МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра дискретной математики и информационных технологий

РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ GO И RUST

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

студента 2 курса 271 группы
направления 09.04.01 — Информатика и вычислительная техника
факультета КНиИТ
Плишкина Александра Вадимовича

Научный руководитель
должность, степень, звание _____ А. Д. Панфёров

Заведующий кафедрой
доцент, к. ф.-м. н. _____ Л. Б. Тяпаев

Саратов 2024

ВВЕДЕНИЕ

С момента появления вычислительных систем на основе электронных компонентов (ЭВМ) прогресс в их производительности является почти синонимом общего технического прогресса нашей цивилизации. Сменялись поколения используемых аппаратных решений, технологии производственного процесса осваивали микроны и нанометры, но взятый с самого начала темп роста производительности сохраняется уже практически три четверти века.

Последние два десятка лет развитие идет по пути роста аппаратного параллелизма. Прежде всего это выражается в росте количества ядер в процессорах. Современный процессор для настольной системы общего назначения может иметь 6, 8, 12 и более ядер. Серверные процессоры могут иметь вплоть до 64 физических ядер и более. Индустрия вернулась к практике создания и использования специализированных вычислительных сопроцессоров, количество ядер в которых исчисляется тысячами. К сожалению, поддержка роста аппаратных возможностей требует и существенных усилий от разработчиков ПО. Необходимо уметь пользоваться этими ресурсами.

При этом аппаратный параллелизм имеет свою достаточно продолжительную и интересную историю. Но в течении длительного времени он использовался в ограниченных масштабах в системах максимально высокой производительности. Для его реализации разрабатывались решения на основе таких классических языков программирования как Fortran и C, воплощаемые в программы специалистами высокого уровня.

С приходом аппаратного параллелизма в массовые системы, аналогичные проблемы приходится решать при разработке практически любого системного или прикладного ПО. Существовавшим к этому моменту языкам программирования пришлось более или менее успешно приспосабливаться к новой параллельной реальности. В качестве альтернативы появился ряд новых языков, которые, по задумке создателей, должны иметь и предоставлять удобные инструменты для написания параллельных программ. Характерными примерами являются Rust и Go - молодые и активно развивающиеся языки программирования. Они оба компилируемые, с строгой системой типизации, и оба делают серьёзный акцент на естественной конкурентности, реализуя различные инструменты, предназначенные для упрощения работы с ней в параллельной среде.

В качестве цели работы было определено изучение особенностей реализации параллелизма в языках программирования Go и Rust и демонстрация используемых для этого инструментов на примере работы с массивами данных численных экспериментов по моделированию физических процессов. Задачи работы:

1. Изучение реализации параллелизма с использованием конкурентности в языке программирования Go.
2. Изучение реализации параллелизма с использованием конкурентности в языке программирования Rust.
3. Демонстрация инструментов распараллеливания языков программирования Go и Rust.
4. Разработка программных решений для обработки данных по заданному сценарию средствами языков программирования Go и Rust.
5. Разработка программных решений для обработки данных по заданному сценарию в параллельном режиме средствами языков программирования Go и Rust.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

В первой главе

В первой главе представлен обзор базовых понятий параллелизма вычислительных процессов, терминология, используемые классификации, причины использования и сферы применения.

Основные понятия параллелизма - процесс, поток, ресурс.

Процесс- одно из основных понятий параллельного программирования. Существующие в литературе определения процесса сводятся к его пониманию как "некоторой последовательности команд, претендующей наравне с другими процессами программы на использование процессора для своего выполнения".

Операционная система воспринимает выполняемые программы как процессы, которые параллельно выполняются, взаимодействуют и конкурируют друг с другом за использование ядер процессора, или процессоров вычислительной системы (аппаратные ресурсы системы). При этом, процесс достаточно "тяжеловеснен"– создание процесса, переключение ядер процессора или самих процессоров на использование других процессов и выполнение других подобных действий занимает достаточно много машинного времени. Также, процессы выполняются в своих, разных адресных пространствах. В результате, организация их параллельного взаимодействия требует сравнительно много ресурсов.

Вышеперечисленные особенности приводят к необходимости введиния в практику программирования потоков (англ. thread) - более простой альтернативы процессу. Поток, как и процесс - последовательность команд программы, которая конкурирует за использование процессора вычислительной системы для своего выполнения. Потоки отличаются от процесса тем, что их группы работают в одном общем адресном пространстве. Соответственно, потоки разделяют данные программы, которая их вызвала. Важно отметить, что общность данных существенно упрощает организацию взаимодействия потоков, так как результат вычисленный одного потока, сразу становится доступен остальным потокам программ. При этом, общность данных требует соблюдения определенных строгих правил использования разделяемых данных.

Ресурсы - любые объекты вычислительной системы, которые могут быть

использованы процессом для своего выполнения. К ресурсам относятся, например, процесс, память, программы, данные и т.п.

Важным понятием,возникшим еще в эпоху последовательных программ, но полезным и удобным для понимания работы параллельного кодя является конкурентность. Конкурентность описывает свойство программы, части которой работают самостоятельно друг от друга. Параллелизм описывает программу, независимые части которой запускаются на разных процессорах. Иными словами, конкурентность - свойство кода вашей программы, параллелизм - свойство работы вашей программы.

Одна из наиболее распространенных классификаций вычислительных систем - систематика Флинна. В рамках этой систематики основное внимание при анализе устройства вычислительных систем уделяется тому, как взаимодействуют последовательности (потоки) выполняемых команд и обрабатываемых данных. По классификации Флинна, системы разделяются на 4 группы:

1. SISD - один поток команд и один поток данных. К этому типу отнесятся обычные последовательные ЭВМ.
2. SIMD (Single Instruction, Multiple Data) - один поток команд и множество потоков данных.
3. MISD (Multiple Instruction, Single Data) - множество потоков команд и один поток данных.
4. MIMD (Multiple Instruction, Multiple Data) - К этому типу относятся системы, использующие множество потоков команд и множество потоков данных.

Ярким примером параллельных систем являются суперкомпьютеры. При их разработке используются самые передовые аппаратные и архитектурные решения. Практически с момента появления этого термина все системы такого класса используют аппаратный параллелизм для достижения максимально возможной производительности. На сегодня абсолютным лидером является система Frontier, эксплуатируемая в Теннеси (США) в национальной лаборатории Oak Ridge. В её составе 8699904 вычислительных ядер (CPU и GPU), что обеспечивает теоретическую пиковую производительность более полутора эксафлопс. Самый производительный суперкомпьютер, эксплуатируемый в России, занимает 42 место в мировом рейтинге. В его составе 193440 ядер (CPU и GPU).

Во второй главе

Во второй главе были рассмотрены различные алгоритмы сортировок и их параллельные реализации. Это было обусловлено поставленной задачей продемонстрировать возможности языков Go и Rust на обработке массивов данных численных экспериментов. Рассматривались три алгоритма сортировки: пузырьковая сортировка, сортировка Шелла, и быстрая сортировка.

Сортировка - одна из типовых задач при обработке данных. Обычно под сортировкой понимается задача размещения элементов неупорядоченного набора данных в порядке возрастания или убывания. Упорядочивание является стравнительно трудоёмкой задачей. Для ряда известных простых методов, например, для пузырьковой суртировки, количество необходимых операций определяется квадратичной зависимостью от числа упорядочиваемых данных $T = n^2$. Для более эффективных алгоритмов, например, для быстрой сортировки, трудоемкость определяется величиной $T = n \log_2 n$

Пузырьковая сортировка - один из наиболее широко известных методов упорядочивания данных, но как правило используется в учебных целях, в связи с его низкой эффиктивностью. Алгоритм пузырьковой сортировки сравнивает и обменивает соседние элементы в последовательности, которую нужно отсортировать. Пузырьковая сортировка в своём прямом виде сложна для распараллеливания, так как сравнение пар значений происходит строго последовательно. Поэтому, при параллельной реализации используется модификация этого алгоритма - метод чет-нечетной перестановки. Модификации состоит в введении в алгоритм двух правил выполнения итераций метода, в зависимости от четности или нечетности номера итерации. Соответственное, при чётной итерации выбираются элементы с четными индексами, при нечётной итерации - нечетными индексами.

Алгоритм сортировки Шелла состоит в сравнении на начальных стадиях сортировки пар значений, находящихся далеко друг от друга в наборе данных, который необходимо отсортировать. На первом шаге алгоритма упорядочиваются элементы $n/2$ пар $(a_i, a_{n/2+i})$ для $1 <= i <= n/2$. На втором шаге упорядочиваются элементы в $n/4$ группах из четырех элементов $(a_i, a_{n/4+i}, a_{n/2+i}, a_{3n/4+i})$ для $1 <= i <= n/4$. Третий шаг - упорядочивание элементов в группах из восьми элементов и т.д. Последнеий шаг - упорядочивание элементов всего набора данных $(a_1, a_2, ..., a_n)$. На каждом шаге эле-

менты в группах упорядовачиваются с использованием метода сортировки вставками. Общее количество итераций алгоритма Шелла является равным $\log_2 n$

Алгоритм быстрой сортировки относится к числу эффективных методов сортировки данных и активно используется в практических задачах. Суть алгоритма быстрой сортировки заключается в делении сортируемый данных на блоки меньшего размера так, что между значениями разных блоков обеспечивается упорядоченность. То есть, при разделении данных на две части, все данные в первой части меньше данных во второй. На первой итерации исходный набор данных разделяется на две части. Для организации деления один из элементов выбирается в качестве ведущего. Все значения набора, меньшие ведущего элемента, переносятся в первый формируемый блок. Элементы со значением больше ведущего, соответственно, переносятся во второй. Далее, описанный алгоритм применяются рекурсивно для обоих сформированных на первом шагов блоков до тех пор, пока данные не будут отсортированы. Если ведущие элементы были выбраны удачно, то данные будут упорядочены после выполнения $\log_2 n$ итераций.

Эффективность быстрой сортировки сильно зависит от того, какой элемент выбирается ведущим при создании блоков. В худшем случае трудоемкость метода такая же, как и у пузырьковаой сортировка - т.е. $T1 = n^2$. В случае, когда ведущие элементы были выбраны оптимально, каждый блок разделяется на равные по размеру части. В этом случае, трудоемкость алгоритма является одной из самых низких, и равна $(T1 = n \log_2 n)$.

Алгоритмы Шелла и быстрой сортировки удобны для параллельной реализации.

В третьей главе

В третьей главе представлены результаты изучения языков программирования Go и Rust, сферы их применения, итзывы и сравнения в литературе, пример решения зач типа SIMD с использованием их инструментов.

Go - компилируемый язык программирования, однако процесс компиляции несколько отличается от более старших языков, таких как C и C++. Go является языком с реализованным сборщиком мусора. Сборка мусора - автоматическая очистка памяти от неиспользуемых элементов. Являясь строго типизированным языком программирования, Go также поддерживает «небез-

опасное» программирование, с использованием пакета «unsafe». Это позволяет, например, создать указатель int32, который указывает на переменную int64.

В Go реализован собственный уникальный, инновационный способ обеспечения конкурентности. API для создания конкурентных программ, встроенный в Go, основывается на модели Communicating Sequential Processes, сокращённо CSP – взаимодействующие последовательные процессы. Использование этой модели позволяет избежать проблем, связанных с блокировками, такими как, например, их приобритение и освобождение. Синхронизация потоков обеспечивается с использованием каналов- путём передачи данных через них.

Дополнительно необходимо отметить, что в Go реализованы и другие способы синхронизации потоков, такие как мьютексы (mutex) и группы ожидания.

Основным примитивом реализации конкурентности в Go является горутина. Она является самой маленькой сущностью языка программирования, которая может самостоятельно выполняться. В Go все выполняется посредством горутин. Важная особенность горутин, которая отличает их от потоков, например, в Rust, это то, что горутина не является потоком операционной системы, и, соответственно, управляется планировщиком Go, а не планировщиком ОС.

Следующий важный примитив конкурентности в Go - каналы. Каналы используются для передачи данных между горутинами, и могут быть использованы для синхронизации. У каналов есть ряд правил. Во-первых, каждый канал позволяет передавать данные только определенного типа. Этот тип данных называется типом элемента канала. Это означает, что через канал созданный для передачи данных типа Int не может передавать данные типа String. Во-вторых, для работы канала необходим как отправитель, так и получатель данных. Ими могут выступать, например, две горутины. При использовании канала в качестве аргумента функции, можно указать его направление — канал может быть использован только для принятия или передачи данны. Изначально, канал может как принимать данные, так и отправлять их.

Go используется в широком спектре задач, таком как написание ней-

ронных систем, создание микросервисов, и системное программирование.

Rust - компилируемый язык программирования, вышедший в 2010 году (первая стабильная версия - в 2015). Rust позиционируется как "язык, предоставляющий возможность писать эффективный код и комфортный уровень абстракции". Он должен улучшить подход других языков к типобезопасности, параллелизму, и доступу к ресурсам. Rust не является настоящим объектно-ориентированным языком, несмотря на наличие некоторых черт, типичных для таких языков программирования. При этом, Rust так же не является функциональным языком, хотя стремится сделать результат вычисления более явно выраженным. Rust, в определённой степени, похож на C и C++, но между ними достаточно различий, чтобы многие идиомы этих языков были неприминимы к Rust. В результате, типичный код на Rust лишь поверхностно схож с кодом на более старших языках программирования, таких как C или C++.

Важным элементом работы с памятью в Rust является система владения и заимствования. Правила владения:

1. Каждое значение в языке Rust имеет переменную, которая называется его владельцем.
2. В каждый момент времени может существовать только один владелец.
3. Если владелец выйдет из области видимости, значение будет отброшено.

В Rust семантика передачи значения применятеся почти во всех случаях с использованием значений. При передаче аргументов в функцию передается владение ее параметрами. Соответственно, при возврате значения из функции владение переходит вызывающей стороне. При построении кортежа владение значениями переходит к кортежу, и т.д.

Безопасная и эффективная работа с конкурентностью в программировании является одной из главных целей языка Rust. Rust реализована модель 1:1 - один поток языка на один поток ОС. Благодаря правилам владения, помогающим при работе с памятью в Rust, при доступе потоков к общей памяти невозможны гонки за данные.

Так же как и в выше описанном Go, в Rust реализованы каналы. Когда в канал отправляется значение, происходит передача владения, а не копирование данных. Процедура передачи владения происходит значительно быстрее. В Rust существует два типа каналов - синхронные и асинхронные.

Язык программирования Rust может применяться в большом количестве видов приложений, но в первую очередь он предназначен для системного программирования - операционные системы, драйвера, и програмное обеспечение, предназначенное для работы в условиях ограничения в ресурсах. Наиболее интересным моментом, касающимся использования Rust в системном программировании является его включение в ядро Linux - до этого, для написания ядра разрешался только оригинальный C.

В литературе, изучающей языки программирования, Go и Rust чаще сравниваются с более старшими языками программирования, и реже напрямую между собой. Обычно принимается, что Go показывает производительность схожую с C, Rust показывает ещё более высокую производительность.

В качестве демонстрационного примера задачи, которая будет решена с использованием конкуретности, было выбрано численное интегрирование одномерного определенного интеграла от аналитически заданной функции. Для решения этой задачи были написаны две аналогично работающие программы на изучаемых языках программирования.

Результаты работы программы на Rust:

1. Один поток: 1257 милисекунд,
2. Два потока: 802 милисекунд.
3. Четыре потока: 447 милисекунд.
4. Восемь потоков: 335 милисекунд.

Результаты работы программы на Go:

1. Один поток: 1701 милисекунд,
2. Два потока: 847 милисекунд.
3. Четыре потока: 652 милисекунд.
4. Восемь потоков: 389 милисекунд.

Можно заметить, что Rust справляется с задачей быстрее.

В четвёртой главе

В четвёртой главе описывается решение задачи обработки результатов численных экспериментов с моделями, описывающими поведение двумерного материала во внешнем электрическом поле с заданными параметрами. Для этого были независимо разработаны программы на Go и Rust. Для выполнения тестирования были предоставлены примеры с наборами результатов моделирования с данными из реальных численных экспериментов объёмом

от $\sim 10^6$ до $\sim 10^7$ строк.

Тесты выполнялись на системе с характеристиками: Процессор Intel Core I5 8265u (4 ядра) и 8 гигабайт ОЗУ. В качестве ОС использовался Endeavour Os семейста Linux.

В силу сравнительной простоты реализации распараллеливания для сортировки данных был выбран алгоритм быстрой сортировки - каждый подмассив обрабатывается отдельным потоком. В этом случае нет перечечений по данным и не создаются ситуации гонки. Алгоритм обработки данных одинаков:

1. Файл считывается в память.
2. Значения сортируются. Для синхронизации используются каналы - каждый новый поток передаёт в поток его вызвавший сообщение о том, что он закончил.

Было разработано и протестировано несколько версий программ. Так, на языке программирования Rust была разработана программа решающая задачу сортировки последовательно. Из представленных на Рис. 1 результатов её работы видно, что программа очень хорошо справляется с ростом объёма задачи при увеличении размера файла.

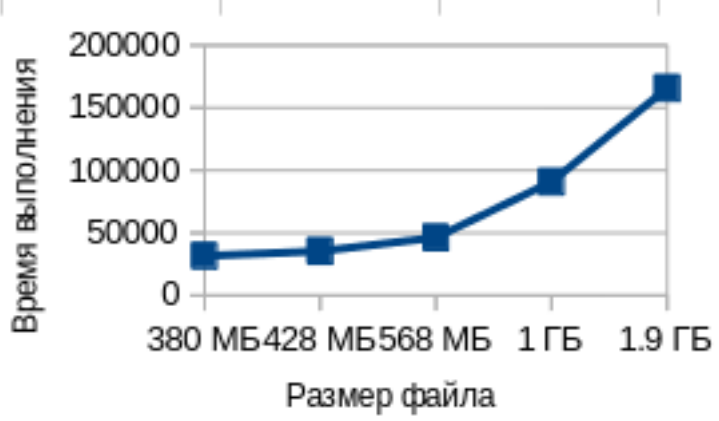

Рисунок 1. Зависимость времени работы одногопоточной программы на Rust, от размера файла данных

На языке программирования Go были реализованы программы, решающие аналогичную задачу в последовательном и в параллельном режиме. Результаты тестирования работы последовательной версии показаны на Рис. 3. Она также хорошо реагирует на увеличение сортируемого файла. Необходимо отметить, что реализация на Go оказалась более быстрой при том, что на тестовых задачах типа SIMD более быстрым был Rust.

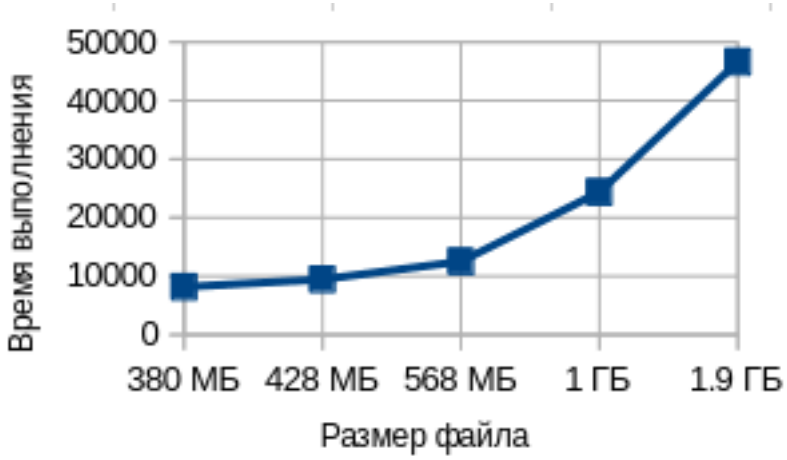

Рисунок 2. Зависимость времени работы одногопоточной программы на Go, от размера файла данных

Параллельная версия программы, тестировавшаяся на системе с 4 физическими и 8 логическими ядрами, смогла показать ускорение только около 25% – 30%. При работе параллельной версии программы с файлом в 1.9 ГБ возникал дефицит памяти и полученные в этом случае результаты не репрезентативны.

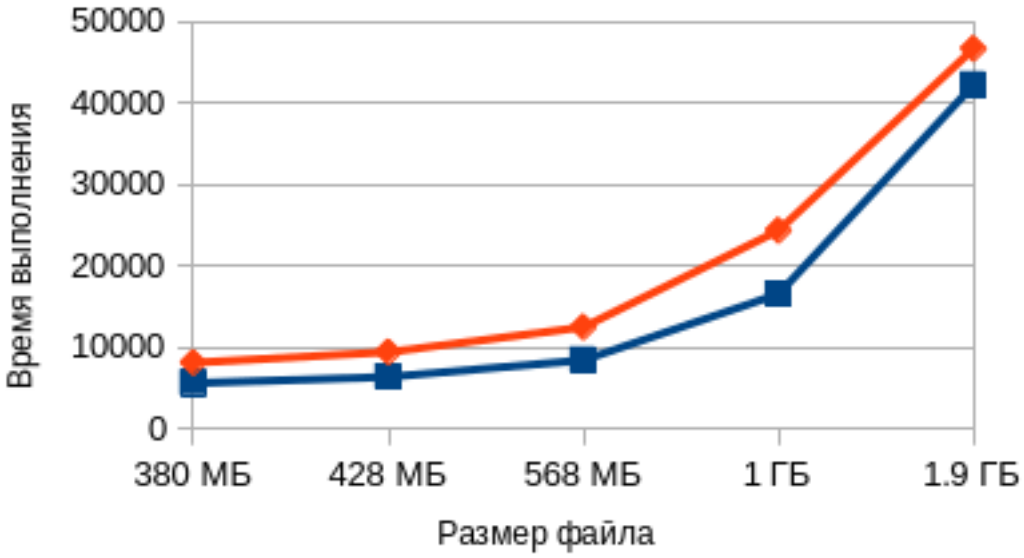

Рисунок 3. Зависимости времени работы многопоточной и одногопоточной программ на Go, от размера файла данных

ЗАКЛЮЧЕНИЕ

Поставленная цель по изучению особенностей реализации параллелизма в языках программирования Go и Rust была достигнута. Выполнены демонстрации на примере тестовых задач и работы с массивами данных численных экспериментов по моделированию физических процессов.

Реализация параллелизма в изучавшихся языках прпограммирования основана на автоматизации управления конкурентными ветвями кода. При этом механизмы реализаци такого управления различны. В ходе работы продемонстрировано, что для задач с явным параллелизмом по данным оба языка достаточно успешно обеспечивают использование параллельных аппаратных ресурсов. Обработка данных с более сложными сценариями требует аккуратной реализации с учетом различных факторов, влияющих на результат. Для последних в представленных решениях не удалось обеспечить хорошую масштабируемость и эта проблема требует дополнительного исследования.

Проделанная работа по реализации поставленных задач предоставила большой объём фактического материала по особенностям и возможностям новых перспективных языков программирования. Для их успешного использования необходимо глубокое понимание принципов работы инструментов и их реальных возможностей. Полученные результаты могут служить основой для формулировки новых тем будущих исследований.

Основные источники информации:

1 Гергель В, П. ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ВЫЧИСЛЕНИЯ ДЛЯ МНОГОЯДЕРНЫХ МНОГОПРОЦЕССОРНЫХ СИСТЕМ / П. Гергель, В. — Нижний Новгород: Издательство Нижегородского госуниверситета, 2010.

2 Cox-Buday, K. Concurrency in Go / K. Cox-Buday. — First edition edition. — Beijing: O'Reilly, 2017. — Includes bibliographical references and index.

3 Serdar, B. Effective Concurrency in Go / B. Serdar. — 1. edition. — Birmingham: Packt Publishing Limited, 2023.

4 Цукалос, М. Golang для профи: работа с сетью, многопоточность, структуры данных и машинное обучение с Go. / М. Цукалос. — Санкт- Петербург: Питер, 2020

5 Guerrieri, A. Hands-On System Programming with Go Build Modern and Concurrent Applications for Unix and Linux Systems Using Golang / A.

Guer- rieri. — Birmingham: Packt Publishing, Limited, 2019
6 Ставонин, А. Ключевые возможности rust / А. Ставонин // RSDN Magazine. — 2013. — No 1. — С. 23–28
7 Джим Блэнди, Программирование на языке Rust. / Джим Блэнди. — Москва: ДМК Пресс, 2018
8 Клабник Стив, Программирование на Rust. / Клабник Стив. — Санкт-Петербург: Питер, 2021.
9 Bos, M. Rust Atomics and Locks / M. Bos. — Beijing: O'Reilly, 2023. — Description based on publisher supplied metadata and other sources.
10 McNamara, T. Rust in Action / T. McNamara. — Shelter Island, NY: Manning Publications, 2021.