инструментальные решения

В силу экономических причин иностранные инструментальные средства имеют сравнительно высокую цену для российского рынка. Также ограничением является тот факт, что только некоторые иностранные инструментальные средства [22] локализованы для Российской Федерации (РФ). Исследование показало, что найденные отечественные инструментальные средства для инженерии требований для программных систем имеют небольшую функциональность в сравнении с наиболее функциональными решениями, описанными ранее (Jama [21], Helix ALM [33], CradleRM [57]), но имеют доступную цену [76] и поддержку русского языка и государственных регламентов. Отечественные инструментальные средства рациональнее использовать, когда инженерная документация между заказчиком и исполнителем ведется на русском языке. Итак,

найдены следующие инструментальные средства для программных систем:

Коммерческие:

- Devprom ALM, выпускается с 2008 г. компанией ООО Девпром [76];
- Система управления требования на платформе Техэксперт компании АО Кодекс [77], работающей на рынке с 1991 г.

Некоммерческие:

- Requality [78], выпускается с 2010 г. институтом системного программирования им. В. П. Иванникова РАН.

Продукт компании Девпром — это один из наиболее функциональных представителей отечественных решений. Он попадает в класс "Системы управления жизненным циклом программного обеспечения" и имеет средний объем функциональности относительно наиболее функциональных продуктов данного класса. Однако если в этом инструменте переключить язык интерфейса с русского на английский, то существенная часть функциональности становится недоступной.

В рамках анализа отечественных решений ниже представлены инструментальные средства в области инженерии требований для аппаратных систем (механических и электронных изделий):

- Лоцман: PLM [79], выпускается с 2003 г. компанией АСКОН;
- Союз-PLM [80], выпускается с 2010 г. объединением PLM-союз;
- T-Flex RM [81], выпускается с 1992 г. компанией ЗАО Топ Системы.

Отсутствие программно-аппаратных аспектов в этих продуктах является существенным недостатком. Даже в инженерных областях, где традиционно использовались только механические или электронные компоненты, все чаще используются программно-аппаратные решения. Например, автомобильная подсистема для управления впрыском топлива программируется на микроконтроллерах [82]. Программно-аппаратные системы дают гибкость, позволяют создавать системы с алгоритмами нового типа и существенно сокращают жизненный цикл разработки систем [82].

Итак, на текущий момент найденные отечественные инструментальные средства в области инженерии требований *для программных систем* в основном находятся на начальных этапах развития, но уже могут использоваться для повышения эффективности работы.

Инструментальные средства с открытым исходным кодом

В процессе работы над данным обзором стало понятно, что многие коммерческие продукты (например, Jama Connect [21], IBM Doors [22], Modern Requirements [24], Helix ALM [33] и др.) в области инженерии требований имеют высокую стоимость лицензий на одного пользователя в год, а также — ограничения на минимальное число покупаемых лицензий. Для небольших организаций такая стоимость лицензий будет преградой. В связи с этим далее рассмотрены бесплатные инструментальные средства.

В статье [83] рассматривается 20 инструментальных средств в области инженерии требований с открытым исходным кодом и утверждается, что большинство из них либо заброшены авторами, либо имеют низкое качество. В целом это верно, но есть исключения, которые не описаны в работе [83], и будут описаны здесь. Авторы статьи [83] отметили пять проектов. После перепроверки этого только три проекта являются релевантными к ранее описанным классам:

- OSRMT (*Open Source Requirements Management Tool*) [84] — система управления требованиями с Desktop UI, последние изменения в 2020 г., исходный код под лицензией GPL, а с 2014 по 2016 гг. выпускается параллельная версия с Web UI — aNimble Platform [85];
- ReqHeap (*Requirement Heap*) [86] — система управления требованиями, последние изменения в 2013 г., исходный код под лицензией GPL версии 2.0;
- OpenReq!Live [87] — набор инструментов, связанных с инженерией требований, последние изменения в 2021 г., исходный код под лицензией ESL версии 2.0.

Другие инструментальные средства:

- Modelio [88] — инструментальное средство моделирования и поддержки управления требованиями, выпускается с 2009 г. компанией Softteam, Франция, исходный код под лицензией GPL версии 3.0;
- RMF (*Requirements Management for Eclipse*) [89] — система управления требованиями, разрабатывается с 2014 г. некоммерческой организацией Eclipse Foundation, Канада, исходный код под лицензией ESL версии 1.0, последние изменения в 2016 г.;
- ReqT [90] — экспериментальный инструмент, позволяющий задать требования на формальном декларативном языке и автоматически сгенерировать из этого диаграммы (подобно UML/SysML), выпускается с 2010 г. Лундским университетом, Швеция, исходный код под лицензией BSD 2-clause, последние изменения в 2014 г.;
- Fret [91] (*Formal Requirements Elicitation Tool*) — инструмент, позволяющий задать требования на формальном ограниченном языке и автоматически из этого создавать модели для симуляции поведения системы: блок диаграммы или временные диаграммы сигналов (формы волны, Waveform); выпускается с 2019 г. при поддержке NASA, исходный код под лицензией Nasa Open Source Agreement версии 1.3;
- jUCMNav [92] плагин для Eclipse — инструментальное средство моделирования и поддержки управления требованиями, разрабатывается с 2006 г. под лицензией ESL версии 1.0, последние изменения в 2021 г.

Modelio по функциональности имеет значительные средства для моделирования, включая поддержку UML и BPMN. Однако он имеет менее функциональный редактор требований, чем у решений из класса "Системы управления требованиями".

С точки зрения *основной проблемы статьи*, наиболее релевантное инструментальное средство — это OpenReq!Live. Оно разрабатывается группой специалистов четырех университетов различных стран (Финляндия, Испания, Германия, Австрия), а также поддерживается пятью коммерческими компаниями (Siemens, The QT Company и др.). Этот инструмент — это набор экспериментальных модулей (изолированных микросервисов) различных исследователей в области инженерии требований, объединенных под одним брендом. Цель OpenReq — это создание инструментов поддержки разработки требований для открытых систем (*Community based Requirements Engineering*), например, в проектах с открытым исходным кодом, когда нет единого лица принимающего решения. Для этой цели в OpenReq!Live есть следующая экспериментальная функциональность:

- групповое принятие решений через голосование, включая средства определения конфликтов и возможность делегирования своего голоса;
- средства импорта пожеланий из социальных сетей (Twitter) — идея того, что конечные пользователи оставляют обратную связь в социальной сети (жалобы, комментарии) и возможно ее использовать для формирования новых требований;
- средства для автоматического нахождения ответственных лиц на основе данных в системе (истории участия в проектах), использующие NLP4RE методы;
- средства для автоматического нахождения связанных (зависимых) требований, проверки правил изложения единичных требований, нахождения дублирующих требований с использованием NLP4RE-методов.

В большинстве случаев найденные инструментальные средства имеют экспериментальный характер. Они мало полезны на практике в проектах промышленно создаваемых программных систем, за исключением продукта Modelio, который был коммерческим проектом до того, как его исходный код поместили в открытый доступ. Исследовательский проект OpenReq!Live касается основной *проблемы,* описанной в статье. Перечисленные выше средства в нем *прямо* помогают по некоторым аспектам сфер *координации* и *коммуникации*, в частности, голосование и автоматическое нахождение ответственных лиц. Однако в силу ориентации на сообщества разработчиков с открытым исходным кодом данный инструмент не имеет средств для взаимодействия с заказчиком, отслеживания внешних и внутренних зависимостей, отслеживания технической исполнимости требований, и в целом инструмент не имеет средств оперативного контроля. Фактически он является ранним *прототипом* для частичной демонстрации идей, а не готовым продуктом, который можно внедрять.

Другие инструментальные средства

Представленный обзор инструментальных средств сфокусирован на самых больших классах по количеству функциональности. Однако есть и другие инструментальные средства в области инженерии требований, которые напрямую не попадают в описанные раннее классы. Такие инструменты были рассмотрены, классифицированы и наиболее функциональные представители отмечены далее:

- Agile-инструменты с поддержкой инженерии требований, например, Aha! Roadmap [93], выпускается компанией Aha!, основанной в 2013 г., США;
- инструментальные средства с фокусом на бизнес-моделирование процессов в целях дальнейшего создания информационных систем, например, Blueprint [94], выпускается компанией Blueprint Software Systems, Inc., основанной в 2004 г., Канада;
- плагины для Atlassian JIRA, добавляющие поддержку инженерии требований, например, R4J [95] компании Ease solutions Pte Ltd, основанной в 2007 г., Сингапур;
- плагины для MS Word, добавляющие утилиты для инженерии требований, например, Reqchecker [96] компании Khilogic, Франция;
- редакторы ReqIF-формата (открытый формат для хранения требований Requirements Interchange Format [97]), например, ReqEdit [98] компании ReqTeam GmbH, основанной в 2014 г., Германия;
- средства трассировки документов, в первую очередь требований на дизайн и тесты, например, Reqtify [99], поддерживает более 100 форматов файлов проектных сущностей, выпускается компанией Dassault Systèmes® (до 2013 г. Geensoft), Франция;
- конверторы форматов требований — импорта данных из PDF, MS Word и т. п., и экспорт в ReqIF, IBM Doors и Jama, с возможностью сравнения версий файлов (Diff), например, ReqMan [100] компании em AG, Германия;
- инструменты сбора и оценки идей (*Idea Management*), в конечном итоге, конвертации их в требования, например, Aha! Idea [101], выпускается с 2020 г. компанией Aha!, США.

Заключение

Как отмечено в работе [4], во многих организациях процесс разработки требований выполняется несистемно и неформально — в ручном режиме происходит документирование требований в стандартных офисных пакетах. При увеличении масштаба промышленно создаваемой программной системы дополнительно возникает *проблема коммуникации, координации и контроля*, которая становится на критическом пути процесса разработки требований и только усугубляет ситуацию с ручной работой. Как показал обзор, чтобы помочь с этим существует множество инструментальных средств, в основном представляющих собой специализированный редактор требований и таблицы трассировки для упрощения простых рутинных операций. Большинство инструментов начинали разрабатываться более 10—15 лет назад, когда даже для простых операций было мало автоматизации. Сейчас видно, что поставщики встречаются с *проблемой коммуникации, координации и контроля*, поскольку в некоторых продуктах добавляются соответствующие средства для упрощения этих аспектов. Для наглядности приве-

дена релевантная функциональность во всех этих инструментальных средствах:

- сфера коммуникации — комментирование единичных требований, комментирование спецификаций, e-mail-нотификации об изменениях, нотификации в мессенджеры, средства импорта пожеланий и жалоб из социальных сетей (экспериментальный прототип [87]);
- сфера координации — рецензирование требований, голосование, автоматическое выявление ответственных лиц (экспериментальный прототип [87]), сбор идей (отдельный продукт [101]);
- сфера контроля — встроенные средства контроля отсутствуют, но у многих инструментов есть интеграция с внешними системами (JIRA), не учитывающие аспекты инженерии требований.

Как следует из списка, существующие инструментальные средства не помогают по всем типовым активностям инженера по требованиям ПО, поскольку в каждом инструменте есть только часть функциональности, приведенной в списке. Поэтому можно сделать вывод, что не существует системного решения проблемы.

Далее перечислены основные недостатки существующих инструментальных средств:

- слабая интеллектуальная составляющая — в большинстве систем не используются достижения методов машинного обучения и искусственного интеллекта, либо используются только для проверки качества единичных требований [64, 66—69], а не для решения *проблемы коммуникации, координации и контроля* (подобно [72]);
- отсутствует или слабо используются базы знаний и связь с предметными областями (за исключением решений для автомобильной [62] и медицинской индустрий [38—41]);
- некоторые из систем [22, 25, 49, 51, 59—65, 67, 75] традиционно разрабатывались как десктоп-приложения и имеют слабый Web UI или не имеют его вообще, аналогично, отсутствует мобильный UI и приложения для мобильных телефонов и планшетов, необходимые для работы в полевых условиях с заказчиком и доменными экспертами, когда нет возможности использовать ноутбук или персональный компьютер;
- ограниченные средства коммуникации — имеются e-mail-нотификации, средства комментирования единичных требований, но отсутствует двунаправленная интеграция с мессенджерами и другими современными средствами связи;
- слабые или отсутствующие механизмы координации с заказчиком и экспертами доменных областей — в некоторых решениях [21, 24, 33, 59] есть средства проведением рецензирования, но нет возможности проведения опросов, создания фокус-групп для решения вопросов технической исполнимости, нет средств создания комитетов для решения междисциплинарных вопросов; многие инструментальные средства ориентированы на одного пользователя, а не на совместную коллективную работу, которая необходима для разработки промышленно создаваемых систем.
- ограниченные средства контроля, за исключением ALM- и PLM-систем, где контроль выполняется через трассировку между компонентами программной и системной инженерии, нет средств оперативного контроля того, что задача выполнена в срок, либо наличие роли контролера в системе с необходимыми средствами поддержки.

Для уменьшения влияния *проблемы коммуникации, координации и контроля* нужны дальнейшие исследования. Необходимо сделать следующее (подобно, как было указано в работе [12, с. 346]): провести анализ и систематизацию структур коммуникации участников процесса, систематизировать типовые структуры координации, выявить активности (коммуникации и координации), дающие наибольший вклад в критический путь процесса разработки требований. На основе этого необходимо создание новых подходов и интегрированного инструментального средства поддержки коммуникации, координации и автоматизированного контроля для инженерии требований, в том числе с исправлением перечисленных выше недостатков существующих инструментальных средств, что сделает возможным сокращение критического пути процесса разработки требований.

Несмотря на все выявленные недостатки существующих инструментальных средств и отсутствие в них системного решения основной *проблемы* статьи, внедрение таких инструментальных средств, в особенности наиболее функциональных, даст экономический эффект и увеличит эффективность организаций, даже с учетом того, что инженер по требованиям ПО продолжит выполнять существенную часть активностей в ручном режиме.

Список литературы

1. **Косяков А., Свит У., Сеймур С., Бимер С.** Системная инженерия. Принципы и практика / Пер. с англ. под ред. В. К. Батоврина. М.: ДМК Пресс, 2014. 636 с.
2. **IEEE 610.12—1990** IEEE Standard Glossary of Software Engineering Terminology. URL: https://standards.ieee.org/standard/610_12-1990.html
3. **Халл Э., Джексон К., Джерими Д.** Разработка и управление требованиями: практическое руководство пользователя, 2-е изд.: Пер. с англ. 2005. 240 с.
4. **Леффингуэлл Д., Уидриг Д.** Принципы работы с требованиями к программному обеспечению. Унифицированный подход. Пер. с англ. М: Издательский дом "Вильямс", 2002. 448 с.
5. **Samer R., Atas M., Felfernig A., Stettinger M., Falkner A., Schenner G.** Group Decision Support for Requirements Management Processes // Proceedings of the 20th Configuration Workshop, 2018. P. 19—24.
6. **Motger Q., Borrull R., Palomares C., Marco J.** OpenReq-DD: A Requirements Dependency Detection Tool // Joint Proceedings of REFSQ-2019 Workshops, Doctoral Symposium, 2019. URL: http://ceur-ws.org/Vol-2376/NLP4RE19_paper01.pdf
7. **The Standish** Group Report — Chaos report // The Standish Group Report — Chaos report. URL: https://www.projectsmart.co.uk/white-papers/chaos-report.pdf
8. **Ian A.** Requirements Management with Doors: A Success Story // Impacts of RE Research on Practice. URL: http://www.scenarioplus.org.uk/papers/doors_story/doors_success_story.htm
9. **Requirements** Management Tools List — Project Performance International. URL: https://www.ppi-int.com/wp-content/uploads/2019/04/PPI-005107-8-Requirements-Management-Tools-190403-1.pdf

10. **Birk A., Heller G.** List of Requirements Management Tools // List of Requirements Management Tools. URL: https://makingofsoftware.com/resources/list-of-rm-tools/
11. **Ian A.** Requirements Tools // Ian F Alexander. 2014. URL: https://scenarioplus.org.uk/vendors.htm
12. **Кодубец А. А., Артемьева И. Л.** Обзор исследований в области инженерии требований для программных систем // Программная инженерия. 2021. Том 12, № 7. С. 339—349.
13. **Lieble G., Tichy M., Knause E., Ljungkrantz O., Stlegibauer G.** Organization and communication problems in automotive requirements engineering // Requirements Eng 23. 2018. P. 145—167.
14. **MBSE** Tools — Model Based Systems Engineering Wiki. URL: https://www.incosewiki.info/Model_Based_Systems_Engineering/index.php?title = MBSE_Tools
15. **DPM-10** Best Requirements Management Tools & Software of 2021. URL: https://thedigitalprojectmanager.com/requirements-management-tools/#overview
16. **Krotov S.** Top 10 Best Requirements Management (RM) Tools — 2021 | Cllax — Top of IT. URL: https://www.softwaretestinghelp.com/requirements-management-tools/
17. **Martin M.** 30 Best Requirements Management Tools in 2021. URL: https://www.guru99.com/requirement-management-tools.html
18. **Top 20+** Best Requirements Management Tools (The Complete List). URL: https://www.softwaretestinghelp.com/requirements-management-tools/
19. **The 10 Best** Requirements Traceability Tools. URL: https://www.inflectra.com/tools/requirements-management/10-best-requirements-traceability-tools
20. **Chappell D.** What is application lifecycle management? Chappell and Associates, Chicago, December, 2008. 6 p.
21. **Product** Development Lifecycle Solution | Jama Connect™. URL: https://www.jamasoftware.com/platform/jama-connect/
22. **Engineering** Requirements DOORS Family — Обзор — Российская Федерация | IBM. URL: https://www.ibm.com/ru-ru/products/requirements-management
23. **Requirements** Management Software | Dimensions RM | Micro Focus. URL: https://www.microfocus.com/en-us/products/dimensions-rm/overview
24. **Requirements** Management Tools — Modern Requirements. URL: https://www.modernrequirements.com/products/modern-requirements4devops/
25. **Visure** Requirements — Visure Solutions. URL: https://visuresolutions.com/requirements-management-tool/
26. **Accompa** — Requirements Management Software Tool. URL: https://web.accompa.com/
27. **ReqSuite®** RM — The Requirements Management Tool that Thinks Ahead. URL: https://www.osseno.com/en/requirements-management-tool/
28. **ReqView —** SW & HW Requirements Management Tool / Easy & Flexible. URL: https://www.reqview.com/
29. **RequirementONE** Home. URL: https://www.requirementone.com/index.html
30. **TraceCloud:** Project Requirements Management & Traceability. URL: https://www.tracecloud.com
31. **PEARLS** — The Smartest Requirement Management Tool for BA. URL: https://pearls-inc.com/
32. **ISO 26262-1:2018(en),** Road vehicles — Functional safety — Part 1: Vocabulary. URL: https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-2:v1:en
33. **Helix** ALM | ALM Tools | Perforce. URL: https://www.perforce.com/products/helix-alm
34. **Application** Lifecycle Management — ALM Tools, Software — Kovair. URL: https://www.kovair.com/alm-studio/
35. **Application** Lifecycle Management Software | codeBeamer ALM. URL: https://intland.com/codebeamer/application-lifecycle-management/
36. **Requirements** Management Software | Requirements Management Solution. URL: https://techexcel.com/products/devspec/
37. **Home |** Valispace. URL: https://www.valispace.com/
38. **Orcanos** Application Lifecycle Management Tool & Software — ALM Software Tool — Orcanos Software — ALM And Quality Management. URL: https://www.orcanos.com/compliance/
39. **Cockpit** Enterprise | For Medical Device | Cognition Corporation. URL: https://cognition.us/solutions/cockpit-enterprise/
40. **Aligned** AG — Aligned Elements — the Medical Device Application Lifecycle Management (ALM) software for Design History File management Corporation. URL: https://www.aligned.ch/
41. **Medical** Device Design and Documentation Product | Matrix Requirements. URL: https://matrixreq.com/
42. **ALM** Octane — Agile Testing, Release Management & Value Stream Insights | Micro Focus. URL: https://www.microfocus.com/en-us/products/alm-octane/overview
43. **Aqua** cloud. URL: https://aqua-cloud.io/
44. **ENOVIA** 3DEXPERIENCE — Dassault Systmes®. URL: https://www.3ds.com/ru/produkty-i-uslugi/enovia/produkty/3dexperience/
45. **Application** Lifecycle Management — ALM Tools by Inflectra. URL: https://www.inflectra.com/SpiraTeam/
46. **Requirements** Management, Requirements Gathering, Requirements Management tools — Polarion REQUIREMENTS. URL: https://polarion.plm.automation.siemens.com/products/polarion-requirements
47. **Easiest** Requirements Management Software for IT Teams | ReQtest. URL: https://reqtest.com/features/requirement-management/
48. **Rommana** Requirement Management Tools — User Story Tools. URL: https://rommanasoftware.com/requirement-tools.php
49. **Software** — EnCo Software GmbH. URL: https://www.enco-software.com/en/software/
50. **Requirements** Management Software — Xebrio. URL: https://www.xebrio.com/requirements-management-software
51. **Requirements** Management and Validation | PTC. URL: https://www.ptc.com/en/technologies/plm/requirements-management
52. **ISO 14971:2019** — Medical devices — Application of risk management to medical devices. URL: https://www.iso.org/standard/72704.html
53. **ISO 13485** — Medical devices. URL: https://www.iso.org/iso-13485-medical-devices.html
54. **Alebrahim A.** Bridging the Gap between Requirements Engineering and Software Architecture A Problem-Oriented and Quality-Driven Method. Springer Vieweg, 2016. 514 p.
55. **IBM** Rational Software Architect Designer — Overview | IBM ALM. URL: https://www.ibm.com/products/rational-software-architect-designer?mhsrc=ibmsearch_a&mhq=IBM%20Rational%20Architect
56. **25 BEST** UML Tools | FREE UML Diagram Software in 2021. URL: https://www.guru99.com/best-uml-tools.html
57. **Cradle** requirements management, alm, mbse, system engineering software, 3SL. URL: https://www.threesl.com/
58. **objectiF** RM: Requirements Engineering Software. URL: https://www.microtool.de/en/products/objectif-rm/
59. **Full** Lifecycle Modeling for Business, Software and Systems | Sparx Systems. URL: https://sparxsystems.com/products/ea/index.html
60. **Engineering** Systems Design Rhapsody — Overview | IBM. URL: https://www.ibm.com/products/systems-design-rhapsody
61. **Cameo** Systems Modeler — CATIA — Dassault Systmes®. URL: https://www.3ds.com/products-services/catia/products/no-magic/cameo-systems-modeler/
62. **PREEvision** | The E/E Engineering Solution | Vector. URL: https://www.vector.com/int/en/products/products-a-z/software/preevision/
63. **MacA&D** and WinA&D — Requirements Management with Definition Templates, Views & Queries, Reports and Traceability. URL: https://www.excelsoftware.com/requirementsmanagement
64. **QVscribe** by QRA Corp — Analyze Requirements Documents in Seconds. URL: https://qracorp.com/qvscribe/
65. **RAVEN** for Microsoft Office | Ravenflow. URL: http://www.ravenflow.com/raven-for-microsoft-office
66. **VT Docs —** Document Analysis Software for Proposal and Contract Teams. URL: https://www.visiblethread.com/vt-docs/
67. **RQA —** QUALITY Studio. URL: https://www.reusecompany.com/rqa-quality-studio
68. **ReqSuite®** QC — Automated Quality Control for Requirements. URL: https://www.osseno.com/en/quality-control/
69. **Engineering** Requirements Quality Assistant — Overview | IBM. URL: https://www.ibm.com/products/requirements-quality-assistant
70. **Guide** for Writing Requirements // Guide for Writing Requirements, INCOSE-TP-2010-006-02.1. URL: https://connect.incose.org/Pages/Product-Details.aspx?ProductCode=TechGuideWR2019Soft
71. **Mavin A., Wilkinson P., Harwood A., Novak M.** Easy approach to requirements syntax (EARS) // Requirements Engineering Conference. 2009. P. 317—322.

72. **Samer R., Atas M., Felfernig A., Stettinger M., Falkner A., Schenner G.** Group Decision Support for Requirements Management Processes // Proceedings of the 20th Configuration Workshop, Graz, Austria, September 27th to 28th, 2018. P. 19—24.
73. **25 Best** FREE Wireframe Tools & Software in 2021 IBM. URL: https://www.guru99.com/best-wireframe-tools.html
74. **Irise** — Best prototyping & requirements platform for remote teams. URL: https://www.irise.com/
75. **Free** prototyping tool for web & mobile apps — Justinmind. URL: https://www.justinmind.com/
76. **Devprom** ALM — автоматизация процессов разработки ПО. URL: https://devprom.ru/
77. **Система** Управления Требованиями (СУТр) — Техэксперт. URL: https://техэксперт.рф/sutr
78. **Requality**. URL: http://www.requality.ru/ru/index.html
79. **ЛОЦМАН:**PLM. URL: https://ascon.ru/products/889/review/
80. **Платформа** "Союз-PLM" | plmsoyuz. URL: https://www.плмсоюз.рф/platform
81. **T-FLEX** RM | Управление требованиями. URL: https://www.tflex.ru/products/docs/rm/
82. **Ютт В. Е., Морозов В. В., Чепланов В. И.** Аппараты систем управления зажиганием и впрыском топлива: учеб. пособие. М.: МАДИ, 2013. 112 с.
83. **Santana S., Perero L., Delduca A**. Evaluation of Open Source Tools for Requirements Management // 25th Argentine Congress of Computer Science, CACIC 2019, pp. 188—204.
84. **Open** Source Requirements Management Tool download | SourceForge.net. URL: https://sourceforge.net/projects/osrmt/
85. **aNimble** Platform download | SourceForge.net net. URL: https://sourceforge.net/projects/nimble/
86. **Requirement** Heap download | SourceForge.net. URL: https://sourceforge.net/projects/reqheap/
87. **Requirements** Engineering — tools and solutions offered by OpenReq. URL: https://openreq.eu/
88. **Modelio** Open Source — UML and BPMN free modeling tool. URL: https://www.modelio.org/
89. **Eclipse** Requirements Modeling Framework. URL: https://www.eclipse.org/rmf/
90. **reqT —** a scalable requirements tool. URL: http://www.reqt.org/
91. **FRET:** Formal Requirements Elicitation Tool(ARC-18066-1) / NASA Software Catalog. URL: https://software.nasa.gov/software/ARC-18066-1
92. jUCMNau — WebHome < ProjetSEG < Foswiki — jUCMNav: Juice up your modelling! URL: http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome
93. **Aha!** — The World's #1 Roadmap Software, 2021. URL: https://www.aha.io/
94. **Process** Automation Design Platform to Scale RPA | Blueprint. URL: https://www.blueprintsys.com/
95. **R4J —** ease solutions. URL: https://www.easesolutions.com/r4j/
96. **Reqchecker —** Democratizes requirement engineering. URL: https://reqchecker.eu/
97. **Abou**t the Requirements Interchange Format Specification Version 1.2. URL: https://www.omg.org/spec/ReqIF/
98. **Products** ReqEdit.URL: https://www.reqteam.com
99. **Traceability** | Reqtify — Dassault Systmes®. URL: https://www.3ds.com/products-services/catia/products/reqtify/
100. **ReqMan**® — requests for quotes in record time. URL: https://www.em.ag/en/reqman/
101. **Aha!** Software — Idea Managment Software System. URL: https://www.aha.io/product/ideas. URL: https://www.aha.io/product/ideas

Requirements Management Tools for Software Systems: a Systematic Tools Review

A. A. Kodubets, e-mail: alexey@kodubets.ru, Far Eastern Federal University, Vladivostok, 690922, Russian Federation

Corresponding author:

Kodubets Alexey A., Postgraduate Student, Far Eastern Federal University, Vladivostok, 690922, Russian Federation
E-mail: alexey@kodubets.ru

Received on October 25, 2021
Accepted on November 18, 2021

This article contains a systematic review of Requirements Management tools (RM tools) for Software Systems. A research question was defined as the following: requirements development process of large-scale software system (with thousands of requirements) and an interaction problem during this process (communication, coordination and control). The problem is caused by the fact that the requirements development process is a cross-disciplinary task and it involves multiple parties — stakeholders, domain experts, and suppliers with own goals and constraints, and thus, the interaction between them seriously slows down the overall requirements development process more than writing the requirements specification itself. The tools were classified into the following classes: Requirements Management Tools, ALM/PLM/SDLC Tools, Model-based System Engineering tools with Requirements support, Natural Language based tools for Requirements Engineering, WYSIWYG-prototyping and Requirements platform, Open Source tools, and tools developed by Russian vendors. Each tool class was described and represented with list of relevant tools — including feature-rich solutions, and a description of key features. A contribution of functionality from each tool into the research question was analyzed and summarized including potential further improvement steps. Advantages and disadvantages of the existing RM tools were represented in this work. To approach the research question, further potential directions were described.

Keywords: *requirements engineering, requirements management tools, large-scale software systems*

For citation:

Kodubets A. A. Requirements Management Tools for Software Systems: A Systematic Tools Review, *Programmnaya Ingeneria*, 2022, vol. 13, no. 1, pp. 17—31.

DOI: 10.17587/prin.13.17-31

References

1. **Kossiakoff A., Sweet W. N., Seymour S. J, Bierner S. M.** *System Engineering Princeples and Practie 2nd Edition*, Wiley-Interscience, 2011, 531 p.
2. **IEEE** 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology, available at: https://standards.ieee.org/standard/610_12-1990.html
3. **Hull E., Jackson K., Dick J.** *Requirements Engineering*, Second Edition, Springer, 2005, 198 p.
4. **Leffingwell D. Widrig D.** *Managing Software requirements: a use case approach*, Addison-Wesley, Boston, 2003, pp. 17.
5. **Samer R., Atas M., Felfernig A., Stettinger M., Falkner A., Schenner G.** Group Decision Support for Requirements Management Processes, *Proceedings of the 20th Configuration Workshop*, 2018, pp. 19—24.
6. **Motger Q., Borrull R., Palomares C., Marco J.** OpenReqDD: A Requirements Dependency Detection Tool, *Joint Proceedings of REFSQ-2019 Workshops, Doctoral Symposium,* 2019. URL: http://ceur-ws.org/Vol-2376/NLP4RE19_paper01.pdf.
7. **The Standish** Group, Chaos report 2014. available at: https://www.projectsmart.co.uk/white-papers/chaos-report.pdf
8. **Ian A.** Requirements Management with Doors: A Success Story, *Impacts of RE Research on Practice*, available at: http://www.scenarioplus.org.uk/papers/doors_story/doors_success_story.htm
9. **Requirements** Management Tools List — Project Performance International, available at: https://www.ppi-int.com/wp-content/uploads/2019/04/PPI-005107-8-Requirements-Management-Tools-190403-1.pdf
10. **Birk A., Heller G.** List of Requirements Management Tools, available at: https://makingofsoftware.com/resources/list-of-rm-tools/
11. **Ian A.** Requirements Tools, available at: https://scenarioplus.org.uk/vendors.htm
12. **Kodubets A. A., Artemieva I. L.,** Requirements Engineering for Software Systems: A Systematic Literature Review, *Programmnya Ingeneria*, 2021, vol. 12, no. 7, pp. 339—349.
13. **Lieble G., Tichy M., Knause E., Ljungkrantz O., Stlegibauer G.** Organization and communication problems in automotive requirements engineering, *Requirements Eng* 23, 2018, P. 145—167.
14. **MBSE** Tools — Model Based Systems Engineering Wiki, available at: https://www.incosewiki.info/Model_Based_Systems_Engineering/index.php?title=MBSE_Tools
15. **DPM** — 10 Best Requirements Management Tools & Software of 2021, available at: https://thedigitalprojectmanager.com/requirements-management-tools/#overview
16. **Krotov S.** Top 10 Best Requirements Management (RM) Tools — 2021 | Cllax — Top of IT, available at: https://www.softwaretestinghelp.com/requirements-management-tools/
17. **Martin M.** 30 Best Requirements Management Tools in 2021, available at: https://www.guru99.com/requirement-management-tools.html
18. **Top 20+** Best Requirements Management Tools (The Complete List), available at: https://www.softwaretestinghelp.com/requirements-management-tools/
19. **The 10** Best Requirements Traceability Tools**,** available at: https://www.inflectra.com/tools/requirements-management/10-best-requirements-traceability-tools
20. **Chappell D.** *What is application lifecycle management?,* Chappell and Associates, 2008, 6 p.
21. **Product** Development Lifecycle Solution | Jama Connect, available at: https://www.jamasoftware.com/platform/jama-connect/
22. **Engineering** Requirements DOORS Family — Overview — United Kingdom | IBM available at: https://www.ibm.com/uk-en/products/requirements-management
23. **Dimensions** RM — Requirements Management Software Micro Focus, available at: https://www.microfocus.com/en-us/products/dimensions-rm/overview
24. **Modern** Requirements — Requirements Management Tools, available at: https://www.modernrequirements.com/products/modern-requirements4devops/
25. **Visure** Requirements — Visure Solutions, available at: https://visuresolutions.com/requirements-management-tool/
26. **Accompa** — Requirements Management Software Tool, available at: https://web.accompa.com/
27. **ReqSuite®** RM — The Requirements Management Tool that Thinks Ahead, available at: https://www.osseno.com/en/requirements-management-tool/
28. **ReqView** — SW & HW Requirements Management Tool | Easy & Flexible, available at: https://www.reqview.com/
29. **RequirementONE** Home, available at: https://www.requirementone.com/index.html
30. **TraceCloud**: Project Requirements Management & Traceability, available at: ttps://www.tracecloud.com
31. **PEARLS** — The Smartest Requirement Management Tool for BA, available at: https://pearls-inc.com/
32. **ISO 26262-1:2018(en)**, Road vehicles — Functional safety — Part 1: Vocabulary, available at: https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-2:v1:en
33. **Helix** ALM | Perforce, available at: https://www.perforce.com/products/helix-alm
34. **Kovair** ALM Tools — Application Lifecycle Management, Software, available at: https://www.kovair.com/alm-studio/
35. **codeBeamer** ALM — Application Lifecycle Management Software, available at: https://intland.com/codebeamer/application-lifecycle-management/
36. **Devspec** — Requirements Management Software | Requirements Management Solution, available at: https://techexcel.com/products/devspec/
37. **Valispace,** available at: https://www.valispace.com/
38. **Orcanos** Application Lifecycle Management Tool & Software — ALM Software Tool — Orcanos Software — ALM And Quality Management, available at: https://www.orcanos.com/compliance/
39. **Cockpit** Enterprise | For Medical Device | Cognition Corporation, available at:: https://cognition.us/solutions/cockpit-enterprise/
40. **Aligned** AG — Aligned Elements — the Medical Device Application Lifecycle Management (ALM) software for Design History File management Corporation, available at: https://www.aligned.ch/
41. Matrix Requirements — Medical Device Design and Documentation Product, available at: https://matrixreq.com/
42. **ALM** Octane — Agile Testing, Release Management & Value Stream Insights | Micro Focus, available at: https://www.microfocus.com/en-us/products/alm-octane/overview
43. **Aqua** cloud, available at: https://aqua-cloud.io/
44. **ENOVIA** 3DEXPERIENCE — Dassault Systmes®, available at: https://www.3ds.com/ru/produkty-i-uslugi/enovia/produkty/3dexperience/
45. **Application** Lifecycle Management — ALM Tools by Inflectra, available at: https://www.inflectra.com/SpiraTeam/
46. **Polarion** — Requirements Management, Requirements Gathering, Requirements Management tools, available at: https://polarion.plm.automation.siemens.com/products/polarion-requirements
47. **ReQtest** — Easiest Requirements Management Software for IT Teams, available at: https://reqtest.com/features/requirement-management/
48. **Rommana** Requirement Management Tools — User Story Tools, available at: https://rommanasoftware.com/requirement-tools.php
49. **Sox** — Software — EnCo Software GmbH, available at: https://www.enco-software.com/en/software/
50. **Xebrio** — Requirements Management Software, available at: https://www.xebrio.com/requirements-management-software
51. **Requirements** Management and Validation | PTC, available at: https://www.ptc.com/en/technologies/plm/requirements-management
52. **ISO** — ISO 14971:2019 — Medical devices — Application of risk management to medical devices, available at: https://www.iso.org/standard/72704.html
53. **ISO** — ISO 13485 — Medical devices, available at: https://www.iso.org/iso-13485-medical-devices.html
54. **Alebrahim A.** *Bridging the Gap between Requirements Engineering and Software Architecture A Problem-Oriented and Quality-Driven Method,* Springer Vieweg, 2016, 514 p.
55. **IBM** Rational Software Architect Designer — Overview, available at: https://www.ibm.com/products/rational-software-architect-designer?mhsrc=ibmsearch_a&mhq=IBM%20Rational%20Architect
56. **25 BEST** UML Tools | FREE UML Diagram Software in 2021, available at: https://www.guru99.com/best-uml-tools.html
57. **Cradle** requirements management, alm, mbse, system engineering software, 3SL, available at: https://www.threesl.com/
58. **objectiF** RM: Requirements Engineering Software, available at: https://www.microtool.de/en/products/objectif-rm/
59. **Enterprise** Architect — Full Lifecycle Modeling for Business, Software and Systems | Sparx Systems, available at: https://sparxsystems.com/products/ea/index.html

60. **Rhapsody** — Engineering Systems Design Rhapsody — Overview | IBM, available at: https://www.ibm.com/products/systems-design-rhapsody
61. **Cameo** Systems Modeler — CATIA — Dassault Systemes®, available at: https://www.3ds.com/products-services/catia/products/no-magic/cameo-systems-modeler/
62. **PREEvision** | The E/E Engineering Solution | Vector, available at: https://www.vector.com/int/en/products/products-a-z/software/preevision/
63. **MacA&D** and WinA&D — Requirements Management with Definition Templates, Views & Queries, Reports and Traceability, available at: https://www.excelsoftware.com/requirementsmanagement
64. **QVscribe** by QRA Corp — Analyze Requirements Documents in Seconds, available at: https://qracorp.com/qvscribe/
65. **RAVEN** for Microsoft Office | Ravenflow, available at: http://www.ravenflow.com/raven-for-microsoft-office
66. **VT Docs —** Document Analysis Software for Proposal and Contract Teams, available at: https://www.visiblethread.com/vt-docs/
67. **RQA** — QUALITY Studio, available at: https://www.reusecompany.com/rqa-quality-studio
68. **ReqSuite®** QC — Automated Quality Control for Requirements, available at: https://www.osseno.com/en/quality-control/
69. **RQA** — Engineering Requirements Quality Assistant — Overview | IBM, available at: https://www.ibm.com/products/requirements-quality-assistant
70. **Ryan M., Wheatcraft L., Zinni R., Dick J., Baksa K.** Guide for Writing Requirements, *INCOSE Publications Office,* 2019.
71. **Mavin A., Wilkinson P., Harwood A., Novak M.** Easy approach to requirements syntax (EARS), *Requirements Engineering Conference,* 2009, pp. 317—322.
72. **Samer R., Atas M., Felfernig A., Stettinger M., Falkner A., Schenner G.** Group Decision Support for Requirements Management Processes, *Proceedings of the 20th Configuration Workshop*, 2018, pp. 19—24.
73. **25 Best** FREE Wireframe Tools & Software in 2021 IBM, available at: https://www.guru99.com/best-wireframe-tools.html
74. **Irise —** Best prototyping & requirements platform for remote teams, available at: https://www.irise.com/
75. **Justinmind** — Free prototyping tool for web & mobile apps, available at: https://www.justinmind.com/
76. **Devprom** ALM — Software Development Process Automation, available at: https://devprom.ru/
77. **Requirements** Management Tools (SUTr) — TechExpert, available at: https://техэксперт.рф/sutr
78. **Requality**, Ivannikov Institute for System Programming of the RAS (ISP RAS), available at: http://www.requality.ru/ru/index.html
79. **LOCMAN**: PLM, available at: https://ascon.ru/products/889/review/
80. **Souz-PLM** platform | plmsoyuz, available at: https://www.плмсоюз.рф/platform
81 **T-FLEX** RM | Requirements Management, available at: https://www.tflex.ru/products/docs/rm/
82. **Utt V., Morozov V., Cheplanov V.** *Devices of ignition and fuel injection control systems*, Moscow, Madi, 2013, 112 p. (in Russian).
83. **Santana S., Perero L., Delduca A**. Evaluation of Open Source Tools for Requirements Management, *25th Argentine Congress of Computer Science, CACIC 2019*, pp. 188—204.
84. **OSRMT** — Open Source Requirements Management Tool download | SourceForge.net, available at: https://sourceforge.net/projects/osrmt/
85. **aNimble** Platform, available at: https://sourceforge.net/projects/nimble/
86. **hedHeap** — Requirement Heap, available at: https://sourceforge.net/projects/reqheap/
87. **OpenReq R** — Requirements Engineering — tools and solutions offered by OpenReq, available at: https://openreq.eu/
88. **Modelio** Open Source — UML and BPMN free modeling tool, available at: https://www.modelio.org/
89. **RMF** — Eclipse Requirements Modeling Framework, available at: https://www.eclipse.org/rmf/
90. **reqT —** a scalable requirements tool, available at: http://www.reqt.org/
91. **FRET**: Formal Requirements Elicitation Tool(ARC-18066-1) | NASA Software Catalog, available at: https://software.nasa.gov/software/ARC-18066-1
92. **WebHome** < ProjetSEG < Foswiki — jUCMNav: Juice up your modelling!, available at: http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome
93. **Aha!** — The World's #1 Roadmap Software, available at: https://www.aha.io/
94. **Blueprint** — Process Automation Design Platform to Scale RPA, available at: https://www.blueprintsys.com/
95. **R4J** — ease solutions, available at: https://www.easesolutions.com/r4j/
96. **Reqchecker** — Democratizes requirement engineering, available at: https://reqchecker.eu/
97. **ReqIF** — the Requirements Interchange Format Specification Version 1.2, available at: https://www.omg.org/spec/ReqIF/
98. **ReqEdit**, available at: https://www.reqteam.com/en
99. **Traceability** | Reqtify — Dassault Systmes®, available at: https://www.3ds.com/products-services/catia/products/reqtify/
100. **ReqMan®** — requests for quotes in record time, available at: https://www.em.ag/en/reqman/
101. **Aha!** Software — Idea Managment Software System, available at: https://www.aha.io/product/ideas

ИНФОРМАЦИЯ

УДК 004.65 DOI: 10.17587/prin.13.32-40

Е. А. Курако, канд. техн. наук, ст. науч. сотр., kea@ipu.ru,
В. Л. Орлов, канд. техн. наук, вед. науч. сотр., ovl@ipu.ru,
Институт проблем управления им. В. А. Трапезникова РАН, Москва

К вопросу миграции баз данных из среды Oracle в среду PostgreSQL

Рассмотрены методы переноса данных из среды Oracle в среду PostgreSQL с учетом поэтапной организации действий и определения последовательности миграции. Выделены основные направления для преобразования структуры баз, форматов данных и выполняемых объектов. Определены методы преобразования хранимых процедур и триггеров, написанных на процедурных языках, в качестве которых используются PL/SQL и PL/pgSQL. Рассмотрены вопросы изменения программного обеспечения на узлах, непосредственно связанных (сопряженных) с сервером базы данных и вопросы минимизации остановок в процессе миграции, что дает возможность проводить перенос для географически распределенных и круглосуточно функционирующих объектов. Приведены примеры изменения скорости выполнения запросов до и после миграции для систем различных типов.

Ключевые слова: *база данных, БД, СУБД, Oracle, PostgreSQL, миграция, структура, данные, функции, хранимые процедуры, PL/SQL, PL/pgSQL*

Введение

Под миграцией баз данных (БД) подразумевается перевод той или иной БД, работающей под управлением определенной системы управления базами данных (СУБД), на функционирование под управлением другой СУБД. Отметим, что СУБД взаимодействует с другими компьютерами информационной системы, образующими сложную сеть. Изменение узла с БД при этом потребует изменения узлов, связанных с ним.

Если раньше необходимость смены СУБД была практически не востребована, то сейчас миграция в ряде случаев становится просто необходимой. Эта необходимость в основном вызывается перечисленными далее обстоятельствами.

- Информация в базах данных накапливается в течение нескольких лет, а иногда и десятилетий. Казалось бы, при столь долгой эксплуатации программное обеспечение (ПО), обслуживающее хранимые и корректируемые данные, не должно изменяться. Однако ценность накапливаемых данных растет, а к процессу управления этими данными возникают дополнительные требования. Иногда эти требования приводят к необходимости замены СУБД.
- Часто возникают не чисто технические проблемы, а обстоятельства, связанные с обеспечением требований безопасности. Это происходит в тех случаях, когда применяемая СУБД — закрытая, доступ к ее исходным кодам практически отсутствует и возникает подозрение о возможности использования поставщиком закладок и других недокументированных средств, позволяющих обойти декларированные средства защиты. В этом случае также требуется замена СУБД при сохранении содержимого БД.
- Существует также вопрос эксплуатационной стоимости, которая обычно включает не только стоимость лицензии, но и затраты на техническое сопровождение. Здесь преимущество получают открытые системы, которые распространяются бесплатно, а стоимость технического сопровождения существенно ниже. В этом случае появляется перспектива существенной экономии средств.

Вместе с тем перенос системы под управление другой СУБД в сложных сетях является далеко не тривиальной задачей. Это определяется тем обстоятельством, что СУБД имеют описанные далее особенности.

- Каждая СУБД имеет свою структуру данных. Нужно иметь в виду, что, хоть эти структуры и близки, но их описание различается, а значит скрипты, по которым формируется база данных, будут различными.
- Перенос и преобразование данных проводятся в соответствии с учетом согласования структур.
- В развитых БД неотъемлемой частью является также использование хранимых процедур (функций), которые содержат написанный на определенном языке исходный код, выполняемый при соответствующих вызовах. Следует иметь в виду, что в этом случае возможна разница не только в синтаксисе, но и в семантике, так как здесь обрабатываются различные структуры различными методами.
- Кроме того, в различных СУБД имеются различные дополнительные параметры, обработка которых при перенесении данных должна проводиться с учетом особенностей этих параметров.

Основные способы миграции баз данных из среды СУБД Oracle в среду СУБД PostgreSQL

Рассмотрим способы миграции БД из среды СУБД Oracle в среду СУБД PostgreSQL. Это преобразование вызывает особый интерес, так как СУБД этих типов могут работать с большими массивами данных, что важно для современных систем. Кроме того, одна из СУБД — коммерческая (Oracle), другая — открытая (PostgreSQL), что предпочтительнее с точки зрения анализа безопасности, так как в последнем случае всегда доступны исходные тексты. Следует отметить, что и та и другая СУБД включают в свой состав языки для написания хранимых процедур PL/SQL и PL/pgSQL соответственно, что существенно для построения эффективных структур работы с БД.

Заметим, что процедуру миграции в автоматическом режиме полностью провести невозможно в силу сложности задачи и необходимости учета смысловых особенностей алгоритма при переносе хранимых процедур. Вместе с тем существует довольно большое число коммерческих и некоммерческих инструментальных средств [1], которые позволяют облегчить этот процесс (табл. 1).

Как известно, БД непосредственно включает следующие составляющие: описание структуры, данные, выполняемые объекты (рис. 1).

Все приведенные средства обеспечивают, по крайней мере, преобразование и перенос одной составляющей — структуры БД. Большинство проводит анализ структуры, принадлежащей БД Oracle, готовит на основе этого анализа семейство скриптов, позволяющее создать новую структуру, но уже формата PostgreSQL. Таким образом, результатом является новая структура, созданная с учетом особенностей PostgreSQL, включающая таблицы, представления, ключи, связи и другие объекты. Вместе с тем созданная к этому времени база является пустой.

Создание новой структуры возможно, например, с использованием таких инструментальных срдеств, как Enterprise DB Migration Toolkit, Full Convert, Ora2Pg. Кроме того, для компактного преобразования структуры в Институте проблем управления Российской академии наук (ИПУ РАН) разработана отечественная программа "Транслятор структур Oracle—PostgreSQL (OracleToPostgres_Scheme)" [2]. Особенность этого ПО заключается в том, что в качестве исходного материала берется скрипт, описывающий создание базы данных в Oracle. Этот скрипт или формируется изначально при проектировании БД и существует к моменту преобразования, например, созданный таким средством, как ErWin. Или такой скрипт может быть восстановлен из реальной БД с использованием сторонних инструментов, таких как, например, dbForge for Oracle. Результатом работы программы OracleToPostgres_Scheme является также скрипт для создания структуры БД, но уже представленный в формате PostgreSQL.

Отметим, что помимо множества необходимых преобразований структуры (описаний таблиц, связей, представлений), основным является преобразование типов данных, некоторые из которых приведены в табл. 2.

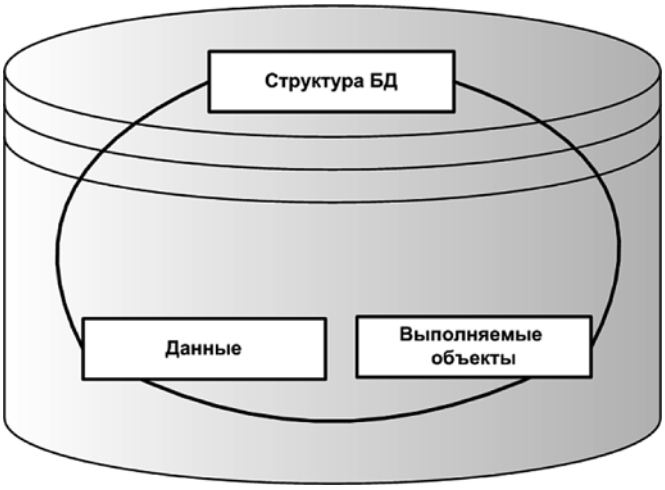

Рис. 1. Основные составляющие БД

Таблица 1

Основные инструментальные средства для миграции БД

Инструментальное средство	Коммерческое/ некоммерческое	Основные функции	Недостатки
Enterprise DB Migration Toolkit	Коммерческое	Переносит структуру и данные	Не переносит хранимые процедуры
Oracle Golden Gate	Коммерческое	Обеспечивает репликацию БД, осуществляя перемещение в базы другого вида, в том числе и PostgreSQL	Высокая стоимость. Работает только с данными
SymmetricDS	Некоммерческое	Обеспечивает репликацию БД, перенося данные, в частном случае, в PostgreSQL	Работает только с данными. Есть ограничения при использовании полей типа BLOB
Full Convert	Коммерческое	Переносит структуру и данные. Работает с различными СУБД.	Минимальная настройка
Ora2Pg	Некоммерческое	Переносит структуру, данные. Проект активно развивается, надежен	Усложненная настройка

Таблица 2

Основные типы данных для перевода

Основные типы данных Oracle	Соответствующие типы данных PostgreSQL	Примечание
BLOB	BYTEA	⩽ 4 ГБ
CHAR	CHAR	$1 \leqslant n \leqslant 2000$
CLOB	TEXT	⩽ 4 ГБ
DATE	TIMESTAMP	Дата и время
TIMESTAMP	TIMESTAMP	—
DECIMAL	DECIMAL	—
FLOAT	DOUBLE PRECISION	—
LONG	TEXT	⩽ 2 ГБ
NUMBER	INTEGER	—
VARCHAR	VARCHAR	$1 \leqslant n \leqslant 4000$
VARCHAR2	VARCHAR	$1 \leqslant n \leqslant 4000$
XMLTYPE	XML	XML data

Вторая стадия миграции обеспечивает перенос реальных данных из базы данных Oracle в БД PostgreSQL. Для этого может использоваться Ora2Pg или специально для этого разработанная отечественная программа "Трансфер данных Oracle—PostgreSQL (OracleToPostgres_Data)" [3].

Для ускорения переноса данных программа OracleToPostgres_Data временно устраняет все связи в структуре БД, созданной на первой стадии миграции. После этого проводится перенос потаблично всех данных. При этом данные читаются из таблиц Oracle и записываются в таблицы PostgreSQL. Когда перемещение данных завершено, то все связи восстанавливаются.

На этом этапе также важно провести анализ последовательностей, использующихся в Oracle, и в процессе переноса завести соответствующие последовательности в PostgreSQL, установив их текущие значения на момент миграции из первичной базы.

На втором этапе миграция баз данных могла бы быть завершена. Действительно — создана новая база данных, созданы таблицы, индексы, последовательности, установлены все необходимые связи, перенесены все данные. Однако существуют по крайней мере несколько существенных причин, по которым работу нельзя признать законченной. Это определяется тем обстоятельством, что в современных БД хранятся не только статические данные, но и выполняемые объекты, по существу представляющие собой программы для обработки данных. Например, к ним относятся хранимые процедуры, триггеры, типы, представления и материализованные представления.

Перенос выполняемых объектов

Хранимые процедуры и триггеры обычно пишутся на процедурном языке, в качестве которого в Oracle используется PL/SQL, а в PostgreSQL — язык PL/pgSQL [4]. Эти языки весьма близки, в то же время они имеют существенные отличия, которые исключают простой формальный перевод программ. В большинстве случаев тело функции содержит выражения на языке SQL, которые также могут иметь разный синтаксис в различных СУБД.

Прежде всего, нужно иметь в виду, что процедуры Oracle компонуются в пакеты (*Packages*). Эта возможность отсутствует в PostgreSQL, что вызывает и будет еще вызывать определенные споры среди разработчиков, однако факт остается фактом. В процессе переноса все функции и процедуры Oracle, входящие в определенный пакет, принято объединять в схему PostgreSQL с названием, идентичным названию исходного пакета. Но вместе с пакетами исчезают и пакетные переменные, а это изменяет логику программы.

Отметим, что в PostgreSQL применяются, как правило, функции. Если требуется перевести какую-либо процедуру из Oracle, то она тоже оформляется как функция. При этом все функции вызываются из других функций или внешних программ. Триггеры выполняются как реакция на то или иное событие.

Начинать перенос целесообразно с поиска используемых ключевых слов. Списки слов есть на официальных сайтах, рассматриваемых СУБД в разделе "Документация". Например, в СУБД PostgreSQL в качестве ключевого слова используется `current _ date`, которое заменяет ключевое слово `sysdate`, используемое в Oracle. В Oracle есть функция с одноименным названием, но СУБД позволяет использовать `current _ date` в качестве столбца. И будет корректен такой SQL-запрос:

```
insert into voc _ table (first _ f, current _ date)
values (1, sysdate);
```

В PostgreSQL это выражение вызовет ошибку. В то же время, использование ключевого слова `sysdate` легко обнаруживается анализатором PostgreSQL. Как уже отмечалось выше, языки PL/SQL и PL/pgSQL очень похожи. Рассмотрим фрагмент заголовка типовой функции для Oracle:

```
function s _ main.get _ doc _ list _ count(p _ num
in integer,
            p _ type  in  varchar2)  return
            integer is
```

И заголовок аналогичной функции для PostgreSQL:

```
create or replace function s _ main.get _ doc _
list _ count(p _ num in integer,
          p _ type in varchar) returns integer
language plpgsql
as $function$
```

Как можно видеть, из первого заголовка достаточно легко получить второй. Сложности начинаются, когда требуется возвратить несколько параметров. Пример для Oracle:

```
function get_filename(p_num in integer,
                      p_type in varchar2,
                      p_path out varchar2,
                      p_filename out varchar2)
                      return integer is
```

Который будет преобразован в:

```
create or replace function get_filename(p_
num in integer,
                    p_type in varchar,
                    p_path out varchar,
                    p_file out varchar,
                    p_return_code  out
                    integer) returns record
language plpgsql
as $function$
```

Потребовалось добавить один возвращаемый параметр типа `integer` и результат функции преобразовать в составной тип `record`, так как возвращаются разные типы в параметрах. В предыдущем примере возвращаемый тип был один.

Основную сложность вызывает использование в заголовках в качестве параметров специальных или пользовательских типов. Программам перевода потребуется помощь программиста, например, в определении уровня доступа пользовательского типа. Так в Oracle есть различные уровни доступа к пользовательскому типу, в числе которых доступ на уровне пакета функций (package). Значит, два пакета могут иметь одно и то же название типа с различным описанием. При автоматическом переводе потребуется уточнить, что надо сделать с этими типами: объединить, использовать только первый, переименовать второй и т. д.

При изменении исходного кода даже в простых вещах возможны ошибки или семантические неопределенности при конвертации. Рассмотрим исходный код для Oracle:

```
if (p_value is null) then ...
```

В этом выражении идет проверка на пустое значение. Если переменная `p_value` равна `null` или `p_value` равна `''`, то в Oracle условие выполнится. В PostgreSQL данное условие выполнится, если только `p_value` равна `null`. Таким образом, эквивалентное выражение в PostgreSQL:

```
if ((p_value is null) or (p_value = '')) then ...
```

Однако следует учитывать логику программы, в которой может потребоваться вариант проверки переменой на строгое равенство значению `null`, т. е. необходимо вмешательство программиста.

Рассмотрим второй пример с объединением строк:

```
'value='||p_value
```

В Oracle, если переменная `p_value` равна `null`, то результат будет `'value='`. В PostgreSQL ситуация другая, результат будет `null`. Для получения одинакового результата подходит выражение

```
concat('value=', p_value)
```

Здесь следует обратить внимание программиста на данное выражение для уточнения логики выполнения.

Работа с исключениями отличается мало и хорошо автоматически переводится. Пример для Oracle:

```
begin
  INSERT INTO voc_table VALUES (1, 'red');
exception
when others then
  raise_application_error(-20001,'Не удалось записать в БД '|| sqlerrm)
end;
```

Для PostgreSQL:

```
begin
  INSERT INTO voc_table VALUES (1, 'red');
exception
  when others then
    raise exception 'Не удалось записать в БД: %',sqlerrm;
  end;
```

Как видно, в данных примерах отличается лишь вызов исключения. Однако в случае использования пользовательских исключений опять придется использовать труд программиста, так как в PostgreSQL нет пользовательских исключений.

Наиболее важное отличие заключается в том, что нет управления транзакциями внутри хранимых функций в PostgreSQL. Транзакции возможны только во внешнем коде, на компьютерах, которые непосредственно работают с узлом БД, которые к тому же называются сопряженными. В случае использования внутри функции Oracle блока транзакции, такую функцию необходимо будет переписать вручную, разделив на несколько функций, каждая из которых отвечает за свой блок транзакции. Соответственно, вызывать ее придется во внешнем коде.

При преобразовании SQL-выражений в случае простых запросов могут потребоваться лишь небольшие правки, или можно будет обойтись без правок. Например, в PostgreSQL не поддерживаются сокращенные наименования (алиасы) в выражениях `INSERT` и `UPDATE` (в блоке изменяемых столбцов), который легко правится автоматически.

```
UPDATE voc_table vt SET vt.first_f = 1 WHERE
vt.second_f = 1;
```

Указанное выражение будет работать в Oracle, но выдаст ошибку в PostgreSQL. Потребуется убрать алиас `"vt."` из секции `SET`:

```
UPDATE voc _ table vt SET first _ f = 1 WHERE
vt.second _ f = 1;
```

Еще пример простого запроса, который можно легко изменить, связан с ключевым словом rownum:

```
SELECT rownum as rownum, first _ f as id
FROM voc _ table;
```

В БД PostgreSQL это будет выглядеть так:

```
SELECT row _ number() over () as rownum,
first _ f as id FROM voc _ table;
```

В случае сложных запросов, например, рекурсивных, потребуется полностью вручную переписать сам запрос, так как PostgreSQL не поддерживает "connect by". То есть запрос вида

```
SELECT iie.e _ type, iie.name FROM item _
in _ folder iie,
    (SELECT level lv,il.* FROM item _ location il
    START WITH (il.i _ num = 111 AND il.i _
type = 'D')
    CONNECT BY PRIOR il.folder _ num = il.i _ num
AND PRIOR il.folder _ type = il.i _ type) rrr
WHERE iie.i _ num = rrr.i _ num AND iie.i _
type = rrr.i _ type
ORDER BY rrr.lv;
```

должен быть преобразован в

```
WITH RECURSIVE rrr AS (
  SELECT 1 as level, il.* FROM item _
location il
  WHERE il.i _ num = 111 AND il.i _ type = 'D' AND
  UNION
SELECT rrr.level + 1, lev.* as level FROM
item _ location lev
JOIN rrr ON rrr.folder _ num = lev.i _ num
AND rrr.folder _ type = lev.i _ type)
SELECT iie.element _ type, iie.name FROM
rrr, item _ in _ folder iie
WHERE iie.i _ num = rrr.i _ num AND iie.i _
type = rrr.i _ type
ORDER BY rrr.level)
```

Хотелось бы также отметить, что использование встроенных функций в Oracle очень полезно, однако автоматический перевод их затруднителен. Понятно, что аналог каждой функции можно написать для PostgreSQL, но их очень много. Как следствие, обычно при переводе исходников создают только те функции, которые реально используются.

Также в силу того, что PostgreSQL не транслирует исходный код при сохранении в БД, возникают ситуации, когда функция при очередном выполнении выдает ошибку. Значит, написан исходный код функции, где есть ветвления. Вследствие особенностей использования функции, есть одна ветка ветвления, которая используется очень редко. В этой ветке есть, например, ссылка на несуществующий столбец. Тогда обычно программа будет работать правильно, но когда-нибудь внезапно будет появляться неприятная ошибка.

Что же касается представлений и материализованных представлений, то это, как известно, результат выполнения SQL-запроса. Преобразование такого SQL-выражения выполняется таким же образом, как и в хранимых процедурах.

Таким образом, перевод исходных кодов — сложный и кропотливый процесс, который требует как большого количества рутинной работы, так и достаточного опыта разработки для СУБД Oracle и PostgreSQL. Отечественная программа (ИПУ РАН) "Предконвертер хранимых процедур Oracle—PostgreSQL (OracleToPostgres_Procedure)" [5] часть исходного кода, которая относится к типовой, переводит самостоятельно, а остальную часть пытается распознать и дать подсказки.

Полученный в результате миграции код программа OracleToPostgres_Procedure проверяет на известные ей ошибки.

Изменение программного обеспечения на сопряженных узлах

Миграция БД не исчерпывается преобразованием структуры, модернизацией и переносом данных, преобразованием хранимых процедур (функций) и триггеров.

Также требуется изменение ПО, которое функционирует на узлах, непосредственно связанных (сопряженных) с узлом БД (рис. 2). Это связано с двумя обстоятельствами. Во-первых, для вызова из внешнего ПО базы данных в зависимости от типа БД используется своя процедура вызова. Таким образом, требуется заменить все точки вызова из ПО. Во-вторых, необходимо учитывать, что основная часть работы с БД может быть сосредоточена либо в хранимых процедурах (в теле БД), либо во внешнем ПО, где формируются внешние SQL-запросы и направляются в БД для выполнения. В последнем случае большинство процедур для работы с базой данных будут размещены вне БД. Более того, такие процедуры могут формироваться динамически. Здесь нужно иметь в виду две особенности. С одной стороны, для подготовки фрагментов для манипуляции с данными в БД используется SQL-язык и в процессе преобразования Oracle—PostgreSQL нужно менять только его диалекты. С другой стороны, обработка полученных из БД данных проводится на одном из языков программирования (а может и нескольких), причем фрагменты подготовки запросов к БД и другие информационные фрагменты, как правило, не отделены друг от друга. То есть при переносе нужно учитывать достаточно широкий круг обработки, что требует тщательных проверок. Конечно, если запросы изначально проектировались так, что подразумевалась возможность последующей миграции, то это меняет дело. Но, как правило, при написании ПО в первую очередь код оптимизируется для быстродействия в конкретной СУБД. Необходимо отметить, что есть два способа работы с БД:

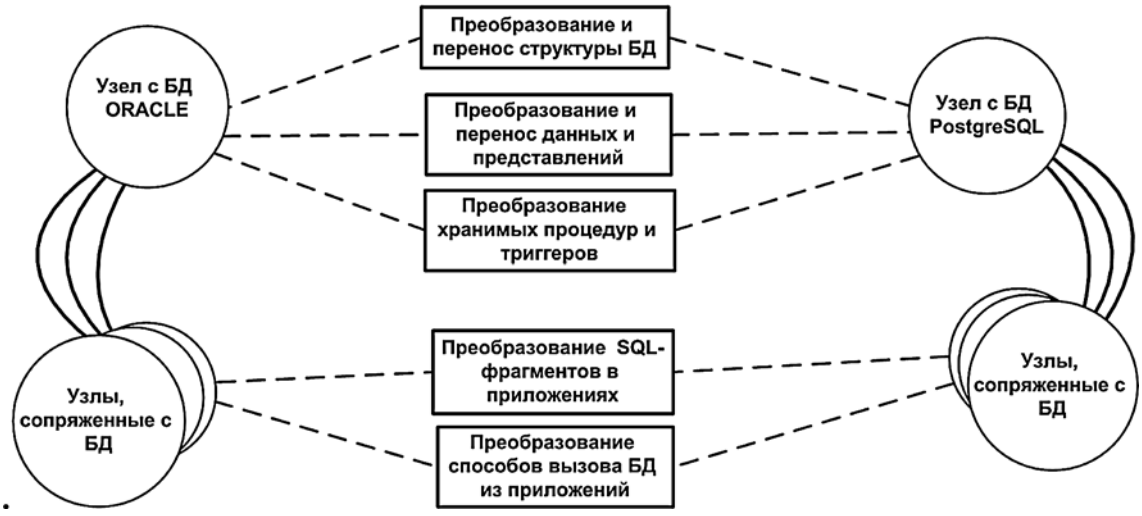

Рис. 2. Изменение серверов БД и сопряженных узлов

1) когда практически вся логика обработки данных сосредоточена в хранимых процедурах, которые являются неотъемлемой частью БД;

2) когда хранимые процедуры отсутствуют, а запросы к БД подготавливаются и запускаются из приложений, например, используются Object-relational mapping (ORM — объектно-реляционное отображение, или преобразование [6]).

Конечно, можно предположить и смешенный вариант, но обычно традиции проектирования в конкретной компании перевешивают к выбору одной из двух указанных выше возможностей. При использовании хранимых процедур пропадает необходимости каждый раз доставлять запрос на сервер базы данных. Кроме того, хранимые процедуры в теле БД обычно оттранслированы и даже могут быть постоянно размещены в оперативной памяти, что повышает быстродействие выполнения запросов.

Но соблюдение принципа, согласно которому в базе практически хранятся только данные, а все выполняемые программы размещены вне БД, также имеет право на существование, и даже, более того, широко используется. Отметим также, что программы, написанные таким образом, могут легче переводиться под управление другой СУБД. Так как здесь речь идет только о приведении в соответствие диалектов SQL, что в принципе легче, чем перевод процедур, написанных на одном языке (например, PL/SQL), на другой язык (например, PL/pgSQL) с учетом многих особенностей. Хотелось бы особо отметить применение ORM, который предназначен для сокрытия языка SQL и облегчения миграции на другую БД. Однако следует заметить, что в этом случае жертвуют производительностью.

Не вдаваясь в дискуссию о том, какая же возможность имеет преимущество, отметим лишь, что в практической жизни приходится менять и программы подготовки запросов в приложениях, и хранимые процедуры.

Миграция баз данных и минимизация остановок

С учетом стадий миграции, представленных на рис. 2, необходимо проводить перевод БД в описанной далее последовательности.

1. Подготовить скрипт для создания пустой БД нового вида, на основе существующей, например, с использованием программы "Транслятор структур Oracle—PostgreSQL (OracleToPostgres_Scheme)".

2. С помощью этого скрипта создать пустую БД нового вида.

3. После этого перенести данные из действующей БД во вновь созданную пустую БД, заполнив ее с использованием, например, программы "Трансфер данных Oracle—PostgreSQL (OracleToPostgres_Data)". Этот перенос носит предварительный характер и необходим для тестирования работ, выполняемых на следующих этапах.

4. На этом этапе необходимо подготовить все хранимые процедуры и триггеры для работы с созданной БД. Этот этап наиболее длительный, так как он подразумевает создание нового множества функций и триггеров на основе существующих. На начальном этапе нужно создать шаблоны функций с использованием программы "Предконвертер хранимых процедур Oracle—PostgreSQL (OracleToPostgres_Procedure)". Нужно отдавать себе отчет, что в автоматизированном режиме перевод всех хранимых процедур является практически невозможной задачей. Это определяется тем, что для перевода необходимо знать не только синтаксис двух языков, но и семантическое содержание каждой функции. Вместе с тем использование предконвертера упрощает проблему, по крайней мере с формальной точки зрения. Однако объем "ручных" работ, включая тестирование, на этом этапе весьма существенен.

5. Параллельно можно провести работу по преобразованию SQL-фрагментов в приложениях и преобразованию способов вызова запросов к БД из приложений.

6. Когда все готово, если содержание базы изменилось, можно приступить к окончательному переносу. Для этого следует очистить базу, созданную на этапе 3, затем поместить в нее все хранимые процедуры и триггеры, затем скопировать все измененные модули приложений туда, откуда они должны запускаться.

7. На этом этапе необходимо корректно остановить действующую БД и скопировать из нее все данные в новую БД, используя, например, ранее применяемую программу OracleToPostgres_Data.

8. Запустить новую базу и продолжить работу.

Следует отметить, что остался один проблемный вопрос. Если БД сравнительно небольшая, то последняя стадия ее миграции (остановка и копирование данных) проходит от нескольких минут до нескольких часов. На это время активная работа останавливается, и обработка текущей информации не ведется. Для многих баз такая остановка допустима, особенно в ночное время. Однако БД, обслуживающие множество пользователей, которые обращаются к ней по сети Интернет и имеют широкий географический разброс клиентов, должны работать в непрерывном режиме. Здесь остановки исключаются или могут, в крайнем случае, укладываться в несколько минут. Для таких баз алгоритм должен несколько меняться. При этом возможно использование нескольких способов.

Первый подразумевает использование параллельной записи в две базы данных Oracle и PostgreSQL из приложений. В этом случае возможен вариант с поочередным подключением таблиц. Это значит, что может отлаживаться дублирование первой таблицы, затем второй или нескольких и т. д. Разумеется, связанные таблицы должны подключаться одновременно. Результатом является параллельное использование двух БД с проведением контроля, сравнения и с последующим отключением Oracle. Этот вариант непростой, однако он является для заказывающей организации удобным, хотя и затратным, так как обеспечивает параллельное функционирование двух баз с возможным возвратом на первоначальную базу в течение определенного периода в случае выявления недостатков. В то же время — это решение задачи "в лоб" и на него разработчики, как правило, не идут.

Второй способ — использование механизмов репликации и сторонних программ, которые умеют работать с несколькими типами БД одновременно. Выше упоминалась возможность применения такого инструмента, как SymmetricDS, который позволяет одновременно подключаться к БД Oracle и PostgreSQL. Использование SymmetricDS для работ, проводимых в рамках перевода некоторых БД на Яндексе из Oracle в среду PostgreSQL [1], позволило добиться результата практически без остановки системы.

Функционирование после миграции

Очень важным вопросом является сравнительный анализ функционирования БД после миграции. Все разработчики отдают себе отчет, что Oracle — хорошо проработанная СУБД и сравнительные проверки скорости выполнения запросов, полученные на вновь созданной базе, обычно могут уступать [1] первоначальным исходным значениям на 10...15 % после переноса и первичной доработки.

Поэтому авторам было интересно проверить, как реально изменяется скорость после процедуры миграции без какой-либо первичной доработки и оптимизации. Для этого были выбраны две БД:

1) промышленная БД с накоплением информации более 10 лет объемом несколько ТБ;

2) финансовая база данных небольшого предприятия с периодом хранения информации 2—3 года объемом до 100 МБ.

Приведем примеры измерений времени поиска до и после переноса данных. Результаты отдельных измерений для промышленной базы данных представлены на рис. 3.

Здесь важным является то, что даже без каких-либо дополнительных доработок скорость поиска в БД PostgreSQL оказалось не меньше, чем в Oracle, а даже несколько больше (в пределах погрешности измерений).

Для финансовой базы небольшого предприятия картина несколько изменилась. Результаты времени подготовки простого отчета (отчет 1, без дополнительной оптимизации) приведены на рис. 4.

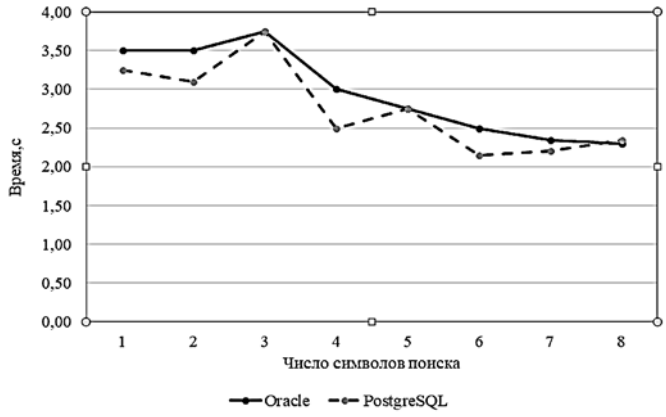

Рис. 3. Время поиска списков физических лиц в промышленной БД до переноса данных (Oracle) и после переноса (PostgreSQL)

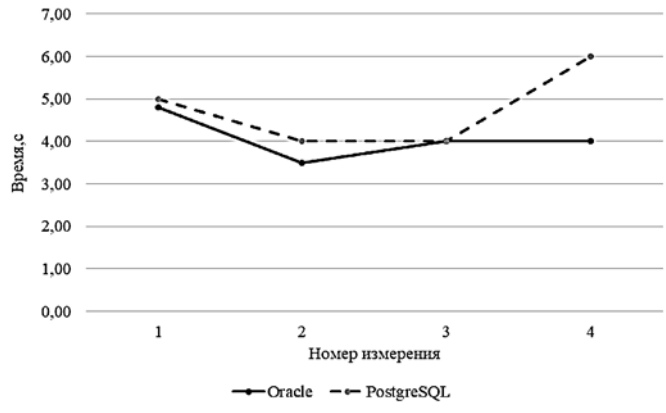

Рис. 4. Время подготовки отчета 1 в финансовой БД предприятия для Oracle и PostgreSQL

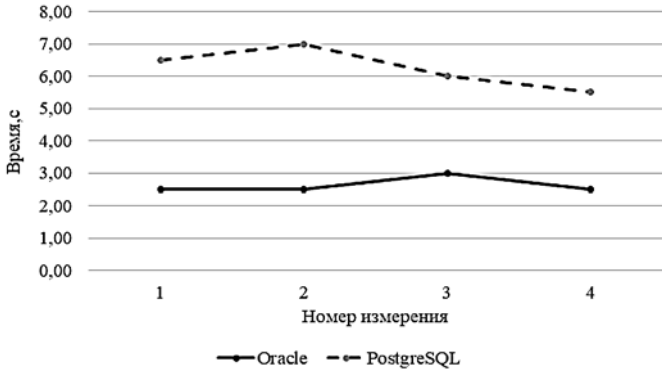

Рис. 5. Время подготовки отчета 2 в финансовой БД предприятия для Oracle и PostgreSQL

На рис. 4 видно, что время подготовки отчета для PostgreSQL несколько больше. Очевидно, объясняется это тем обстоятельством, что подготовка финансового отчета требует существенно более сложных SQL-скриптов, нежели процедура простого поиска. Поэтому для комплексных запросов в PostgreSQL нужно проводить дополнительную оптимизацию уже после переноса БД.

Это же подтверждают данные, приведенные на рис. 5, где отражены результаты времени подготовки отчета 2, несколько более сложного по структуре.

То есть для обеспечения необходимого времени выполнения запросов в БД PostgreSQL требуется проведение дополнительной оптимизации. Такая оптимизация возможна. Часто ее необходимость определяется более совершенным механизмом планирования запросов в Oracle. Хотя рассмотрение вопросов оптимизации не входит в задачу данной статьи, но из опыта авторов следует, что изменение структуры запросов часто дает положительный результат. И скорость выполнения существенно увеличивается, приближаясь к данным Oracle по крайней мере до уровня 10...15 %.

Заключение

Решение задач миграции и разработка способов перемещения с преобразованием БД из среды Oracle в среду PostgreSQL в настоящее время является весьма актуальным. Однако для сложных сетей, которые обладают развитым ПО и объемными базами данных с множеством клиентов, такая миграция характеризуется дополнительными особенностями. Они, как правило, обусловлены накоплением больших объемов данных, обеспечением требований безопасности и учетом ограничений, связанных с эксплуатационной стоимостью комплекса.

Миграция включает:

- преобразование и перенос структуры БД;
- преобразование и перенос данных и представлений;
- преобразование хранимых процедур и триггеров;
- преобразование SQL-фрагментов в приложениях;
- преобразование способов вызова БД из приложений.

В настоящей работе рассмотрено использование трех базовых программных комплексов для проведения преобразований, даны ответы на вопросы минимизации остановок в процессе миграции, а также необходимости повышения эффективности за счет оптимизации выполняемых запросов уже после выполнения основных изменений.

Вопрос о трудозатратах при переносе конкретной базы данных нужно решать отдельно. И что интересно, трудозатраты зависят не столько от объема исходного текста, сколько от его структуры. Так, например, в случае значительного использования особенностей Oracle в текстах языка PL/SQL, объем работы при переводе на язык PL/pgSQL возрастает и требуется достаточно высокая квалификация разработчиков, при условии знания семантической структуры. При минимальном использовании особенностей или их полном отсутствии большая часть работы может выполняться в автоматизированном режиме, и трудозатраты снижаются.

В качестве примера отметим, что реальные трудозатраты при переносе базы [1] без хранимых процедур объемом около 50 таблиц с допустимыми остановками функционирования, не превышающими несколько минут, составило около 400 человеко-дней. Проект миграции базы объемом около 200 таблиц с обширным объемом хранимых процедур при допустимой остановке функционирования 1-2 дня составляет приблизительно 850 человеко-дней с использованием методов [2, 3, 5].

Список литературы

1. **Синицкий В.** Экстремальная миграция на PostgreSQL: без остановки, потерь и тестирования. URL: https://habr.com/ru/company/yoomoney/blog/326998/

2. **Курако Е. А.** Транслятор структур Oracle—PostgreSQL (OracleToPostgres_Scheme). Свидетельство о государственной регистрации программы для ЭВМ № 2019614473 РФ. М.: 05.04.2019.

3. **Курако Е. А.** Трансфер данных Oracle—PostgreSQL (OracleToPostgres_Data). Свидетельство о государственной регистрации программы для ЭВМ № 2019614472 РФ. М.: 05.04.2019.

4. **Документация** PostgreSQL и Postgres Pro. URL: https://postgrespro.ru/docs/

5. **Курако Е. А., Нога Н. Л.** Предконвертер хранимых процедур Oracle—PostgreSQL (OracleToPostgres_Procedure). Свидетельство о государственной регистрации программы для ЭВМ № 2019614616 РФ. М.: 09.04.2019.

6. **Object-relational** mapping. URL: https://en.wikipedia.org/wiki/Object%E2 %80 %93relational_mapping.

On the Issue of Migrating Databases from Oracle to PostgreSQL

E. A. Kurako, keaipu@yandex.ru, **V. L. Orlov**, ovl@ipu.ru, V. A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, Moscow, 117997, Russian Federation

Corresponding author:

Kurako Evgeny A., Researcher, V. A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, Moscow, 117997, Russian Federation
E-mail: keaipu@yandex.ru

Received on October 19, 2021
Accepted on November 18, 2021

The methods of data transfer from Oracle to PostgreSQL environment are considered, taking into account the step-by-step organization of actions and deter-mining the sequence of migration. The main directions for the transformation of the structure of databases, data formats and executed objects are highlighted. New methods and programs for converting the structure and formats in automatic mode are proposed. The methods of transformation of stored procedures and triggers written in procedural languages, which are PL/SQL and PL/PGSQL, are defined. For cases of possible ambiguous translation, programs have been developed that form hints that facilitate manual operation.

The task of managing transactions inside stored functions in PostgreSQL is difficult. If such transactions are possible for Oracle, then in the PostgreSQL environment, it is necessary to organize them only in external code. They are usually published on computers that work directly with the database server. In this case, the division of the original function into several functions is used, each of which is responsible for its own transaction block. Each such function is called from external code.

The issues of changing software on nodes directly connected (interfaced) with the database server and the issues of minimizing stops during migration are considered, which makes it possible to carry out migration for geographically distributed and round-the-clock functioning objects. Examples of changes in the speed of execution of requests before and after migration for various types of systems are given.

Keywords: *database, DBMS, Oracle, PostgreSQL, migration, structure, data, function, stored procedures, PL/SQL, PL/pgSQL*

For citation:

Kurako E. A., Orlov V. L. On the Issue of Migrating Databases from Oracle to PostgreSQL, *Programmnaya Ingeneria,* 2022, vol. 13, no. 1, pp. 32—40.

DOI: 10.17587/prin.13.32-40

References

1. **Sinitsky V.** Extreme migration to PostgreSQL: without stopping, losses and testing, available at: https://habr.com/ru/company/yoomoney/blog/326998/ (in Russian).

2. **Kurako E. A.** Translator of Oracle—PostgreSQL structures (Oracle-ToPostgres_scheme). Certificate of state registration of the computer program No. 2019614473 of the Russian Federation, Moscow: 05.04.2019 (in Russian).

3. **Kurako E. A**. Oracle-PostgreSQL Data Transfer (Oracle-ToPostgres_data). Certificate of state registration of the computer program No. 2019614472 of the Russian Federation, Moscow: 05.04.2019. (in Russian).

4. **PostgreSQL** and Postgres Pro documentation, available at: https://postgrespro.ru/docs/

5. **Kurako E. A., Noga N. L.** Oracle—PostgreSQL stored procedure preconverter (oracletopostgres_procedure), Certificate of state registration of the computer program No. 2019614616 of the Russian Federation. Moscow: 09.04.2019 (in Russian).

6. **Object-relational** mapping, available at https://en.wikipedia.org/wiki/Object%E2 %80 %93relational_mapping

УДК 004.942 DOI: 10.17587/prin.13.41-48

Е. В. Авдеев, ассистент, j-avdeev@yandex.ru,
Самарский национальный исследовательский университет

О некоторых аспектах уточнения температурного распределения в разработанном программном инструментарии моделирования нагрева порошка лазером

Для изготовления прототипов и мелкосерийного производства все большую популярность завоевывает применение аддитивных технологий 3D-печати изделий. В статье представлены результаты разработки программного инструментария для моделирования нагрева порошка лучом лазера, плавления порошка и формирования области ванны расплава. Представленные результаты являются текущим состоянием исследований по разработке приложения для моделирования накопленных деформаций, возникающих в процессе селективного лазерного плавления (SLM). Технология SLM позволяет в короткие сроки изготовить изделие сложной формы. В качестве материала SLM предусматривает использование порошковых металлов и сплавов, включая титан, алюминий, нержавеющую сталь. Изготовленные по технологии SLM детали вследствие цикличного неравномерного нагревания и остывания частиц порошка подвержены появлению деформаций и трещин. Описано приложение, реализующее метод расчета геометрического размера ванны расплава в динамической постановке. Для сокращения вычислительных затрат используется метод адаптации сетки. Проведено сравнение результатов моделирования с данными натурных экспериментов. Проведена оценка эффективности адаптации сетки с точки зрения сокращения вычислительных затрат.

Ключевые слова: *выборочная лазерная плавка, selective laser melting, SLM, аддитивные технологии, OpenFOAM*

Введение

Аддитивные технологии (АТ) позволяют изготавливать трехмерные детали путем постепенного сплавления тонких слоев материала. Главным преимуществом АТ является возможность производства сложных или "индивидуальных" деталей без использования дорогостоящих инструментов, таких как штампы, пуансоны, литейные формы. Технология выборочной лазерной плавки (*Selective Laser Melting*, SLM) позволяет использовать в качестве материалов порошковые металлы и сплавы, включая титан, алюминий, нержавеющую сталь. Это делает технологию SLM очень востребованной для изготовления деталей, необходимых в медицине и аэрокосмической технике. При этом изготовленные детали, вследствие цикличного неравномерного нагревания и остывания частиц порошка, подвержены появлению деформаций и трещин. Задача разработки программного инструментария, позволяющего моделировать процесс 3D-печати методом SLM, возникла в ходе проекта по моделированию нагрева и плавления порошка лазером [1].

Приложение, реализующее модель процесса изготовления деталей методом выборочной лазерной плавки, позволяет получать результаты, хорошо согласующиеся с экспериментальными данными. С использованием разработанного инструментального средства осуществляются настройки изготовления детали, такие как мощность лазера или иного источника теплоты, скорость перемещения лазерного луча, диаметр лазерного пятна. Это позволяет существенно сократить трудоемкость настройки оборудования и сократить стоимость изготовления деталей. Приложение, моделирующее процесс 3D-печати, также позволит прогнозировать и устранять возможное возникновение деформаций и трещин в изготавливаемой детали.

Согласно технологии SLM изготавливаемая деталь формируется путем поочередного нанесения тонкого слоя порошкового материала и его выборочного плавления с помощью лазерного луча или иного источника теплоты. Толщина каждого слоя составляет 20...100 мкм. Лазерный луч сплавляет частички металлического порошка и образует треки (сварные дорожки, сварные швы), которые располагаются на заданном расстоянии друг от друга. Последовательное сплавление расплавленного порошка с относительно холодной подложкой или ранее нанесенными слоями приводит к высоким градиентам температуры, термической деформации и возникновению

накопленных деформаций (*inherent strains*) [2, 3]. Накопленные деформации, в свою очередь, приводят к изменению формы (короблению) и образованию трещин в детали при остывании. В настоящей работе представлены результаты по разработке программного комплекса моделирования нагрева порошка лучом лазера, плавления порошка и формирования области — ванны расплава. Работа разрабатываемого приложения основана на использовании метода накопленных деформаций [4-6].

Постановка задачи

На верхнем уровне систематизации процесс разработки программного инструментария был разделен на перечисленные далее этапы.

1. Моделирование распределения температуры в ванне расплава в стационарной постановке. Геометрия ванны расплава моделируется в зависимости от формы и кривой распределения энергии лазерного луча путем решения уравнения теплопроводности с граничными условиями: подложка/проплавленный слой, слой металлического порошка, среда инертного газа. Моделирование проводится методом конечных объемов. Решение уравнения происходит в стационарной постановке.

2. Моделирование распределения температуры в сплавляемом слое в динамической (нестационарной) постановке в зависимости от технологических параметров (таких как скорость и шаг движения луча лазера). Определение геометрии движущейся ванны расплава и распределения температуры в ванне расплава. Эти данные позволят описать фронт перехода из почти нулевой теплопроводности (вне фронта проводимости) к проводимости перед фронтом ванны расплава.

3. Решение задачи теплообмена и структурного анализа в верхнем слое конечных объемов, содержащем сплавляемый слой материала. Расчет накопленных деформаций в этом слое. Термическое напряжение рассчитывается на основе поля температур [7, 8] и на основе напряжений, связанных с изменением объема в ходе фазового перехода [9].

4. Решение задачи структурного анализа для конечных объемов нижних слоев. К граничным узлам верхнего слоя, в котором происходит сплавление, прикладывается нагрузка, вызванная накопленными деформациями [10, 11]. Рассчитывается упругая реакция в этих узлах от приложенной нагрузки.

5. Определение деформаций на основе рассчитанных деформаций в объеме детали. Инвертируя эти деформации в исходной 3D-модели выращиваемой детали, можно снизить воздействия термических напряжений.

В качестве основы для разработки программного инструментария используется открытая библиотека OpenFOAM. Данная библиотека содержит большое число примеров моделирования задач механики сплошных сред, решателей и готовых реализаций граничных условий. При этом код всех модулей открыт, распространяется по лицензии GPL 3 и может быть использован для создания собственных решателей, реализующих собственные модели. Большая часть исходного кода OpenFOAM написана на языке программирования C++.

Основные модули разрабатываемого приложения

В OpenFOAM процесс моделирования задач механики сплошных сред разделен на следующие этапы.

1. Подготовка геометрии — 2D- или 3D-модели, объема, который будет использоваться для моделирования процесса.

2. Подготовка сетки — разбиение рабочего объема, сформированного на предыдущем шаге, на конечные объемы или конечные элементы. На этом же шаге проводят выделение границ — логически сгруппированных ячеек или границ ячеек.

3. Настройка граничных условий, когда необходимо задать начальные граничные условия для каждой границы, определенной на предыдущем шаге. Набор доступных граничных условий связан с выбранным решателем.

4. Запуск решателя — решатель принимает на вход сетку, перечень границ, назначенных граничных условий. В решателе находится реализация одного или нескольких уравнений баланса. На основе этих уравнений и заданных начальных значений решатель итерационно формирует значения физических величин внутри геометрии вычислительного объема.

В соответствии с перечисленными этапами в приложении моделирования спекания металлического порошка можно выделить следующие модули: модуль разбиения вычислительного объема (построения сетки); модуль моделирования граничных условий; решатель.

В настоящее время в актуальной версии OpenFOAM отсутствуют полностью готовые решатели, позволяющие смоделировать процесс плавления порошка лазером. Тем не менее есть библиотеки, в том числе сторонние, которые можно использовать в качестве основы и значительно сократить трудозатраты по разработке приложения.

Для моделирования граничного условия передачи теплоты от лазера использовался сторонний модуль laserConvBC [12]. Модуль laserConvBC позволяет задать граничное условие, имитирующее нагрев лучом лазера. При этом задаются такие параметры, как мощность лазера, координаты точки лазера (X, Y, Z), направление луча лазера (dX, dY, dZ), размеры пятна (dX, dY), время нагрева и др. Дополнительно laserConvBC позволяет указать траекторию перемещения лазерного луча, если задача решается в динамической постановке.

В качестве основного метода используемого в решателе приложения выбран метод конечного объема (МКО). OpenFOAM включает в себя необходимые библиотеки, которые можно использовать в качестве основы для реализации МКО. Для моделирования распределения температуры в сплавляемом слое в качестве основы выбран решатель библиотеки OpenFOAM — laplacianFoam, который позволяет получить решение уравнения баланса вида

$$\frac{\partial}{\partial t}(T) - \nabla(D_T \nabla T) = S_T, \quad (1)$$

где T — некоторая искомая скалярная величина, в рассматриваемой задаче это температура (K); t — время (с); D_T — коэффициент диффузии, в данном случае это коэффициент температуропроводности (м^2/с); S_T — коэффициент источников, в рассматриваемой задаче это источники теплоты и теплопередача "порошок—воздух".

Схематично все основные разработанные и модифицированные модули приложения представлены на рис. 1. Серым цветом на рис. 1 отмечены блоки, которые были модифицированы или полностью разработаны для реализации функциональных возможностей приложения. Белым цветом отмечены блоки, которые использовались без изменений. Например, в решатель laplacianFoam была интегрирована поддержка динамической сетки. Дополнительно в решатель laplacianFoam интегрирована библиотека laserConvBC, реализующая граничное условие нагрева лучом лазера.

Рассмотрим подробнее модуль построения сетки. Для сокращения вычислительных затрат на моделирование в модуль построения сетки интегрирован модуль динамической адаптации сетки (*Adaptive Mesh Refinement*, AMR). Адаптация сетки проводится на основе матрицы-вектора оценок, формируемого в соответствующем модуле.

При моделировании процесса изготовления детали сложной формы вычислительные затраты достаточно велики. Для сокращения вычислительных затрат был использован метод, основанный на адаптации сетки. В одной из предыдущих работ автора [13] показано, что разбиение вычислительного домена на конечные объемы (построение сетки) влияет на сходимость итерационного решения и время вычисления задачи.

Выражение (1) после проведения дискретизации по пространству и времени может быть преобразовано и представлено в матричном виде:

$$\mathbf{Ax} = \mathbf{b},$$

где **A** — квадратная разреженная матрица $n \times n$, матрица коэффициентов дискретизации; **x** — вектор $n \times 1$, вектор искомых значений; **b** — вектор $n \times 1$, вектор заданных значений.

Адаптация сетки проводится на основе поля **F**, которое, в свою очередь, формируется на основе матрицы дискретизации **A**:

$$f_i = |m_{ii}| + \sum_{i \neq j} |m_{ij}|,$$

$$f_{i\ нормализованное} = \frac{f_i}{\max(f_i)},$$

где m_{ii}, m_{ij} — диагональные и недиагональные элементы матрицы $\mathbf{M} = \mathbf{I} - \mathbf{A}$ соответственно; **I** — единичная матрица.

Схематично процесс адаптации сетки в составе общего процесса моделирования изображен на рис. 2.

При моделировании распределения температуры были сделаны следующие допущения:

- слой порошкового материала моделируется как однородное и непрерывное твердое тело;
- распределение теплового потока внутри пятна нагрева, создаваемого лазерным лучом, от центра к краям моделируется в виде распределения Гаусса;
- модель не учитывает потери теплоты при переходе порошкового материала из твердого в расплавленное состояние;
- пятно лазера имеет форму круга;
- коэффициент теплопередачи между расплавленным материалом и окружающей средой принят постоянным и независящим от температуры.

Прикладная интерпретация

В описываемом далее примере представлены результаты моделирования распределения температуры в ванне расплава в нестационарной постановке. Геометрическая модель нагреваемого лазером порошка представляет собой параллелепипед толщиной 75 мкм (рис. 3).

Исходная "грубая" конечно-объемная модель содержит 25 000 элементов (размерность по осям *X*, *Y*, *Z* составляет 50, 50, 10 ячеек соответственно), тип конечных элементов — гексаэдры. Конечно-объемная

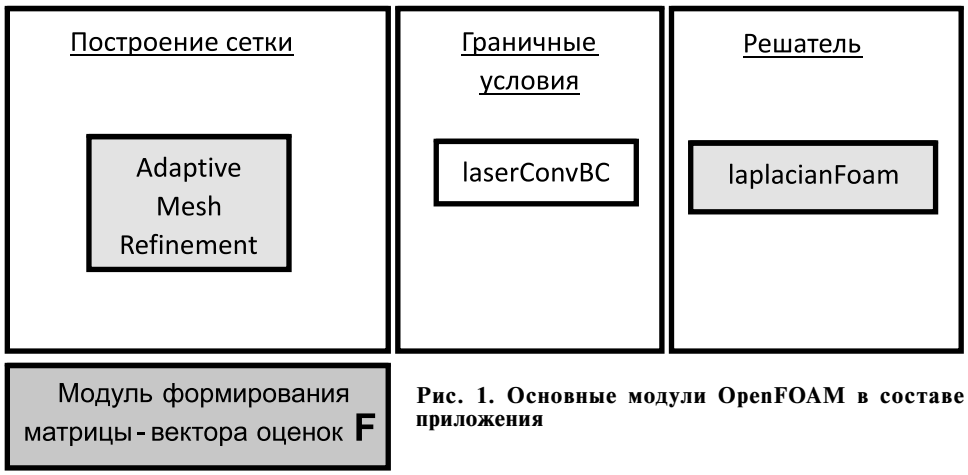

Рис. 1. Основные модули OpenFOAM в составе приложения

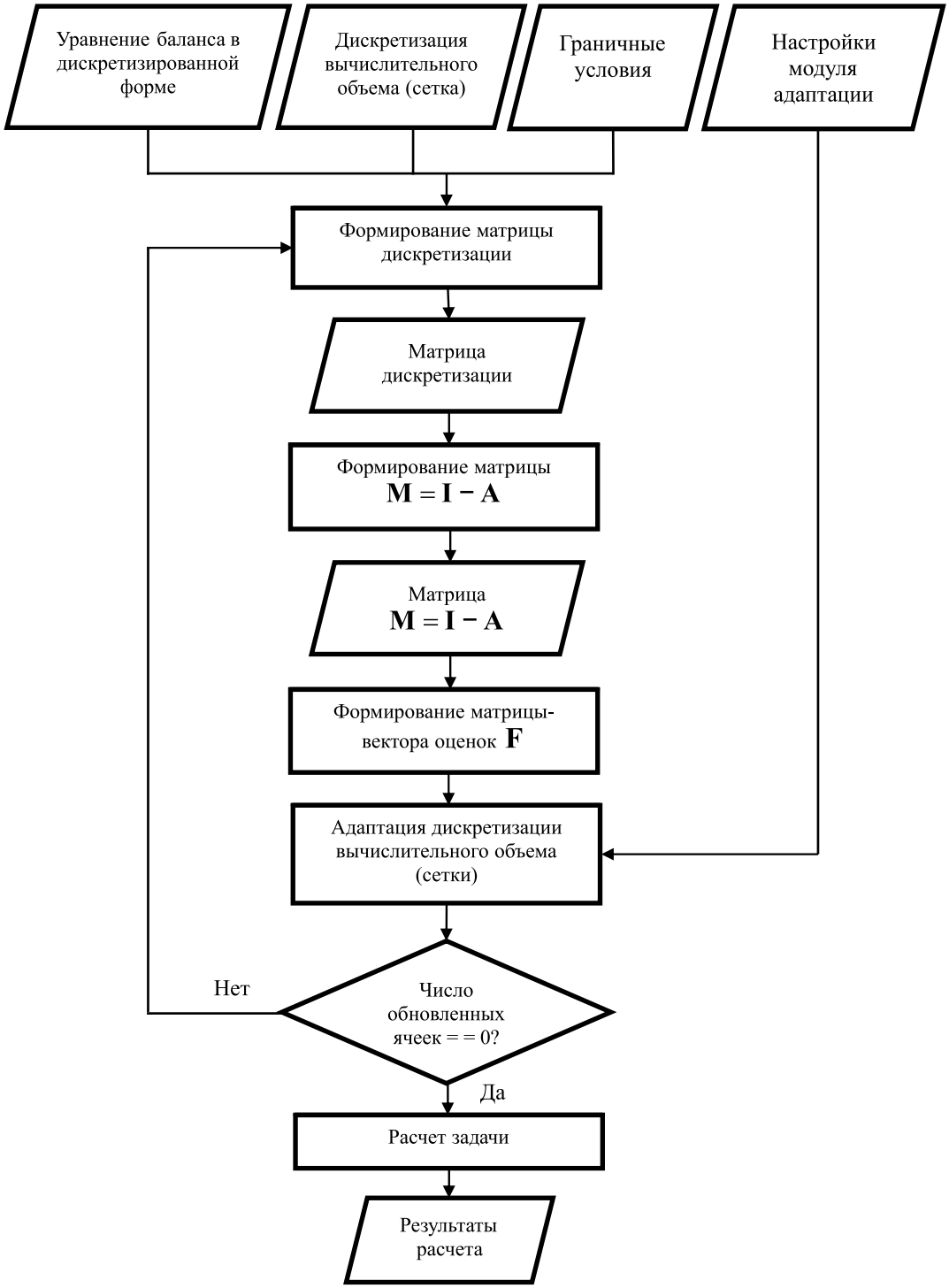

Рис. 2. Структурная схема информационной технологии адаптации сетки

модель (сетка) изображена на рис. 4. Далее этот расчетный случай обозначим как расчетный случай 1.

Для увеличения скорости сходимости итерационного решателя и для сокращения вычислительных затрат дополнительно генерируется адаптированный вариант сетки. Адаптированная сетка формируется на основе исходной сетки с помощью модифицированного решателя с добавленным модулем адаптации сетки. Решатель запускается на одну итерацию, в ходе которой формируется матрица дискретизации и массив значений, на основе которого происходит непосредственно адаптация сетки.

Конечно-объемная модель адаптированного варианта изображена на рис. 5. Данный расчетный случай обозначим как 2.

Адаптированная сетка содержит в 100 000 элементов. Максимальное число 100 000 элементов было задано в настройках модуля адаптации.

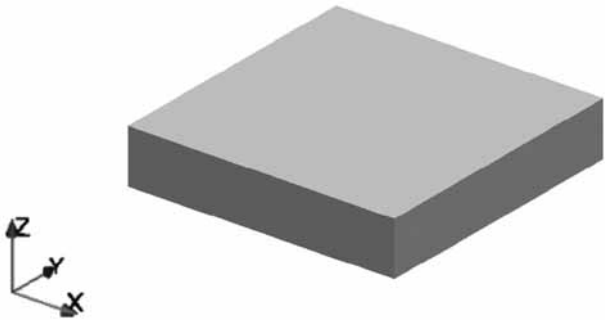

Рис. 3. Геометрия нагреваемого лазером слоя

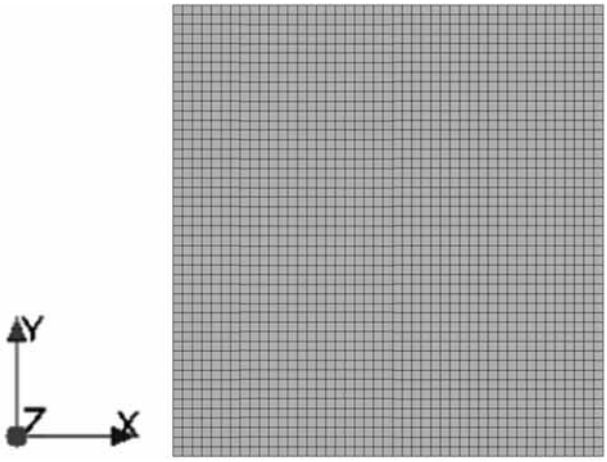

Рис. 4. Исходная равномерная конечно-объемная модель. Расчетный случай 1

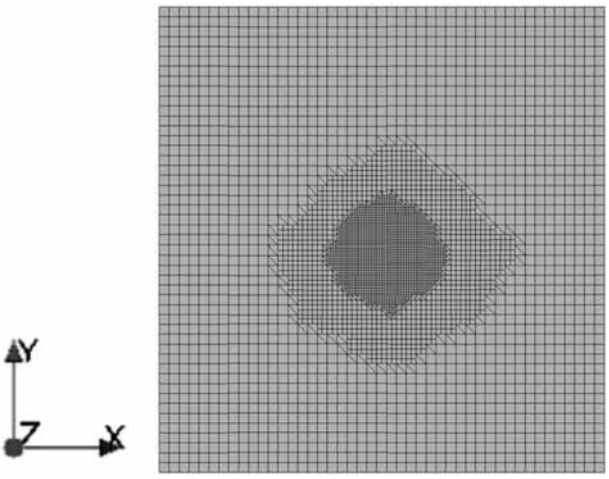

Рис. 5. Адаптированная конечно-объемная модель. Расчетный случай 2

Как видно на рис. 5, в центральной области конечно-объемной модели находятся более мелкие элементы, так как это потенциально позволит уменьшить вычислительную погрешность.

Для оценки эффективности адаптивной сетки был сформирован расчетный случай 3 с измельченной сеткой, изображенной на рис. 6 (размерность по осям *X*, *Y*, *Z* составляет 100, 100 и 20 ячеек соответственно).

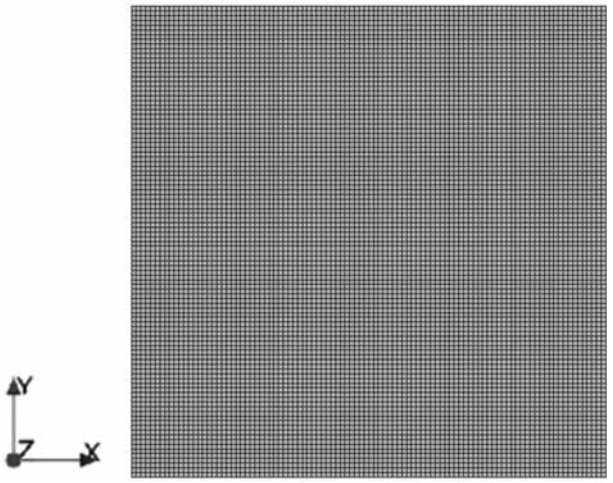

Рис. 6. Конечно-объемная модель. Расчетный случай 3

Для оценки точности моделирования разработанного приложения использовались результаты моделирования, приведенные в работе [14]. Эти результаты получены с помощью программного пакета Ansys. В указанной работе авторы приводят подробное описание граничных условий, полученных результатов и сравнение с экспериментальными данными. Поэтому сравнение с работой [14] позволяет оценить погрешность моделирования с помощью разработанного приложения. Для того чтобы сравнить полученные результаты с опубликованными в работе [14], использовались такие же граничные условия, как и в работе [14]. Граничные условия представлены на рис. 7. Детально граничные условия описаны в табл. 1.

Таблица 1

Граничные условия задачи

Параметр	Значение
Толщина слоя порошка	75 мкм
Температура окружающей среды	293 K
Коэффициент теплопередачи порошок—воздух	10 Вт/(м^2·°C)
Мощность лазера	300 Вт
Скорость перемещения пятна лазера	1,6 мм/с

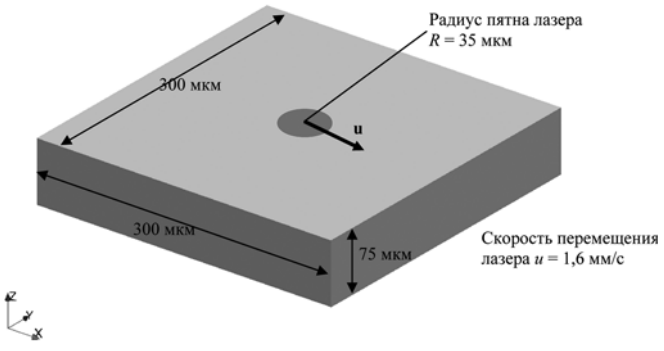

Рис. 7. Граничные условия задачи

Таблица 2

Сравнение точности решения

Параметр	Результаты в работе [14]	Расчетный случай		
		1 (исходная сетка)	2 (адаптированная сетка)	3 (мелкая сетка)
Диаметр ванны расплава, мкм	140	125	147	145
Глубина ванны расплава, мкм	51	32	41	41
Диаметр ванны расплава, отличие от [14], %	0	−11	+5	+4
Глубина ванны расплава, отличие от [14], %	0	−48	−20	−20

При сравнении точности в качестве сравниваемых параметров выбраны продольная, поперечная длина и глубина ванны расплава. В качестве области ванны расплава принималась область, в которой температура материала достигала 660 K и выше.

На рис. 8 (см. вторую сторону обложки) изображено полученное в результате расчета распределение температуры, приведенное в работе [14].

На рис. 9—11 (см. вторую и третью стороны обложки) изображены распределения температуры случаев: 1 — исходная грубая сетка; 2 — адаптированная сетка; 3 — мелкая сетка. Для визуального сравнения формы распределения температуры на рис. 9—11 использовалась одинаковая шкала.

Было также получено и проведено сравнение с результатами работы [14] распределение температуры в сечении. На рис. 12 приведено расположение плоскости сечения, в которой показано распределение температуры на рис. 13—16 (см. третью и четвертую стороны обложки).

Для сравнения полученные результаты расчетов сведены в табл.2. В табл. 2 показано отклонение значений величин, полученных в расчетных случаях 1, 2, 3, от результатов работы [14]. Отклонения связаны с различной реализацией математических моделей в Ansys и OpenFOAM. При этом отклонения результатов, полученных в расчетных случаях 2 и 3, меньше, чем полученные в расчетном случае 1.

На рис. 17 (см. четвертую сторону обложки) представлены графики убывания невязок для всех расчетных случаев. Под невязкой в данном случае подразумевается разница между известными значениями и соответствующими значениями, полученными с использованием приближенных неизвестных значений для заданной итерации. Например, пусть требуется найти такое x, что значение функции:

$$f(x) = b.$$

Примем, что задача решается итерационным методом. Для расчета невязки для итерации 5 необходимо подставить приближенное значение x_5, вместо x:

$$b - f(x_5).$$

На рис. 17 (см. четвертую сторону обложки) видно, что в расчетных случаях 2 и 3 невязка уменьшается немного быстрее, чем в расчетном случае 1. Это связано с тем, что конечно-объемная модель с большим числом элементов позволяет снизить погрешности, вызванные неортогональностью и скошенностью ячеек. Число элементов конечно-объемной модели также влияет на вычислительные затраты расчетного случая.

Таблица 3

Вычислительное время расчетов

Расчетный случай	Вычислительное время, с
1 (исходный)	2
2 (адаптированный)	8
3 (измельченный)	28

В табл. 3 для сравнения приведено вычислительное время, необходимое для расчета распределения температуры на одном шаге лазера. Для расчетного случая 2 время указано с учетом вычислительного времени на адаптацию.

Как видно из данных табл. 3, вычислительное время в расчетном случае 3 намного превышает вычислительное время расчетных случаев 1 и 2. Это связано с большим числом элементов конечно-объемной модели в расчетном случае 3, что потребовало значительных вычислительных ресурсов.

Заключение

Продемонстрировано текущее состояние разработки приложения для моделирования процесса изготовления деталей по технологии SLM. Приложение позволяет моделировать нагревание порошка лазером и анализировать форму ванны расплава. Продемонстрирована согласованность полученных результатов с опубликованными экспериментальными данными. Отклонение от экспериментальных данных составляет не более 20 %.

Продемонстрировано также, что применение адаптации сетки позволяет повысить сходимость решения задачи и снизить вычислительные затраты. Использование адаптации сетки позволяет сократить время расчета задачи более чем в 3 раза.

Исходный код решателя и решаемой задачи доступны по ссылке [15].

Список литературы

1. **Avdeev E. V.** Steady state numerical calculation of the melt-pool shape // Journal of Pysics: Conference Series. 2019. Vol. 1368, Issue 4. URL: https://iopscience.iop.org/article/10.1088/1742-6596/1368/4/042069/pdf
2. **Lewis G., Schlienger E.** Practical considerations and capabilities for laser assisted direct metal deposition // Mater Des. 2000. Vol. 21, No. 4. P. 417—423.
3. **Mercelis P., Kruth J.** Residual stresses in selective laser sintering and selective laser melting // Rapid Prototyp. 2006. Vol.12, No. 5. P. 254—265.
4. **Ueda Y., Fukuda K., Tanigawa M.** New measuring method of three residual stresses based on theory of inherent strain (welding mechanics, strength & design) // Transactions of JWRI. 1979. Vol. 8. P. 249—256.
5. **Yuan M., Ueda Y.** Prediction of residual stresses in welded T- and I-joints using inherent strains // Journal of engineering materials and technology. 1996. Vol. 118. P. 229—234.
6. **Hill M., Nelson D.** The inherent strain method for residual stress determination and its application to a long welded joint // ASME-PUBLICATIONS-PVP. 1995. Vol. 318. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.30.2893&rep=rep1&type=pdf
7. **Lindgren L-E.** Finite element modeling and simulation of welding part 1: increased complexity // J Therm Stresses. 2001. Vol. 24, No. 2. P. 141—192.
8. **Lindgren L-E.** Finite element modeling and simulation of welding part 2: improved material modeling // J Therm Stresses. 2001. Vol. 24, No. 3. P. 195—231.
9. **DebRoy T.** A perspective on residual stresses in welding // Sci Technol Weld Join, 2011. Vol. 16, No. 3. P. 204—208.
10. **Martukanitz R., Michaleris P., Palmer T., DebRoy T., Liu Z.-K., Otis R.** Towards an integrated computational system for describing the additive manufacturing process for metallic materials // Addit Manuf. 2014. Vol. 1-4. P. 52—63.
11. **Michaleris P.** Modeling metal deposition in heat transfer analyses of additive manufacturing processes // Finite Elem Anal Des. 2014. Vol. 86. P. 51—60.
12. **Holzmann T.** laserConvectionBC. URL: https://bitbucket.org/shor-ty/laserconvectionbc/src/master/
13. **Avdeev E., Fursov V., Ovchinnikov V.** An adaptive mesh refinement in the finite volume method // 2015 Computer Science. URL: https://doi.org/10.18287/1613-0073-2015-1490-234—241.
14. **Han Q., Setchi R., Evans S., Qui C.** Three-dimensional finite element thermal analysis in selective laser melting of Al—Al_2O_3 powder // 2016 Solid Freeform Fabrication 2016: Proceedings of the 27th Annual International Solid Freeform Symposium. URL: http://utw10945.utweb.utexas.edu/sites/default/files/2016/010-Han.pdf
15. **LaserCase** OpenFOAM. URL: https://github.com/j-avdeev/LaserCase

On some Aspects of the Refinement of the Temperature Distribution in the Developed Software Toolkit for Modeling Powder Heating by a Laser

E. V. Avdeev, j-avdeev@yandex.ru, Samara National Research University, Samara, 443086, Russian Federation

Corresponding author:

Avdeev Evgeniy V., Assistant, Samara National Research University, Samara, 443086, Russian Federation
E-mail: j-avdeev@yandex.ru

Received on October 31, 2021
Accepted on November 17, 2021

Nowadays, the use of additive manufacturing is gaining popularity. The cost of product manufacturing with additive manufacturing (AM) in large-scale production is usually higher than the traditional technologies (for example, casting, milling), therefore AM is often used for prototyping and small-scale production.

The paper presents results of powder heating by a laser beam modeling, powder melting modeling, and area-bath melt formation. The presented results are part of the work of solver development for modeling the selective laser melting (SLM) process. SLM technology makes it possible to manufacture a product with a complex shape in a fairly short time. As a material SLM provides for the use of powder metals and alloys, including titanium, aluminum, stainless steel. In this case, the shape of the manufactured parts is often prone to deformations and cracks. The reason for the deformation is the presence of inherited stresses due to cyclic uneven heating and cooling, arising during layer-by-layer melting of powder particles. To solve this problem, it was decided to develop a software tool that would analyze the manufacturing process and correct it, reducing the occurrence of residual stresses.

The article presents the results of modeling and calculating the geometric size of the melt pool in a dynamic setting. A comparison is made between the results obtained in the course of this work and the results of experiments.

Modeling the selective fusion process entails high computational costs. The article presents a method based on mesh adaptation, which allows to reduce the computational costs of modeling. The analysis of the effectiveness of this method is carried out.

Keywords: *selective laser melting, selective laser melting, SLM, additive technologies, OpenFOAM*

For citation:

Avdeev E. V. On some Aspects of the Refinement of the Temperature Distribution in the Developed Software Toolkit for Modeling Powder Heating by a Laser, *Programmnaya Ingeneria,* 2022, vol. 13, no. 1, pp. 41—48.

DOI: 10.17587/prin.13.41-48

References

1. **Avdeev E. V.** Steady state numerical calculation of the melt-pool shape, *Journal of Pysics: Conference Series,* 2019, vol. 1368, issue 4, available at: https://iopscience.iop.org/article/10.1088/1742-6596/1368/4/042069/pdf

2. **Lewis G., Schlienger E.** Practical considerations and capabilities for laser assisted direct metal deposition, *Mater Des.,* 2000, vol. 21, no. 4, pp. 417—423.

3. **Mercelis P., Kruth J.** Residual stresses in selective laser sintering and selective laser melting, *Rapid Prototyp.,* 2006, vol. 12, no. 5, pp. 254—265.

4. **Ueda Y., Fukuda K., Tanigawa M.** New measuring method of three residual stresses based on theory of inherent strain (welding mechanics, strength & design), *Transactions of JWRI*, 1979, vol. 8, pp. 249—256.

5. **Yuan M., Ueda Y.** Prediction of residual stresses in welded T- and I-joints using inherent strains, *Journal of engineering materials and technology*, 1996, vol. 118, pp. 229—234.

6. **Hill M., Nelson D.** The inherent strain method for residual stress determination and its application to a long welded joint, *ASME-PUBLICATIONS-PVP,* 1995, vol. 318, available at: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.30.2893&rep=rep1&type=pdf

7. **Lindgren L-E.** Finite element modeling and simulation of welding part 1: increased complexity, *J Therm Stresses*, 2001, vol. 24, no. 2, pp. 141—192.

8. **Lindgren L.-E.** Finite element modeling and simulation of welding part 2: improved material modeling, *J Therm Stresses,* 2001, vol. 24, no. 3, pp. 195—231.

9. **DebRoy T.** A perspective on residual stresses in welding, *Sci Technol Weld Join,* 2011, vol.16, no. 3, pp. 204—208.

10. **Martukanitz R., Michaleris P., Palmer T., DebRoy T., Liu Z-K., Otis R.** Towards an integrated computational system for describing the additive manufacturing process for metallic materials, *Addit Manuf.,* 2014, vol. 1-4, pp. 52—63.

11. **Michaleris P.** Modeling metal deposition in heat transfer analyses of additive manufacturing processes, *Finite Elem Anal Des.,* 2014, vol. 86, pp. 51—60.

12. **Holzmann T.** laserConvectionBC, available at: https://bitbucket.org/shor-ty/laserconvectionbc/src/master/

13. **Avdeev E., Fursov V., Ovchinnikov V.** An adaptive mesh refinement in the finite volume method, *2015 Computer Science*, available at: https://doi.org/10.18287/1613-0073-2015-1490-234-241

14. **Han Q., Setchi R., Evans S., Qui C.** Three-dimensional finite element thermal analysis in selective laser melting of Al-Al2O3 powder, *2016 Solid Freeform Fabrication 2016: Proceedings of the 27th Annual International Solid Freeform Symposium*, available at: http://utw10945.utweb.utexas.edu/sites/default/files/2016/010-Han.pdf

15. **LaserCase** OpenFOAM, available at: https://github.com/j-avdeev/LaserCase

ООО "Издательство "Новые технологии". 107076, Москва, ул. Матросская Тишина, д. 23, стр. 2
Технический редактор *Е. М. Патрушева*

Сдано в набор 23.11.2021 г. Подписано в печать 23.12.2021 г. Формат 60×88 1/8. Заказ PI121
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 **Е. В. Авдеева**

«О НЕКОТОРЫХ АСПЕКТАХ УТОЧНЕНИЯ ТЕМПЕРАТУРНОГО РАСПРЕДЕЛЕНИЯ В РАЗРАБОТАННОМ ПРОГРАММНОМ ИНСТРУМЕНТАРИИ МОДЕЛИРОВАНИЯ НАГРЕВА ПОРОШКА ЛАЗЕРОМ»

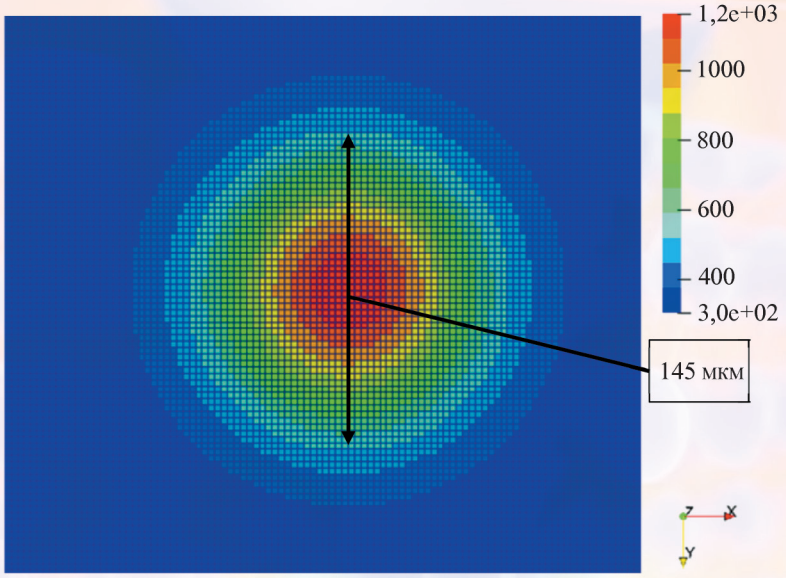

Рис. 11. Распределение температуры, диаметр ванны расплава 145 мкм, расчетный случай 3 (мелкая сетка)

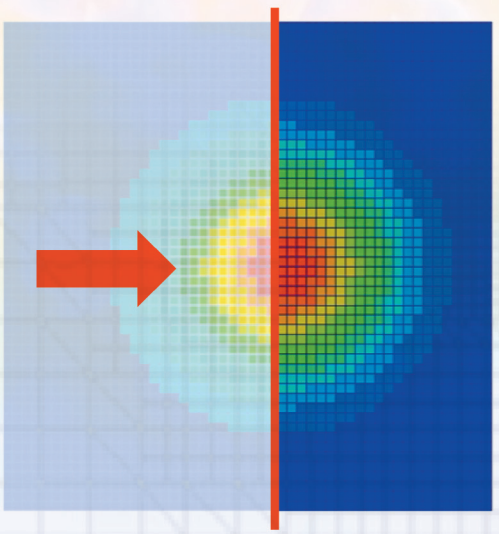

Рис. 12. Расположение плоскости сечения и направление обзора

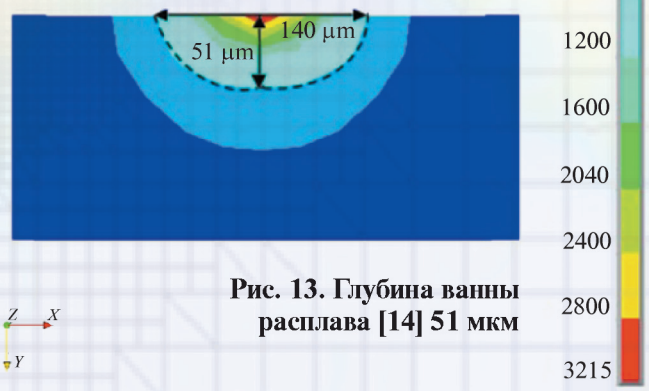

Рис. 13. Глубина ванны расплава [14] 51 мкм

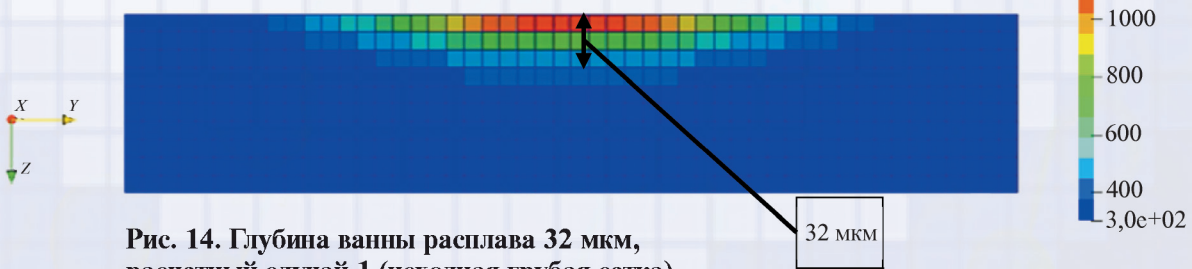

Рис. 14. Глубина ванны расплава 32 мкм, расчетный случай 1 (исходная грубая сетка)

Рисунки к статье **Е. В. Авдеева**

«О НЕКОТОРЫХ АСПЕКТАХ УТОЧНЕНИЯ ТЕМПЕРАТУРНОГО РАСПРЕДЕЛЕНИЯ В РАЗРАБОТАННОМ ПРОГРАММНОМ ИНСТРУМЕНТАРИИ МОДЕЛИРОВАНИЯ НАГРЕВА ПОРОШКА ЛАЗЕРОМ»

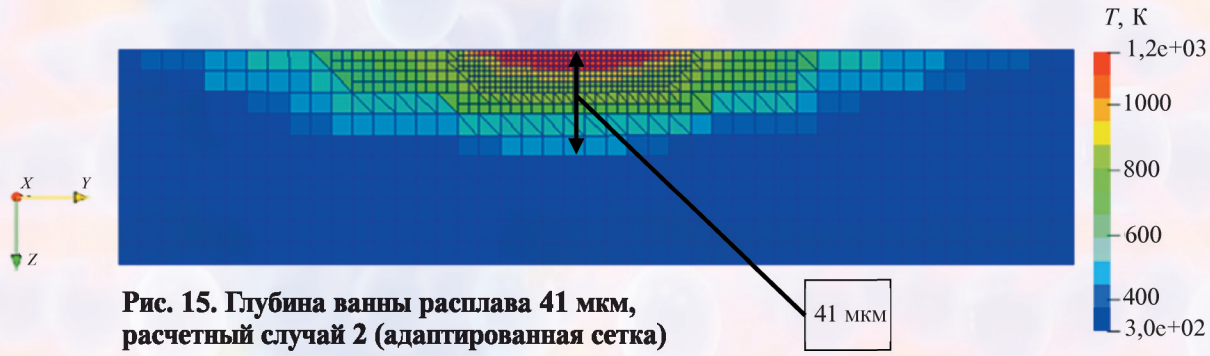

Рис. 15. Глубина ванны расплава 41 мкм, расчетный случай 2 (адаптированная сетка)

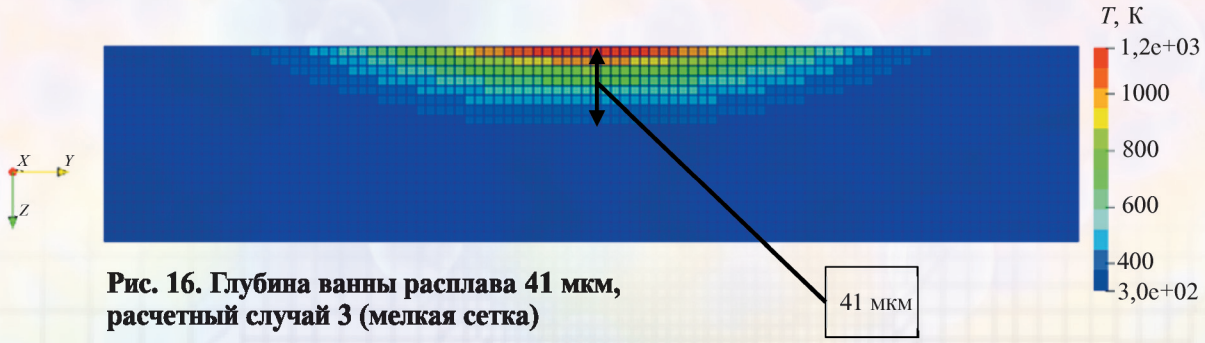

Рис. 16. Глубина ванны расплава 41 мкм, расчетный случай 3 (мелкая сетка)

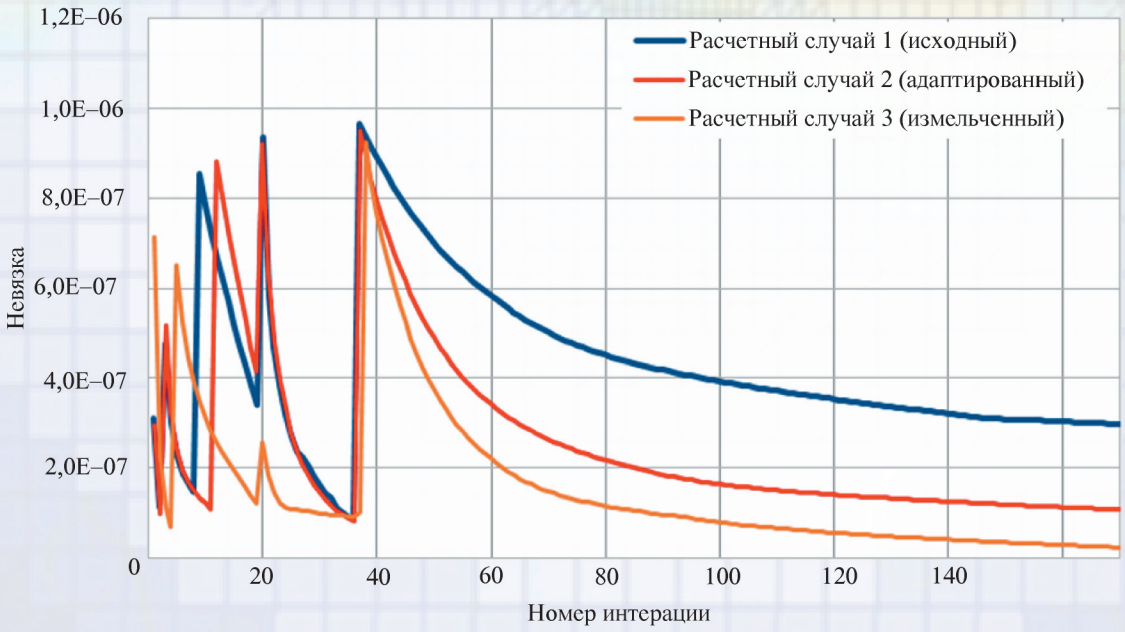

Рис. 17. Графики убывания невязки для расчетных случаев 1, 2, 3

ISSN 2220-3397. Программная инженерия. 2022. Том 13. № 1. 1 – 48. Подписной индекс по Объединенному каталогу "Пресса России" – 22765