ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА

Троицкий авиационный технический колледж – филиал федерального государственного бюджетного образовательного учреждения высшего образования «Московский государственный технический университет гражданской авиации»

Отделение «Программирование в компьютерных системах»

**КУРСОВАЯ РАБОТА**

**Модели параллельных программ и методы планирования параллельных вычислений**

Выполнил курсант 432 группы 4 курса очной формы обучения

Специальность 09.02.03 «Программирование в компьютерных системах»

Стасенко Константин Юрьевич

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2022 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Руководитель курсовой работы: Черевкова Ольга Алексеевна

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2022 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Дата сдачи: \_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Троицк, 2022г.

Троицкий авиационный технический колледж – филиал

федерального государственного бюджетного образовательного

учреждения высшего профессионального образования

«Московский государственный технический университет

гражданской авиации»

УТВЕРЖДАЮ

Зам. директора по УР

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« \_\_\_ » \_\_\_\_\_\_\_\_ 2022 г

ЗАДАНИЕ

на выполнение курсовой работы

Курсант *432* группы *Стасенко Константин Юрьевич*

Специальность *09.02.03 «Программирование в компьютерных системах»*

Тема работы *Модели параллельных программ и методы планирования параллельных вычислений*

Утверждена приказом по колледжу *№ 150 / КС от 10 декабря 2021 г.*

Сроки сдачи курсантом законченной работы *« 3 » апреля 2022 г.*

1. Исходные данные к работе:

*литература по теме курсовой работы, техническое задание на создание базы данных.*

1. Перечень вопросов подлежащих разработке:

*анализ литературы по теме курсовой работы; постановка цели и задач работы; выбор объекта и предмета исследования; методы и средства распараллеливания вычислений; разработка примеров и моделей параллелизма; оформление пояснительной записки.*

Календарный график:

|  |  |  |
| --- | --- | --- |
| **Наименование этапов дипломной работы** | **Срок выполнения** | **Отметка о выполнении** |
| Подбор литературы и ознакомление с ее содержанием. | 23.11.21г. |  |
| Изучение теоретических материалов, составление развернутого плана, написание раздела «Введение» (2 и более страницы), титульный лист, содержание. | 25.12.21г. |  |
| **Раздел 1. Теоретическая часть** (всего 10-13 страниц), список литературы. | 22.02.22г. |  |
| **Раздел 2. Практическая часть** (всего 11-13 страниц). | 17.03.22г. |  |
| Оформление курсовой работы:  заключение (2 и более страницы);  список литературы, аннотация, приложения.  Проверка работы руководителем и исправление замечаний. | 3.04.22г. |  |
| Защита курсовой работы | 6.04.22г. |  |

Задание принял к исполнению\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Дата выдачи

Срок окончания 3.04.2022г.

Руководитель Черевкова О.А.

(подпись)

**АННОТАЦИЯ**

Данная курсовая работа на тему «Модели параллельных программ и методы планирования параллельных вычислений» выполнена курсантом гр. 432 Стасенко Константином Юрьевичем, специальность 09.02.03 «Программирование в компьютерных системах», 2022 год.

В курсовой работе представлены: теоретические основы параллелизма, модели и виды параллелизма включая их примеры; сравнение параллелизма с конкурентностью и распределенностью; основы планирования параллельных вычислений; применение параллелизма к базам данных. Сформулированы цель и задачи работы. Разработаны программы, показывающие отличительные черты каждого вида параллелизма.

Общее число страниц – 51, рисунков – 15, таблиц – 8, литературных источников – 10.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись курсанта)

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ……………………….……………….…………..………………..……4

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Понятие параллелизма……………………….……………….…………..4

1.2 Закон Мура……………………….……………….……………………….4

1.3 Отличия параллелизма от конкурентности……………………...……....4

1.4 Модели и виды параллелизма……………………….……………….…..4

1.5 Планирование параллельных вычислений (scheduling)………………...4

1.6 Параллелизм в системах баз данных…………………………….………4

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Программные средства разработки……………………….……………..4

2.2 Составление примеров.…………………….……………….…………….4

2.3 Разработка БД………...…………………….……………….…………….4

ЗАКЛЮЧЕНИЕ……………………….……………….……………………………..4

СПИСОК ЛИТЕРАТУРЫ……………………….……………….…………………..4

ПРИЛОЖЕНИЕ А……………………….……………….…………………………..4

**ВВЕДЕНИЕ**

Сегодня, когда закон Мура уже перестал действовать, полагаться разработчикам на то что со временем их программы сами станут работать быстрее за счет увеличения мощности процессоров уже не стоит [3]. Одним из способов программного увеличения производительности стал параллелизм. Раньше он применялся в основном в узкоспециализированных областях, например таких, как телефония. В 1986 году в компании Ericsson был разработан язык Erlang позволяющий создавать десятки и сотен тысяч зеленых потоков, так называемые акторы, запускаемые параллельно на нескольких машинах могли обрабатывать большое количество телефонных звонков.

Параллелизм поначалу интересовавший только отдельных исследователей из академических кругов и узконаправленных специалистов со временем прочно вошел в обиход программистов. Сперва пытались обойтись параллелизмом инструкций — компилятор автоматически распределяет выполнение машинных команд по ядрам процессора, но и здесь достигнув своего потолка большинство создателей языков программирования обратили взор на академические исследования. Теперь каждый уважающий себя язык имеет как минимум возможность создания потоков операционной системы. Но управляться с ними на самом деле не особо удобно, отдельный поток мог занимать десятки мегабайт, поэтому стали развиваться в направлении зеленых потоков, например таких, как в Erlang.

Довольно любопытным последствием подобного ажиотажа вокруг параллелизма оказался подъем интереса к функциональному программированию. Проблемой привычной модели процедурного программирования оказался параллельный доступ к разделяемой памяти (shared memory) — когда два потока могли одновременно читать и писать в одну и ту же область памяти. Для решения этой проблемы сперва придумали мьютексы и семафоры — специальные средства позволяющие заблокировать доступ к памяти пока один из потоков пишет в нее [5]. Это оказались не очень удобные инструменты, человек часто совершал ошибки при работе с ними. Функциональное же программирование решило эту проблему довольно радикальным способом — в принципе избавилось от изменяемого состояния [3, 4].

Несмотря на сложность программирования параллельных программ в традиционном стиле, так можно выразить практически что угодно, в то время как более простые способы позволяют обойтись меньшей сложностью при разработке, но предоставляют свои ограничения из-за чего применимы в несколько более узком круге задач.

Многолетнее развитие подобных инструментов, прекращение действия закона Мура и усложнение программных систем приводит нас к тому что разработчикам этих самых систем, а также тем кто их поддерживает и сопровождает, необходимо хотя бы минимальное знание о видах и моделях параллелизма для более осознанного подхода к выбору инструментов для решения той или иной задачи возникающих перед ними.

*Проблема исследования* — заключается в потребности увеличения производительности, надежности и отзывчивости программ за счет введения в них параллелизма.

*Актуальность* — заключается в постоянном росте масштабов и сложности современных программных продуктов которые необходимо контролировать соответствующими инструментами.

*Цель исследования* — показать преимущества и недостатки различных моделей параллелизма, провести ознакомительную работу с целью повышения уровня технической грамотности учащихся ТАТК ГА.

*Задачи:*

1. Исследовать имеющиеся материалы в сфере информатики
2. Исследовать имеющиеся материалы по теме параллелизма
3. Разработать наглядные примеры программ и баз данных для демонстрации параллелизма.
4. Анализ выполненной работ

*Объект исследования —* компьютерные науки или информатика.

*Предмет исследования* — виды параллелизма и параллелизм баз данных.

*Методы исследования* — способ получения знаний и навыков, расширения теоретической базы. Именно они помогут получить или доказать тезисы данной курсовой работы.

Теоретические методы:

* абстрагирование;
* аналогия;
* классификация;
* обобщение;
* сравнительный анализ;
* синтез (объединение);
* изучение и анализ литературы;
* изучение и анализ документации, архивных источников и т. д.

Практические методы:

* эксперименты;
* наблюдение;
* расчеты, измерения;
* моделирование;
* беседа или интервью;
* опрос;
* описание и т.д.

1. **теоретическая часть**
   1. **Понятие параллелизма**

*Параллелизм* — свойство систем, при котором несколько вычислений выполняются одновременно, и при этом, возможно, взаимодействуют друг с другом. Вычисления могут выполняться на нескольких ядрах одного процессора, либо выполняться на физически отдельных процессорах. Для выполнения параллельных вычислений разработаны ряд математических моделей, некоторые из них будут рассмотрены далее. Параллельная программа эксплуатирует множественность вычислительного оборудования для более быстрого решения вычислительных задач. Цель состоит в сокращении времени получения ответа.

Поскольку вычисления в параллельных системах взаимодействуют друг с другом, число возможных путей выполнения может быть чрезвычайно велико, и результирующий итог может стать недетерминированным. Параллельное использование общих ресурсов может стать одним из источников недетерминированности, приводящей к таким проблемам, как взаимная блокировка или ресурсное голодание.

Один из методов внедрения параллелизма — конвейер инструкций, или конвейеризация [5]. При этом методе исключен человеческий фактор так как он реализуется на аппаратном уровне, то есть не зависит от прикладного программиста, и представляет собой способ организации вычислений, используемый в процессорах и контроллерах с целью повышения их производительности. Конвейерный процессор принимает новую инструкцию каждый цикл, даже предыдущие инструкции не завершены. В результате выполнение нескольких инструкций перекрывается и в процессоре находятся сразу несколько инструкций в разной степени готовности.

* 1. **Закон Мура**

Сейчас, когда производители процессоров по большей части сдались в своем стремлении увеличивать быстродействие отдельных процессоров и взамен этого сфокусировались на предоставлении пользователям большего их числа, наибольший прирост производительности программ приходится достигать с помощью приемов параллельного программирования, использующих дополнительные вычислительные ядра. Это происходит из-за прекращения действия закона Мура — эмпирическое наблюдение, изначально сделанное Гордном Муром, согласно которому количество транзисторов, размещаемых на кристалле интегральной схема, удваивается каждые два года что позволяло до определенного момента увеличивать тактовую частоту [12].

Помимо закона Мура существует также закон Амдала который иллюстрирует ограничение роста производительности вычислительной системы с увеличением количества вычислителей [13]. Оно описывает ограничение на рост производительности при распараллеливании вычислений: «В случае, когда задача разделяется на несколько частей, суммарное время ее выполнения на параллельной системе не может быть меньше времени выполнения самого медленного фрагмента». Согласно этому закону, ускорение выполнения программы за счет распараллеливания ее инструкций на множестве вычислителей ограничено временем, необходимым для выполнения ее последовательных инструкций. Этот закон показывает, что прирост эффективности вычислений зависит от алгоритма задачи, т. е. не для всякой задачи имеет смысл наращивание числа процессоров в вычислительной системе. Более того, с определенного момента добавление новых узлов в систему будет увеличивать время расчета задачи. Тем не менее это не умаляет важности и нужности знания и использования механизмов распараллеливания вычислений.

* 1. **Отличия параллелизма от конкурентности**

Во многих областях знаний термины параллельный, конкурентный и распределенный являются синонимами, однако в программировании это не так. Концепции, которым они соответствуют, совершенно разные.

Конкурентность — прием структурирования программ, в которых имеются множественные потоки управления [3]. Концептуально потоки управления выполняются «одновременно», то есть пользователь может наблюдать переплетение эффектов, возникающих в результате их деятельности. Действительно ли они выполняются одновременно или же нет — деталь реализации, конкурентная программа вполне может исполняться как на одном процессоре с переключением контекстов, так и на множестве физических процессоров. Если параллельное программирование заботиться только об эффективности, то конкурентное ориентируется на структурирование программы, взаимодействующей со множеством независимых внешних агентов (таких как пользователь, сервер баз данных и пр.). Конкурентность способствует модульности таких программ: поток, общающийся с пользователем отделяется от потока, работающего с базой данных. В отсутствие конкурентности такие программы пришлось бы писать посредством циклов обработки событий и и функций обратного вызова, что обычно приводит к крайне запутанным монолитным программам.

* 1. **Модели и виды параллелизма**

*Процессы и потоки ОС*

Все современные компьютеры могут делать одновременно

несколько дел. В многозадачной системе процессор переключается между программами, предоставляя каждой от десятков до сотен миллисекунд персонального времени. Следить за работой параллельно выполняющихся процессов достаточно трудно, поэтому со временем разработчики операционных систем разработали концептуальную модель логически упорядоченных процессов, упрощающую эту работу [16].

Процесс — выполняемая в данный момент программа, вместе с текущими значениями счетчика команд, регистров и переменных. У каждого процесса есть собственный виртуальный центральный процессор выполняющий все команды процесса последовательно.

Многозадачность — переключение реального процессора с процесса на процесс. Обеспечивает возможность параллельной обработки нескольких задач.

Каждый процесс может находиться в одном из трех состояний:

1. Активный — в этот конкретный момент использующий процессор
2. Готовый к работе — процесс временно приостановлен, чтобы позволить выполняться другому процессу
3. Заблокированный — процесс не может быть запущен прежде, чем произойдет некое внешнее событие

Управление процессами вручную довольно сложная и трудоемкая задача (см. Приложение 1).

*Акторы*

Акторы — программная сущность моделирующая объекты реального мира. Каждый актор обладает собственной изолированной памятью которую может изменять только он. Чтобы взаимодействовать друг с другом акторы посылают сообщения. Джо Армстронг — пионер компьютерных наук и параллельных вычислений демонстрировал это на примере общения людей [17]:

Сью: Привет, Билл, мой номер телефона — 45 67 89 12

Сью: Ты меня понял?

Билл: Конечно, твой номер телефона — 45 67 89 12

Подобные паттерны взаимодействия очень хорошо известны человечеству — люди функционируют, как независимые сущности, которые общаются с помощью посылки друг другу сообщений. Программы на Erlang (и на других языках с акторами/зелеными процессами) состоят из десятков, сотен, а иногда и сотен тысяч маленьких процессов. И все они работают не завися напрямую друг от друга общаясь между собой путем посылки друг другу сообщений. Они ведут себя как громадное сборище людей, которые общаются друг с другом.

Это позволяет значительно проще управлять и масштабировать программы на Erlang (см. Приложение 2). Предположим у нас есть десять людей (процессов) и слишком много работы, которую они должны сделать. В этой ситуации можно позвать на помощь больше людей. Чтобы управлять этими группами надо просто рассказать им, всем сразу, их инструкции (широковещательное сообщение от одного ко многим).

Для обработки ошибок в Erlang используется принцип: «Пускай упадет» (Let it crash). Кратко его можно описать следующим образом — представьте себе комнату полную людей. Неожиданно кто-то из них падает и умирает. И в самый момент своей смерти он произносит: «Я умер от сердечного приступа» или «Я умер от разрыва слепого отростка желудка». Также поступают и процессы Erlang. Один процесс, умирая, может сказать «Я умер, потому что меня попросили разделить на ноль», а другой, возможно, скажет «Я умер, потому что меня спросили каков последний элемент в пустом списке». И в этой комнате полной людей, есть специально назначенные люди, работа которых заключается в том, чтобы позаботиться о трупах. Так работает механизм ошибок в Erlang. Процессы в нем могут быть связаны вместе. И если один из них умирает, то другие получат сообщение об ошибке, говорящее от том что этот процесс умер.

Рич Хикки (независимый разработчик ПО и консультант с 20-летним опытом работы в различных областях разработки ПО, создатель языка программирования Clojure) объяснял свое нежелание внедрять модель акторов в качестве одной из основ параллелизма в своем языке тем что акторы это гораздо более сложная модель программирования в малом масштабе, требующая 2-х сообщений для простейшего чтения данных и заставляющая использовать блокировку получения сообщений, что создает потенциал для взаимоблокировки. Конечно, Erlang предоставляет дополнительные механизмы, такие как тайм-ауты и деревья наблюдения, для обработки этих сбоев. В контексте отказоустойчивых компонентов и распределенных систем эти механизмы очень полезны для создания надежных систем, но они не требуются для многопроцессорной обработки с разделяемой памятью [18].

*CSP*

Взаимодействующие последовательные процессы (англ. communicating sequential processes, CSP) — формальный язык для описания моделей взаимодействия в параллельных системах. Относится к математическим теориям параллелизма, известных как исчисление процессов (или алгебра процессов), основанных на передаче сообщений по каналам. Оказал влияние на разработку языков Occam, Limbo, Go и Clojure. Теория CSP была впервые описана в статье Энтони Хоара (сэр Чарльз Энтони Ричард Хоар — английский ученый, получил множество наград, в том числе Премию Тьюринга) в 1978 году. Первоначальная версия была неудачной, но впоследствии под влиянием идей, заимствованных из модели акторов теория была значительно развита и остается предметом активных исследований.

CSP занимается рассмотрением любой индивидуальной логики требующей какого-либо ввода и вычисляющей некий вывод — процессами. В Go они представлены при помощи горутин и каналов. Горутины выполняются конкурентно и могут автоматически распараллеливаться с помощью планировщика горутин распределяющего их по доступным ядрам процессора. Они могут синхронизировать друг с другом свой ввод-вывод отправляя или получая данные идущие по каналам.

В общем случае CSP описывает примерно то же что и акторы — всевозможные независимые объекты и процессы взаимодействующие друг с другом посредством отправки и принятия сообщений, но акцент сделан не на самих объектах, а на способах их коммуникации (см. Приложение 3). Такой подход показал себя более гибким и применимым в мелком масштабе что отразилось на сфере применения того же Go — микросервисы и облачные вычисления.

*Параллелизм данных*

Основная идея подхода, основанного на параллелизме данных, заключается параллельной обработки массивов данных т. е. одна операция выполняется сразу над всеми элементами массива. Различные фрагменты такого массива обрабатываются на векторном процессоре или на разных процессорах параллельной машины. Распределением между процессорами занимается программа. Векторизация или распараллеливание в этом случае чаще всего выполняется уже на этапе компиляции — перевода исходного текста программы в машинные команды. Роль программиста в этом случае обычно сводится к заданию настроек векторной или параллельной оптимизации компилятору, директив параллельной компиляции, использованию специализированных языков для параллельных вычислений.

Цель библиотек параллелизма данных состоит в том, чтобы упростить добавление параллелизма в последовательный код, а именно взять существующий код для циклов или итераторов и с минимальными усилиями заставить их работать параллельно (см. Приложение 4). К примеру при создании итераторов в Rust при использовании библиотеки Rayon достаточно будет заменить использование метода iter() на соответствующий библиотечный метод par\_iter() для того что распараллелить ваш код [19]. Параллельные итераторы в данном случае всего лишь абстракция поверх более фундаментального примитива join — принимающего два замыкания и *потенциально* выполняющего их параллельно. Этот фактор является ключевым: решение о том, следует ли использовать параллельные потоки, принимается динамически, основываясь на том, доступны ли простаивающие ядра. Идея состоит в том, что вы можете указывать вызовами join места где параллелизм может быть хорошей идеей, и пусть среда выполнения решает когда действительно нужно воспользоваться этим.

Такой подход является, по сути, ключевым моментом различия между подходом Rayon и потоками. В то время как в других моделях, когда вы работаете с инструментами параллелизма, они всегда стараются выполнить ваш код параллельно, вызов join напротив не обязательно означает параллельное выполнение. Это не только упрощает пользовательский интерфейс, но и может повысить эффективность выполнения. Это связано с тем, что знание того, когда параллелизм выгоден, трудно предсказать заранее, и всегда требует определенного количества глобального контекста.

Правда у потенциального параллелизма есть и свои ограничения, например, попытка использовать канал для связи между двумя замыканиями в join скорее всего зайдет в тупик. Правильный способ думать о join заключается в том, что это подсказка по распараллеливанию последовательного алгоритма. Иногда это не то что нужно, некоторые алгоритмы по своей сути параллельны.

* 1. **Планирование параллельных вычислений (scheduling)**

Планирование или построение расписания (scheduling) является наиболее важной задачей, решаемой при реализации параллельных вычислительных процессов, так как ее результаты в значительной степени влияют на конечные параметры проектируемой или эксплуатируемой информационно-вычислительной системы. Современные методы последовательно-параллельного планирования синтезируют план, вводя шаги управления и состояния в поведенческое описание, распределяют операторы и операции по шагам с целью их параллельного выполнения в пределах одного шага, выполняют минимизацию числа шагов или объема вычислительных ресурсов, необходимых для обеспечения установленного уровня параллелизма [8].

Время выполнения параллельных программ зависит от разделения на единицы вычислений (процессы, задачи) и их планирования (см. рисунок 1). Абсцисса показывает средний размер процесса, ордината — время параллельного выполнения, нормализованное для идеального ускорения. Линии построены с предположением запуска программы на компьютере имеющем 10 процессоров. Эти линии не соответствуют какой-либо конкретной программе, а просто указывают на общую тенденцию. Пунктирная линия представляет идеальное время параллельного выполнения программы разделенной программы без учета накладных расходов. По мере увеличения размера задачи с 10 тактов до 105 тактов, нормализованное идеальное время параллельного выполнения возрастает с 1 до 10 из-за потери параллелизма. Линия состоящая из точек указывает на фактор накладных расходов который снижается с 101.0 до 1.01 по мере увеличения размера задачи с 10 до 105. Сплошная линия указывает на реальное время параллельного выполнения, полученное путем умножения идеального времени параллельного выполнения на фактор накладных расходов [1]. Эта линия иллюстрирует два важным момента:

1. Наличие накладных расходов может сделать невозможным достижение идеального ускорения;
2. Реальное время параллельного выполнения минимизируется при оптимальном промежуточном уровне разделения. Проблема разделения заключается в том чтобы найти соответствующие оптимальные промежуточные оптимальные разделения.

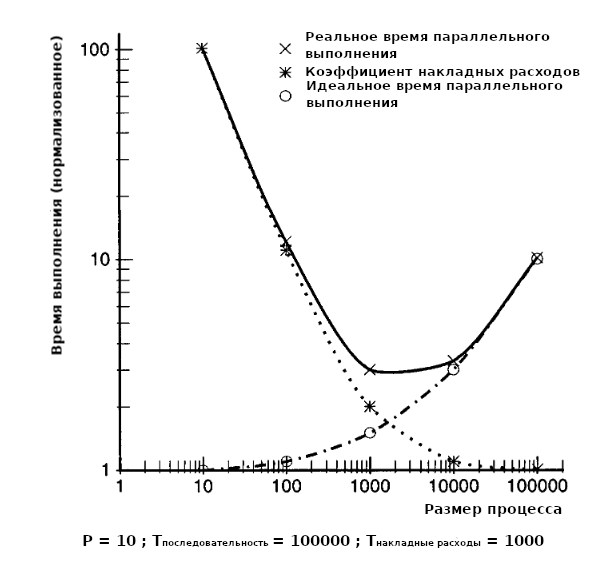


Рисунок 1.1 Компромисс между параллелизмом и накладными расходами

Проблема планирования заключается в назначении задач в разделенной программе на процессоры, чтобы минимизировать время параллельного выполнения. Время параллельного выполнения зависит от использования процессора и накладных расходов на межпроцессорное взаимодействие. Как и проблема разделения, проблема планирования также включает в себя компромисс между параллелизмом и накладными расходами. Параллелизм требует того чтобы процессы были как можно сильнее распределены по разным процессорам. Но накладные расходы на взаимодействие уменьшаются когда процессы назначены одному и тому же процессору. Этот компромисс делает задачу планирования параллельных процессов труднее чем раньше. Всевозможные алгоритмы планирования не могут обеспечить константной производительности из-за накладных расходов на взаимодействие и, в таком случае, проблема планирования является NP-полной даже при условии наличия бесконечного числа доступных процессоров.

* 1. **Параллелизм в базах данных**

В современном мире перед человечеством встал вопрос об обработке больших и сверхбольших данных измеряемых десятками и сотнями террабайт. Фактически единственным эффективным решением этого вопроса является использование параллельных систем баз данных, обеспечивающих параллельную обработку запросов на многопроцессорных вычислительных системах.

Для предоставления подобных возможностей существует массово-параллельная архитектура (MPP - Massively parallel processing) чья особенность заключается в физически разделенной памяти [20]. Система с такой архитектурой строится их отдельных узлов, содержащих процессор, локальный банк оперативной памяти, коммуникационные процессоры или сетевые адаптеры, иногда — жесткие диски и другие устройства ввода-вывода. Доступ к банку оперативной памяти данного узла имеют только процессоры из этого же узла. Узлы соединяются специальными коммуникационными каналами. Главным преимуществом систем с раздельной памятью является хорошая масштабируемость, по тем же причинам что и систем построенных на основе акторов. Каждый процессор имеет доступ только к своей локальной памяти, в связи с чем не возникает необходимости в потактовой синхронизации процессоров. Минусы такого подхода аналогичны минусам акторной модели.

К примеру логическая архитектура параллельной СУБД Teradata представлена следующим образом (см. рисунок 2):

* PE — Parsing Engine, отвечает за контроль сессии и обработку запросов пользователя;
* AMP — Access Module Processor, отвечает за извлечение данных с ассоциированного с ним диска;
* BYNET — Среда обмена сообщениями между компонентами системы.

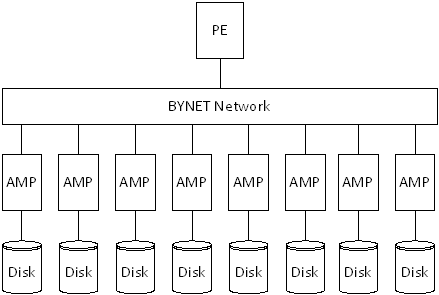


Рисунок 1.2 Логическая архитектура СУБД Teradata

Teradata относится к классу систем MPP, и имеет архитектуру без разделения ресурсов, в рамках которой отдельные узлы системы не имеют общих ресурсов [11].

1. **Практическая часть**
   1. **Программные средства разработки**

*Java*

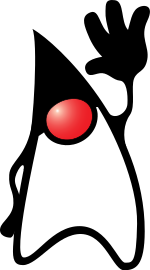
Java — компилируемый статически типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems [21]. Приложения написанные на Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Рисунок 2.1 Дюк — талисман Java

Часто к недостаткам концепции виртуальной машины относят снижение производительности. Ряд усовершенствований несколько увеличил скорость выполнения программ на Java:

* применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-компиляция) с возможностью сохранения версий класса в машинном коде;
* обширное использование платформенно-ориентированного кода в стандартных библиотеках;
* аппаратные средства, обеспечивающие ускоренную обработку байт-кода.

*Rust*

Rust (англ. rust — ржавчина) — мультипарадигмальный компилируемый язык программирования общего назначения, сочетает парадигмы функционального и процедурного программирования с объектной системой, основанной на типажах [22]. Управление памятью осуществляется полуавтоматическим путем через механизм владения основанный на использовании аффиных типах (разновидность линейных типов), что позволяет обходиться без системы сборки мусора. Гарантирует безопасную работу с памятью благодаря встроенной в компилятор системе статической валидации ссылок (borrow checker).

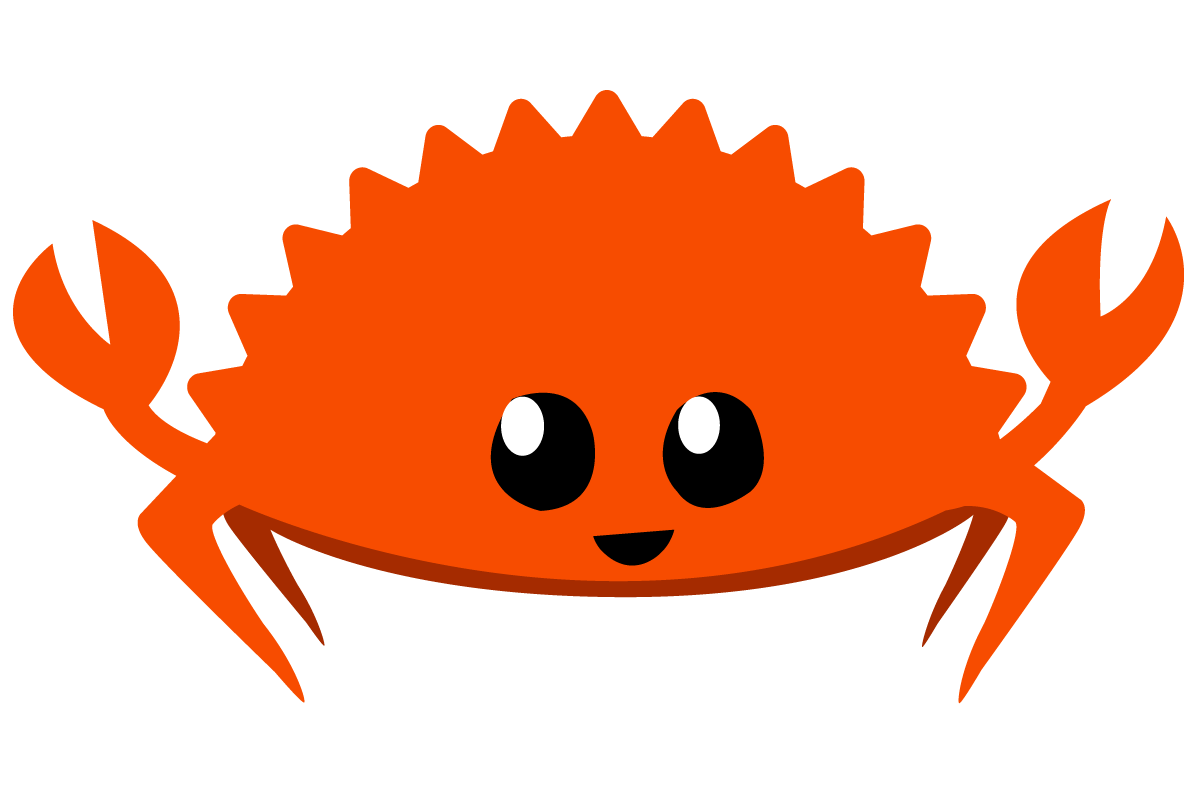


Рисунок 2.2 Феррис — талисман Rust

Ключевые приоритеты языка: безопасность, скорость и параллелизм. Используется для прикладного и системного программирования (в частности рассматривается как перспективный язык для разработки ядер операционных систем). Rust сопоставим по скорости и возможностям с C++/C, однако дает большую безопасность при работе с памятью, что обеспечивается встроенными в язык механизмами контроля ссылок. Производительности программ на Rust способствует использование абстракций с нулевой стоимостью. С 2016 по 2021 год Rust занимает первое место в списке любимых языков программирования по версии ежегодного опроса разработчиков Stack Overflow Developer Survey.

* 1. **Программные средства разработки**

dsf

**СПИСОК ЛИТЕРАТУРЫ**

1. «Partitioning and scheduling parallel programs for multiprocessors» Vivek Sarkar 1989
2. «Операционные системы. Разработка и реализация» Э. Таненбаум, А. Вудхал 2006
3. «Параллельное и конкурентное программирование на языке Haskell» Саймон Марлоу 2014
4. «Семь моделей конкуренции и параллелизма за семь недель» Пол Батчер 2015
5. «Основы параллельного программирования» К.Ю. Богачев 2020
6. «Параллельные системы баз данных» Л.Б. Соколинский 2013
7. «Принципы организации распределенных баз данных» М. Тамер Ёсу, Патрик Вальдуриес 2021
8. «Распараллеливание и планирование вычислительных и информационных процессов» А.А. Прихожий, М.В. Соломенник 2003
9. «Akka в действии» Р. Рестенбург, Р. Баккер, Р. Уильямс 2018
10. «Параллелизм» https://clck.ru/arKpC
11. «Блог компании Terradata» https://habr.com/ru/company/teradata/blog/
12. «Закон Мура» https://clck.ru/JXnTC
13. «Закон Амдала» <https://clck.ru/JmWru>
14. «Распределенные вычисления» https://clck.ru/MnRLq
15. «Распределенные системы. Принципы и парадигмы» Э. Таненбаум, М. ван Стеен 2003
16. «Операционные системы. Разработка и реализация» Э. Таненбаум, А. Вудхалл 2006
17. «Программирование на Эрланге. Программное обеспечение для параллельного мира» Джо Армстронг 2013
18. «Значения и Изменения. Подход Clojure к Идентичности и Состоянию» https://clojure.org/about/state
19. «Rayon: data parallelism in Rust» https://smallcultfollowing.com/babysteps/blog/2015/12/18/rayon-data-parallelism-in-rust/
20. «Массово-параллельная архитектура» https://clck.ru/dYvmo
21. «Язык Java» https://ru.wikipedia.org/wiki/Java
22. «Язык Rust» https://clck.ru/TxLA5