

Конспект по курсу Параллельное программирование ¹

Александра Лисицына ²

17 ноября 2018 г.

¹Читаемый Романом Елизаровым Никитой Ковалем в 2018-2019 годах

²Студентка группы М3334

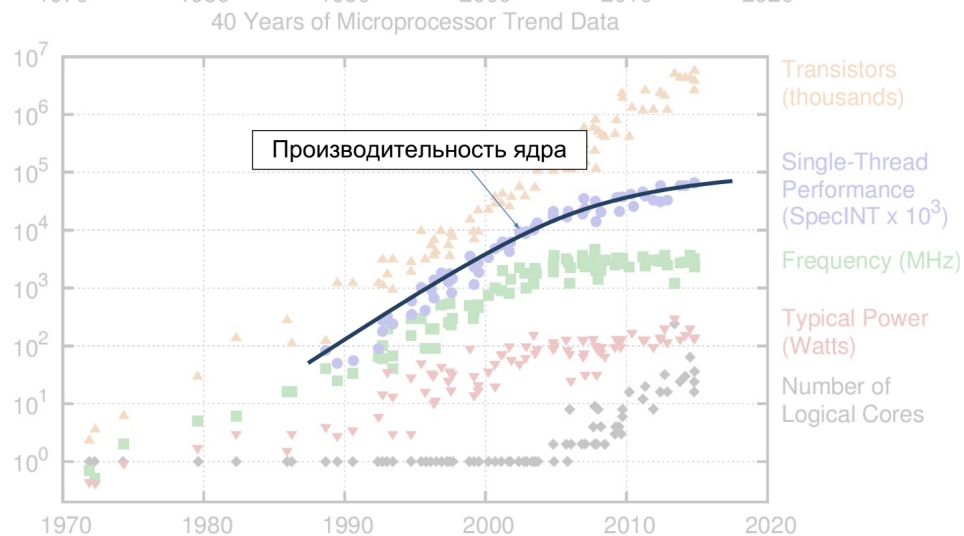
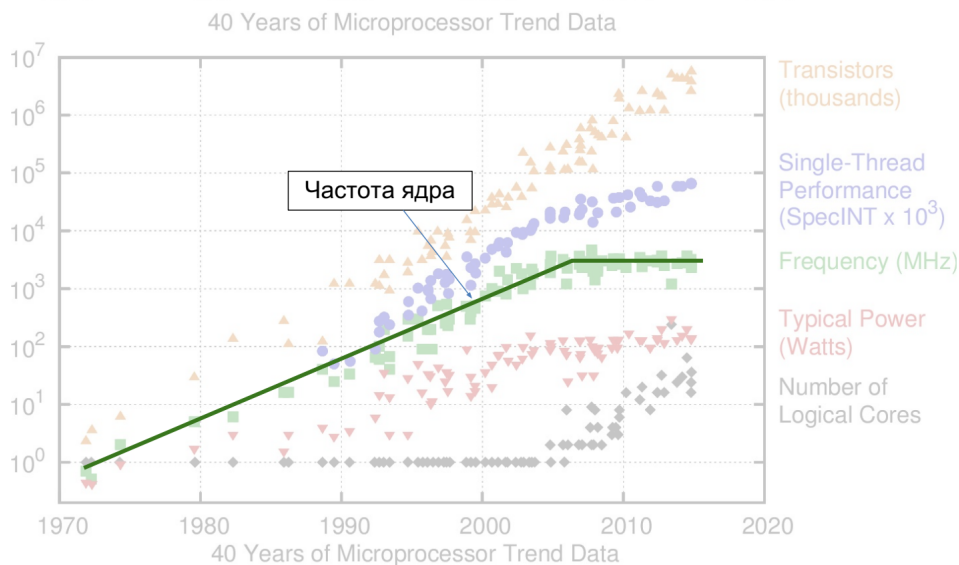
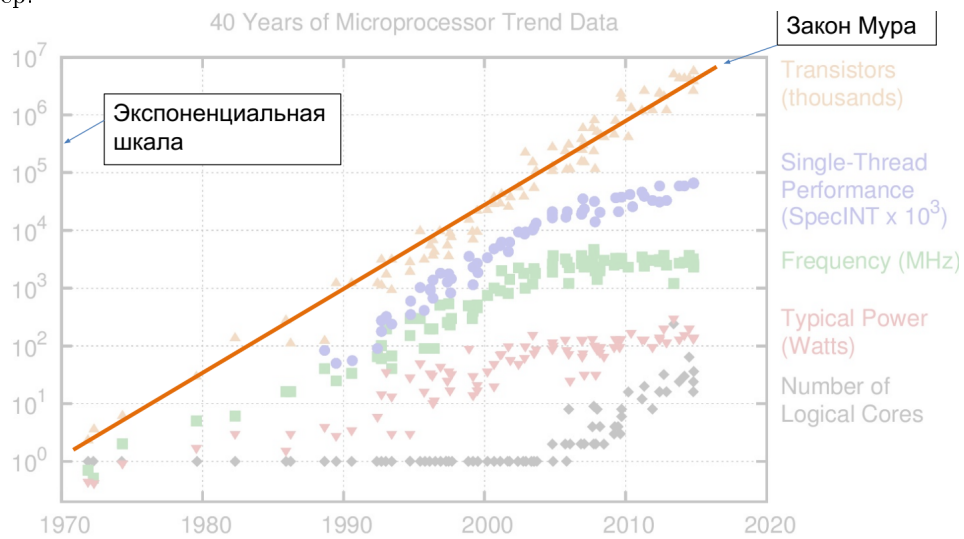
Содержание

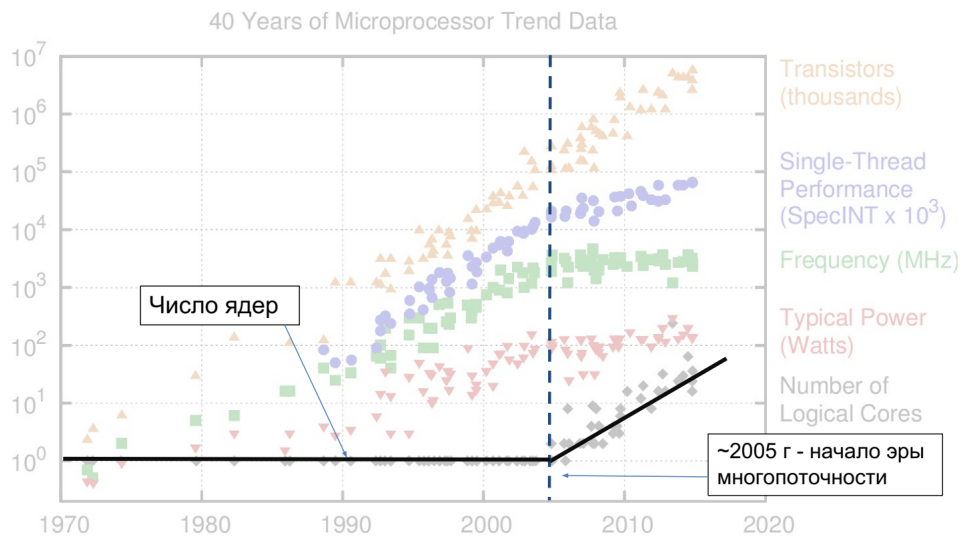
1	Intoduction	2
1.1	Закон Мура	2
1.2	Закон Амдала	3
1.3	Разные виды параллелизма	3
1.3.1	Параллелизм на уровне инструкций (ILP)	3
1.4	Операционные системы	5
1.5	Основные понятия в современных ОС	5
1.6	Формализм	6
1.6.1	Модели программирования	6
1.6.2	Общие объекты	6
1.6.3	Общие переменные	6
1.6.4	Свойства многопоточных программ	7
1.6.5	Моделирование многопоточного исполнения	7
2	Lock-free Treiber Stack and Michael-Scott Queue	8
3	Определения и формализм	8
3.1	Физическая реальность	8
3.2	Модель «произошло до» (happens before)	9
3.3	Модель глобального времени	9
3.4	Обсуждение глобального времени	9
3.5	«Произошло до» на практике	10
3.6	Свойства исполнений над общими объектами	10
3.6.1	Операции над общими объектами	10
3.6.2	Последовательное исполнение	10
3.6.3	Конфликты и гонки данных (data race)	10
3.6.4	Правильное исполнение	11
4	Построение атомарных объектов и блокировки	11
5	Consensus	11
5.1	Задача о консенсусе	11
5.2	Консенсусное число	11
5.3	Модель	11
5.4	Read-Modify-Write регистры	11
5.5	Универсальность консенсуса	11
6	Алгоритмы без блокировок: Построение на регистрах	11
6.1	Безусловные условия прогресса	11
7	CASN	12
7.1	Списки против массивов	12
7.2	CASn	12
7.2.1	DCSS	12
7.3	Наблюдения и замечания	12
7.4	Подход к реализации	12

1 Introduction

1.1 Закон Мура

Каждые 2 года количество транзисторов на процессоре удваивается. До, примерно, 2005 года также росла частота ядра. Также начал замедляться рост производительность ядра. С 2005 года начался рост числа ядер.





Определение. Масштабирование - свойство системы выполнять больше действий при увеличении мощности(традиционное), количества ядер(многопоточное).

В реале не получается сделать все идеально и для этого нужно изучать многопоточное программирование.

1.2 Закон Амдала

$$S = \frac{1}{N}$$

где S - это ускорение кода

Или

$$S = \frac{1}{1 - P + P/N}$$

где P - доля параллельного кода

Максимальное ускорение кода достигается при $N \rightarrow \infty$ и равно $1/(1 - P)$

P	S
60%	2.5
95%	20
99%	100

Поэтому нам необходимо увеличивать долю параллельного кода для достижения наилучшей масштабируемости.

1.3 Разные виды параллелизма

1.3.1 Параллелизм на уровне инструкций (ILP)

Способы использования ILP

- Конвейер
- Суперскалярное исполнение¹
 - Внеочередное исполнение
 - Переименование регистров²
 - Спекулятивное исполнение³
 - Предсказание переходов
- Длинное машинное слово (VLIW⁴)
- Векторизация (SIMD)



Рис. 1:

У параллелизма на уровне инструкций есть предел, поэтому нам необходимо параллельное программирование

Симметричная мультипроцессорность (SMP) Несколько вычислительных ядер у каждого свой поток исполняемых ресурсов

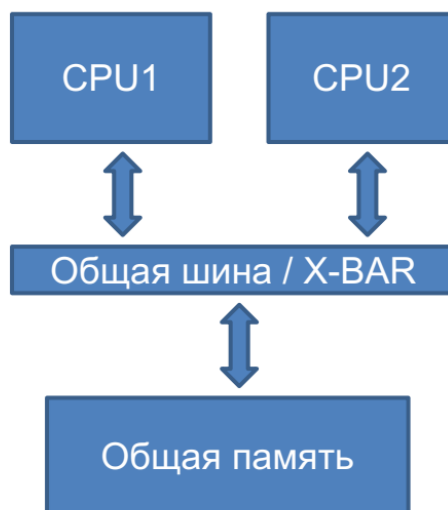


Рис. 2: SMP

Одновременная многозадачность (SMT) Два или более потока одновременно исполняются одним физическим ядром. Снаружи выглядит как SMP.

⁰Instruction Level Parallelism

¹Несколько операций за такт

²Чтобы не возникало ложной зависимости по регистрам

³Начинает выполнять одну из веток перехода, пытаясь ее предсказать

⁴Very Long Instruction Word

⁴Symmetric Multiprocessing

⁴Simultaneous Multithreading

⁴Non-uniform memory access

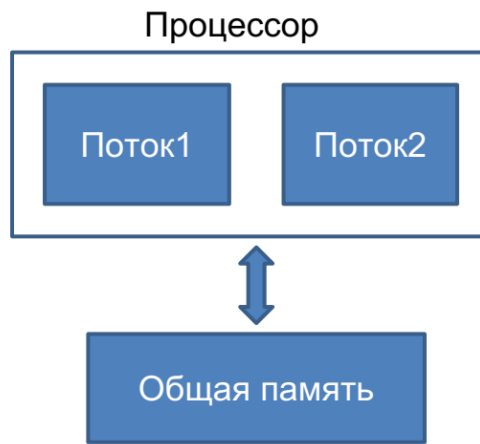


Рис. 3: SMT

Ассиметричный доступ к памяти (NUMA) Модель программирования та же, что в SMP, но без общей памяти.

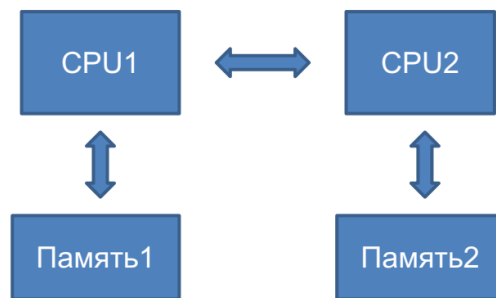


Рис. 4: NUMA

1.4 Операционные системы

- Типы
 - Однозадачные
 - Системы с пакетными задачами (batch processing)
 - Многозадачные / с разделением времени (time sharing)
 - * Кооперативная многозадачность (cooperative multitasking)
 - * Вытесняющая многозадачность (preemptive multitasking)
- История многозадачности
 - Изначально нужно было для раздела одной дорогой машины между несколькими пользователями
 - Теперь нужно для использования ресурсов одной многоядерной машины для множества задач

1.5 Основные понятия в современных ОС

•

Определение. Процесс — владеет памятью и ресурсами.

•

Определение. Поток — контекст исполнения внутри процесса.

- В одном процессе может быть несколько потоков

- Все потоки работают с общей памятью процесса
- Но в теории мы их будем смешивать

1.6 Формализм

1.6.1 Модели программирования

- «Классическое» однопоточное / однозадачное
 - Можем использовать ресурсы многоядерной системы только запустив несколько независимых задач
- Многозадачное программирование
 - Возможность использовать ресурсы многоядерной системы в рамках решения одной задачи
 - Варианты:
 - * Модель с общей памятью
 - * Модель с передачей сообщений (распределенное программирование)

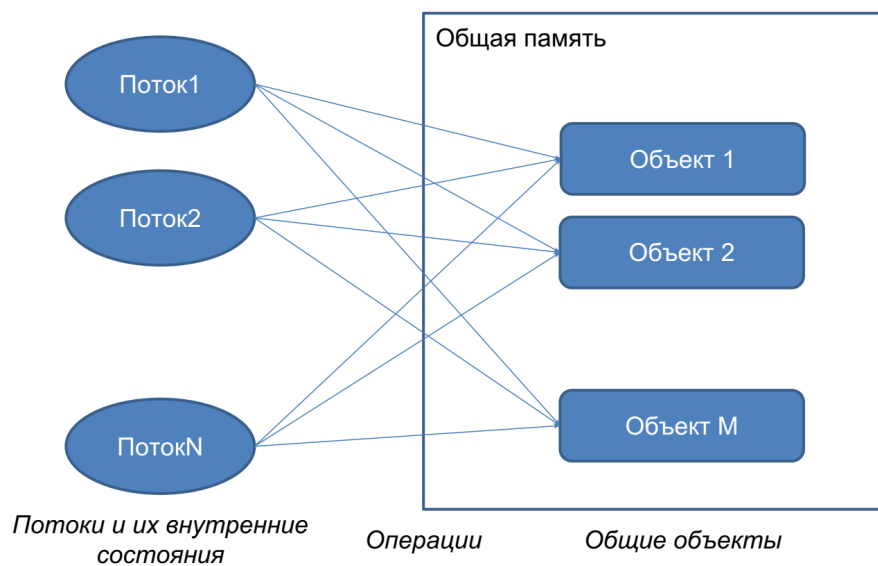


Рис. 5: Модель с общими объектами (общей памятью)

1.6.2 Общие объекты

- Потоки выполняют операции над общими, разделемыми объектами
- В этой модели не важны операции внутри потоков
- Важна только коммуникация между потоками
- В этой модели единственный тип коммуникации между потоками — это работа с общими объектами

1.6.3 Общие переменные

- Общие переменные — это простейший тип общего объекта:
 - У него есть значение определенного типа
 - Есть операция чтения (read) и записи (write)
- Общие переменные — это базовые строительные блоки для многопоточных алгоритмов
- Модель с общими переменными — это хорошая абстракция современных многопроцессорных систем и многопоточных ОС
 - На практике, это область памяти процесса, которая одновременно доступна для чтения и записи всем потокам этого процесса

В теоретических трудах общие переменные называют регистрами

1.6.4 Свойства многопоточных программ

- Последовательные программы детерминированы
 - Если нет использования случайных чисел и другого явного общения с недетерминированным миром
 - Их свойства можно установить анализируя последовательное исполнение при данных входных параметрах
- Многопоточные программы в общем случае недетерминированы
 - Даже если код каждого потока детерминирован
 - Результат работы зависит от фактического исполнения при данных входных параметрах
 - А этих исполнений может быть много
- Говорим «Программа А имеет свойство Р» если она имеет это свойство при любом исполнении

1.6.5 Моделирование многопоточного исполнения

1 shared int x = 0, y = 0

Thread P:

Thread Q:

```
1            x = 1
2            r1 = y
3            stop
```

```
1            y = 1
2            r2 = x
3            stop
```

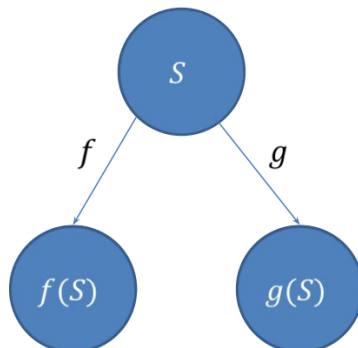


Рис. 6:

Моделирование исполнений через чередование операций

- S — это общее состояние:
 - Состояние всех потоков (IP+locals)
 - И состояние всех общих объектов
- f и g — это операции
 - Количество различных операций в каждом состоянии равно количеству потоков
- $f(S)$ — это новое состояние после применения операции f к состоянию S

После исполнения этого кода для r1, r2 возможны следующие пары значений: (0, 0), (0, 1), (1, 0), (1, 1). Хотя при моделировании через чередование (рисунок 7) первого варианта не получается. Это случается, так как в современном процессоре запись не попадает сразу в общую память, а в начале буферизируется (т.к. запись долгая операция). Поэтому мы можем прочитать старое значение, т.к. чтение

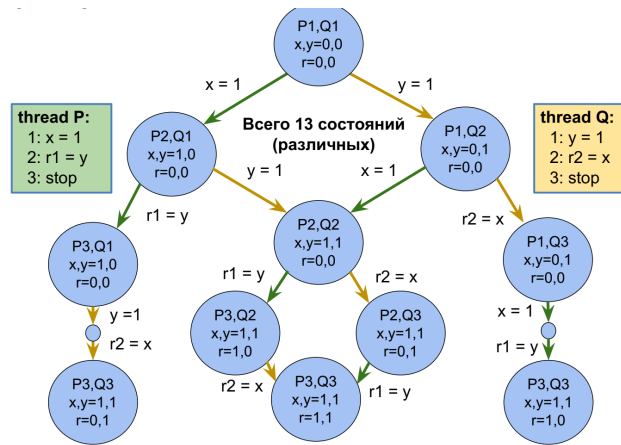


Рис. 7:

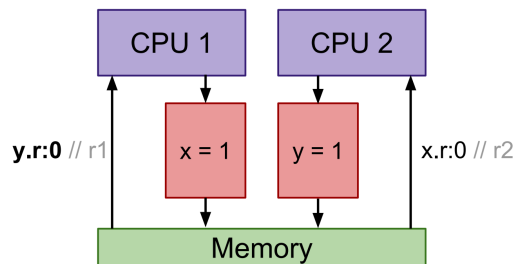


Рис. 8:

быстрая операция и новые значения еще лежат в буфере. Процессор может переставить инструкции, т.к. это может ускорить однопоточный код (процессор не знает о параллельности).

Модель чередования не параллельна

На самом деле в настоящих процессорах операции чтения и записи не мгновенные. Они происходят параллельно как в разных ядрах, так и в одном.

И вообще процессор обменивается с памятью сообщениями о чтении / записи и таких сообщений одновременно в обработке может быть очень много.

2 Lock-free Treiber Stack and Michael-Scott Queue

3 Определения и формализм

3.1 Физическая реальность

- Свет (электромагнитные волны) в вакууме распространится со скоростью $\sim 3 \cdot 10^8$ м/с.
 - Это максимальный физический предел скорости
 - За один такт процессора с частотой 3 ГГц ($3 \cdot 10^9$ Гц) свет в вакууме проходит всего 10 см.
- Соседние процессоры физически не могут синхронизировать свою работу и физически не могут определить порядок происходящих в них событиях.
 - Они работают действительно физически параллельно
- Пусть $a, b, c \in E$ — это физически атомарные (неделимые) события, происходящие в пространстве-времени (рисунок 9)
 - Говорим « a предшествует b » или « a произошло до b » (и записываем $a \rightarrow b$), если свет от точки пространства-времени a успевает дойти до точки пространства-времени b .
 - Это отношение частичного порядка на событиях

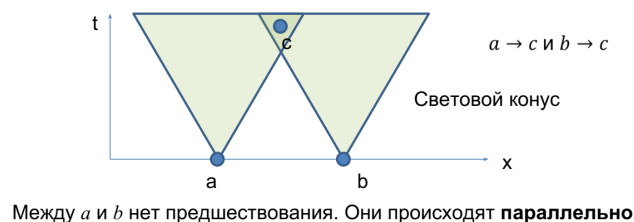


Рис. 9:

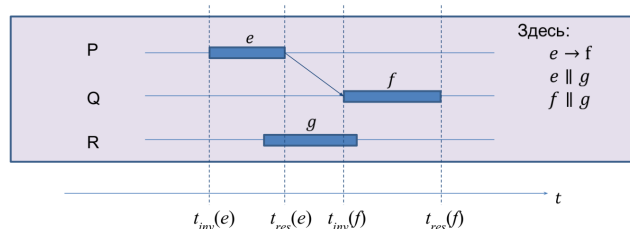


Рис. 10: Модель глобального времени

3.2 Модель «произошло до» (happens before)

- Впервые введена Л. Лампортом в 1978 году.
- Исполнение системы — это пара (H, \rightarrow_H)
 - H — это множество операций e, f, g, \dots (чтение и запись ячеек памяти и т. п.) произошедших во время исполнения
 - \rightarrow_H — это транзитивное, антирефлексивное, асимметричное отношение (частичный строгий порядок) на множестве операций
 - $e \rightarrow_H f$ означает, что « e произошло до f в исполнении H ». Чаще всего исполнение H понятно из контекста и опускается
- Две операции e и f параллельны ($e \parallel f$), если $e \not\rightarrow f \wedge f \not\rightarrow e$.
- Система — это набор всех возможных исполнений системы
- Говорим, что «система имеет свойство P», если каждое исполнение системы имеет свойство P.

3.3 Модель глобального времени

В этой модели каждая операция — это временной интервал (рисунок 10) $e = [t_{inv}(e), t_{res}(e)]$ где $t_{inv}(e), t_{res}(e) \in \mathbb{R}$ и

$$e \rightarrow f \stackrel{\text{def}}{=} t_{res}(e) < t_{inv}(f)$$

3.4 Обсуждение глобального времени

На самом деле никакого глобального времени нет и не может быть из-за физических ограничений.

- Это всего лишь механизм, позволяющий визуализировать факт существования параллельных операций.
- При доказательстве различных фактов и анализе свойств [исполнений] системы время не используется
 - Анализируются только операции и отношения «произошло до»

3.5 «Произошло до» на практике

- Современные языки программирования предоставляют программисту операции синхронизации:
 - Специальные механизмы чтения и записи переменных
 - Создание потоков и ожидание их завершения
 - Различные другие библиотечные примитивы для синхронизации
- Модель памяти языка программирования определяет то, каким образом исполнение операций синхронизации создает отношение «произошло до»
 - Без них разные потоки выполняются параллельно
 - Можно доказать те или иные свойства многопоточного кода, используя гарантии на возможные исполнения, которые дает модель памяти

3.6 Свойства исполнений над общими объектами

3.6.1 Операции над общими объектами

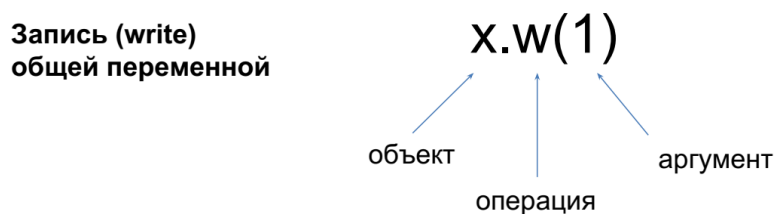


Рис. 11: Запись

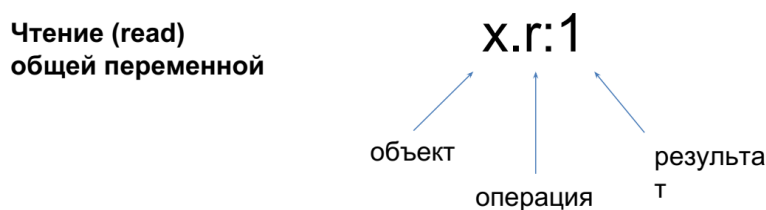


Рис. 12: Чтение

3.6.2 Последовательное исполнение

Определение. *Исполнение системы называется последовательным, если все операции линейно-упорядочены отношением «произошло до», то есть $\forall e, f \in H: (e = f) \vee (e \rightarrow f) \vee (f \rightarrow e)$.*

3.6.3 Конфликты и гонки данных (data race)

●

Определение. Две операции над одной переменной, одна из которых запись, называются конфликтующими.

Конфликтующие операции не коммутируют в модели чередования.

●

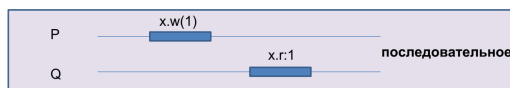


Рис. 13: Последовательное исполнение

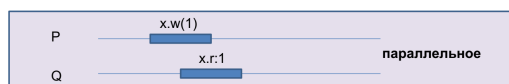


Рис. 14: Параллельное исполнение

Определение. Если две конфликтующие операции произошли параллельно, то такая ситуация называется гонка данных (data race).

– Это свойство конкретного исполнения.

Определение. Программа, в любом допустимом исполнении которой (с точки зрения модели памяти) нет гонок данных, называется корректно синхронизированной.

3.6.4 Правильное исполнение

4 Построение атомарных объектов и блокировки

5 Consensus

5.1 Задача о консенсусе

Каждый поток использует объект Consensus один раз

- Согласованность: все потоки должны вернуть одно и тоже значение из метода decide
- Обоснованность: возвращенное значение было входным значением какого-то из потоков
- Без ожидания

5.2 Консенсусное число

- Если с помощью класса атомарных объектов C и атомарных регистров можно реализовать консенсусный протокол без ожидания с помощью детерминированного алгоритма для N потоков (и не больше), то говорят, что у класса C консенсусное число N .
-

Теорема 5.1. Атомарные регистры имеют консенсусное число 1.

5.3 Модель

- x -валентное состояние системы — консенсус во всех нижестоящих листьях будет x .
- Бивалентное состояние — возможен консенсус как 0, так и 1.
- Критическое состояние — такое бивалентное состояние, у которого все дети одновалентны

5.4 Read–Modify–Write регистры

5.5 Универсальность консенсуса

Теорема 5.2. Любой последовательный объект можно реализовать без ожидания для N потоков, используя консенсусный протокол для N потоков

6 Алгоритмы без блокировок: Построение на регистрах

6.1 Безусловные условия прогресса

- Отсутствие помех — Если несколько потоков пытаются выполнить операцию, то любой из них должен выполнить ее за конечное время, если все другие остановились в любом месте
- Отсутствие блокировки — если несколько потоков пытаются выполнить операцию, то хотя бы один из них должен выполнить ее за конечное время, независимо от действия или бездействия других потоков

- Отсутствие ожидания — если поток хочет выполнить операцию, то он выполнит ее за конечное время независимо от других потоков

7 CASN

7.1 Списки против массивов

- Список
 - $3 * N$ слов памяти (минимум)
 - при многопотном использовании хватает одного CAS для добавления элемента
- Массив
 - $5 + N$ слов памяти (до $5 + 2 * N$ при x2 резерве)
 - Как минимум в два раз быстрее работает с памятью (обычно растёт на попяток) (при одно-поточной работе)
 - Для добавления нужно CAS2 (для size и элемента) — эта операция не поддерживается на процессорах

7.2 CASn

```

1 class Ref<T>(initial: T) {
2     var _v
3     \\ ...

```

7.2.1 DCSS

```

1 fun <A,B> dcsc(
2     a: Ref

```

7.3 Наблюдения и замечания

7.4 Подход к реализации