

# Теория вычислительных процессов

## Process calculus

### Параллельное программирование

Черновик

ИВТ и ПМ  
ЗабГУ

2024

# Outline

## Введение

- Классификация

- Закон Амдала

- Декомпозиция

## Проблемы параллельных вычислений

## Потоки в C++

## Синхронизация

- Мьютекс

## Ссылки

# Зачем?

- ▶ Если задача не решается за достаточное время на одном компьютере (ядре, процессоре, ...) то их нужно увеличить число компьютеров
- ▶ Особенно много времени могут занимать научные и инженерные вычисления
  - ▶ Моделирование физических процессов, прочностные расчёты
  - ▶ Фолдинг белков, SETI, климатические вычисления
  - ▶ БАК в секунду генерирует 100Тб данных
  - ▶ BigData и DataScience

# Зачем?

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	<b>Supercomputer Fugaku</b> - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	<b>Summit</b> - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	<b>Sierra</b> - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	<b>Sunway TaihuLight</b> - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	<b>Perlmutter</b> - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70,870.0	93,750.0	2,589

Top500 supercomputers (ноябрь 2021); Rpeak – теоретическая производительность; Rmax – произв. по LINPACK

Для сравнения: AMD Ryzen 9 3950X 3.5 ГГц, 16 ядер (2019) – 896 ГФлоп/с

# Зачем?

- ▶ Серверы и другие сетевые программы тоже требуют параллельного выполнения задач
- ▶ Банковские процессы (обработка транзакций )
- ▶ Компьютерная графика
- ▶ Машинное обучение
- ▶ ...

## || вычисления в кино

- ▶ Lord of the Rings (2001-2003)
  - ▶ company: Weta Digitals
  - ▶ 3200 processor cluster
  - ▶ single scene contains
    - ▶ per second 24 frames
    - ▶ per frame: 4996 x 3112 points with 32- or 64 bit color encoding
    - ▶ each object means a separate compute cycle



GameMovie.com

# Зачем?

- ▶ Фоновое выполнение задач в программах с интерфейсом пользователя

Фоновые вычисления тем более эффективны, потому что практически все современные устройства (ПК, мобильные устройства) имеют многоядерные процессоры, не говоря уже о серверах и суперкомпьютерах.

# План

1. Введение в || программирование
2. Потоки в C++.
3. Взаимодействие, синхронизация и обмен данными.
4. OpenMP (Open Multi-Processing). Параллельные вычисления на C++
5. MPI (Message Passing Interface)
6. Вычисления на GPU (python - pyTorch, google colab)
7. ???
8. Доклады по желанию

Не будет рассмотрена организация процессов и потоков в ОС



# Ссылки

- ▶ CSC курс || программирования:  
[wiki.osll.ru/doku.php/courses:high\\_performance\\_computing :  
start](http://wiki.osll.ru/doku.php/courses:high_performance_computing:start)
- ▶ Параллельное программирование на C++ в действии.  
Практика разработки многопоточных программ, 2016,  
Энтони Уильямс
- ▶ Ваши любимые книги?

# Асинхронный и синхронный процессы

**Асинхронный процесс** – процесс, в котором операция не требует отклика от другой операции для своего выполнения.

**Синхронный процесс** – процесс, в котором операция требует отклика для продолжения выполнения.

# Concurrency vs Parallelism

**Concurrency** is when two or more tasks can start, run, and complete in overlapping time periods. It doesn't necessarily mean they'll ever both be running at the same instant. For example, multitasking on a single-core machine.

**Parallelism** is when tasks literally run at the same time, e.g., on a multicore processor. is when tasks literally run at the same time, e.g., on a multicore processor.

# Concurrency vs Parallelism

**Concurrency**



**Parallelism**



# Concurrency vs Parallelism

Пример concurrency: попеременная (причем порядок заранее не определён) работа нескольких потоков на одном ядре процессора.

Пример параллелизма: одновременная работа потоков на нескольких ядрах процессора.

Concurrency тоже может дать прирост в производительности, например за счёт того, что ожидание ресурсов может быть выделено в отдельный поток.

# Режимы выполнения программ

- ▶ Многозадачный режим (режим разделения времени)  
Параллельности нет. Программы выполняются попеременно и последовательно. Выигрыш в быстродействии за счёт отсутствия блокирующих программу ожиданий.
- ▶ Параллельный режим
- ▶ Распределённое выполнение то же самое что и параллельный режим, только выполнение на физически отдельных устройствах с существенными временными задержками на синхронизацию.

# Outline

## Введение

- Классификация

- Закон Амдала

- Декомпозиция

## Проблемы параллельных вычислений

## Потоки в C++

## Синхронизация

- Мьютекс

## Ссылки

# Классификация Флинна (Flynn)

- ▶ классификация архитектур ЭВМ по признакам наличия параллелизма в потоках команд и данных
- ▶ Предложена предложена Майклом Флинном в 1960-х

		Instruction stream	
		Single	Multiple
Data stream	Single	SISD	MISD
	Multiple	SIMD	MIMD



# Классификация Флинна (Flynn)

		Instruction stream	
		Single	Multiple
Data stream	Single	SISD	MISD
	Multiple	SIMD	MIMD

- ▶ SISD, single instruction stream over a single data stream
- ▶ SIMD, single instruction, multiple data
- ▶ MISD, multiple instruction, single data
- ▶ MIMD, multiple instruction, multiple data

# SISD

- ▶ Классическая архитектура фон Неймана.
- ▶ || отсутствует
- ▶ Все однопроцессорные (с одним ядром) системы
- ▶ Частный случай SISD – конвейер

# Вычислительный конвейер (Instruction pipelining)

<b>Instr. No.</b> \ <b>Clock cycle</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
(IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back).							

- ▶ Конвейер позволяет достичь равномерной скорости выполнения всего цикла операций.
- ▶ Как будет отличаться частота выполнения цикла операций (из 5 этапов) на конвейере и с помощью 5 параллельных обработчиков?

- ▶ Используется для резервирования
- ▶ Применяется в системах критичных к сбоям (самолёты, космические аппараты, системы SCADA<sup>1</sup>)

---

<sup>1</sup>Supervisory Control And Data Acquisition — диспетчерское управление и сбор данных

- ▶ Одновременное выполнение одной инструкции над разными данными
- ▶ Векторный процессор — это процессор, в котором операндами некоторых команд могут выступать упорядоченные массивы данных — векторы
- ▶ В большинстве современных микропроцессоров имеются векторные расширения (SSE и др)
- ▶ Большинство современных графических процессоров

- ▶ SSE (Streaming SIMD Extensions, потоковое SIMD-расширение процессора) – это SIMD набор инструкций, разработанный Intel и впервые представленный в процессорах серии Pentium III как ответ на аналогичный набор инструкций 3DNow! от AMD.
- ▶ SSE – только один из примеров векторных расширений

Вывод команды `lscpu` (Linux)

```

Архитектура:          x86_64
  CPU op-mode(s):    32-bit, 64-bit
  Address sizes:      48 bits physical, 48 bits virtual
  Порядок байт:      Little Endian
CPU(s):              4
  On-line CPU(s) list: 0-3
ID производителя:    AuthenticAMD
  Имя модели:        AMD A12-9700P RADEON R7, 10 COMPUTE CORES 4C+6G
    Семейство ЦПУ:    21
    Модель:          101
    Потоков на ядро:   2
    Ядер на сокет:    2
    Сокетов:         1
    Степпинг:        1
    Frequency boost:  enabled
    CPU max MHz:      2500,0000
    CPU min MHz:      1300,0000
    BogoMIPS:         4990.92
  Флаги:             fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mc
                    a cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall n
                    x mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_go
                    od acc_power nopl nonstop_tsc cpuid extd_apicid aperfmp
                    erf pni pclmulqdq monitor ssse3 fma cx16 sse4_1 sse4_2
                    movbe popcnt aes xsave avx f16c lahfm cmp_legacy svm
                    extapic cr8_legacy abm sse4a misalignsse 3dnowprefetch
                    osvw ibs xop skinit wdt lwp fma4 tce nodeid_msr tbm top
                    oext perfctr_core perfctr_nb bpext ptsc mwaitx cpb hw_p
                    state ssbd vmcall fsgsbase bmi1 avx2 smep bmi2 xsaveop
                    t arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean
                    flushbyasid decodeassists pausefilter pfthreshold avic
                    v_vmsave_vmload vgif overflow_recover

```


# SSE

**CPU-Z**

**CPU** | Mainboard | Memory | SPD | Graphics | Bench | About

**Processor**

Name	Intel Core i5 11400F		
Code Name	Rocket Lake	Max TDP	65.0 W
Package	Socket 1200 LGA		
Technology	14 nm	Core Voltage	1.136 V



Specification: 11th Gen Intel® Core™ i5-11400F @ 2.60GHz

Family	6	Model	7	Stepping	1
Ext. Family	6	Ext. Model	A7	Revision	B0

Instructions: MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, AES, AVX, AVX2, AVX512F, FMA3, SHA

**Clocks (Core #0)**

Core Speed	798.05 MHz
Multiplier	x 8.0 ( 8 - 6 )
Bus Speed	99.76 MHz
Rated FSB	

**Cache**

L1 Data	6 x 48 KBytes	12-way
L1 Inst.	6 x 32 KBytes	8-way
Level 2	6 x 512 KBytes	8-way
Level 3	12 MBytes	16-way

Selection: Socket #1 | Cores: 6 | Threads: 12

**CPU-Z** Ver. 1.99.0.x64 | Tools | Validate | Close



# Intel SIMD ISA Evolution

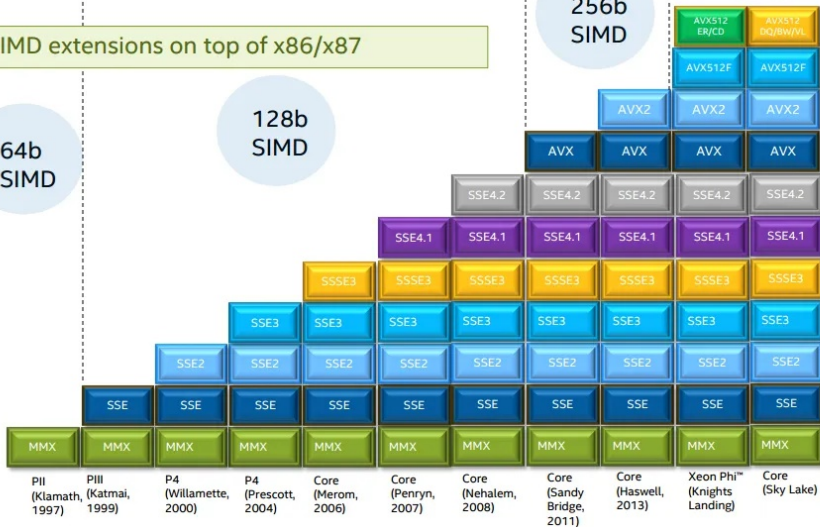
SIMD extensions on top of x86/x87

64b  
SIMD

128b  
SIMD

256b  
SIMD

512b  
SIMD



# SSE

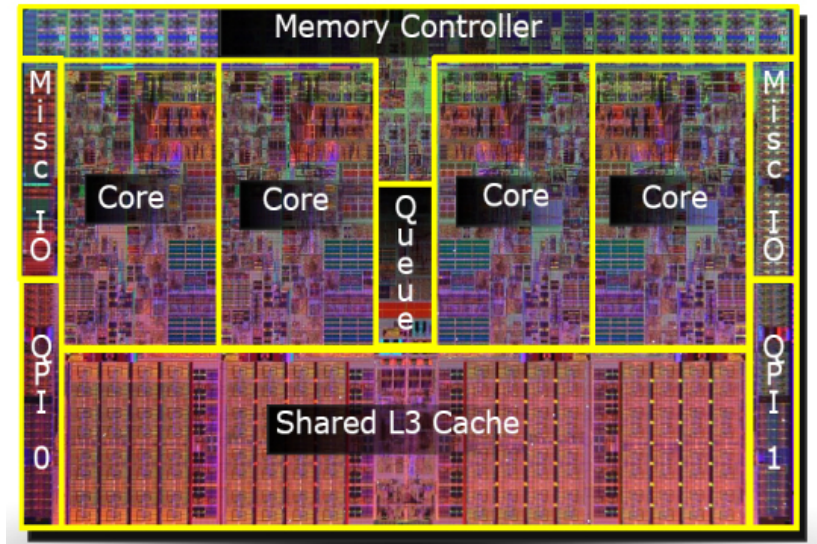
перемножение четырёх пар чисел с плавающей точкой одной инструкцией mulps

```
__declspec(align(16)) float a[4] = { 300.0, 4.0, 4.0, 12.0 };
__declspec(align(16)) float b[4] = { 1.5, 2.5, 3.5, 4.5 };
__asm {
    movups xmm0, a    ; //поместить 4 перем. из a в рег-р xmm0
    movups xmm1, b    ; //поместить 4 перем. из b в рег-р xmm1
    mulps xmm0, xmm1  ; //перемножить пакеты перем:
                        ; //xmm0 = xmm0 * xmm1
                        ; // xmm00 = xmm10 * xmm00
                        ; // xmm01 = xmm11 * xmm01
                        ; // xmm02 = xmm12 * xmm02
                        ; // xmm03 = xmm13 * xmm03

    movups a, xmm0    ; // выгрузить результаты
                        ; // из регистра xmm0 по адресам a
};
```

# Устройство ЦП

Микроснимок четырёхядерного процессора Intel с кодовым именем Nehalem



Почему чаще всего компьютер имеет несколько ядер в одном процессоре, чем несколько процессоров?

- ▶ Разные вычислительные устройства выполняют свои программы на своих данных
- ▶ Почти все суперкомпьютеры.

МКМД:

## Мультипроцессоры (общая память)

Унифицированный доступ к памяти  
(uniform memory access: UMA)

PVP (parallel vector processors)

SMP (symmetric multiprocessors)

Не унифицированный доступ к памяти  
(non UMA: NUMA)

COMA (cash-only memory architecture)

CC-NUMA (cache-coherent NUMA)

NCC-NUMA (non cache-coherent NUMA)

## Мультикомпьютеры (распределенная память)

MPP (massively parallel processor)

Кластеры (Clusters)

# Классификация 2

- ▶ Системы с общей памятью
  - ▶ Легко программировать
  - ▶ Не нужно пересылать сообщения
  - ▶ Трудно масштабировать
  - ▶ Конкуренция за доступ к общим ресурсам
  - ▶ Многопоточность, OpenMP
- ▶ Системы с распределённой памятью
  - ▶ Отказоустойчивы
  - ▶ Сложнее программировать
  - ▶ Нужен обмен данными
  - ▶ Легко масштабировать
  - ▶ Несколько процессов, MPI

# Outline

## Введение

Классификация

**Закон Амдала**

Декомпозиция

Проблемы параллельных вычислений

Потоки в C++

Синхронизация

Мьютекс

Ссылки

# Закон Амдала

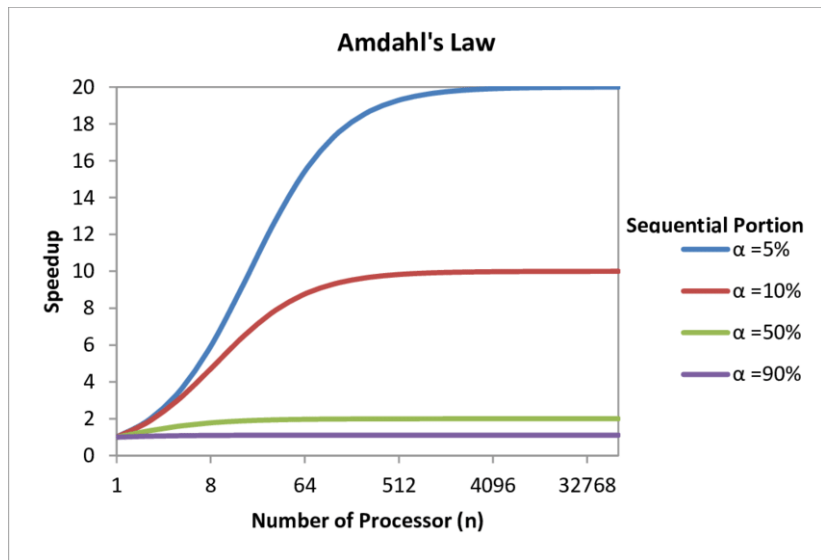
ускорение, которое может быть получено на вычислительной системе из процессоров, по сравнению с однопроцессорным решением не будет превышать величины

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

- ▶  $\alpha$  – доля последовательных вычислений
- ▶  $p$  – число процессов



# Закон Амдала



# Закон Амдала

- ▶ Закон Амдала даёт только верхнюю оценку ускорению, фактическое же ускорение будет ниже из-за накладных расходов на создание процессов и синхронизации.
- ▶ Плохо поддаются распараллеливанию рекурсивные и итерационные алгоритмы

# С чего стоит начинать распараллеливание программы?

- ▶ При распараллеливании алгоритма можно в первом приближении пользоваться правилами оптимизации.
- ▶ Правило 90/10: 90% времени выполнения программы занимает выполнение всего 10% кода.
- ▶ Значительную часть этих 10% занимают циклы

См. также оптимизацию циклом компилятором

# Outline

## Введение

Классификация

Закон Амдала

Декомпозиция

Проблемы параллельных вычислений

Потоки в C++

Синхронизация

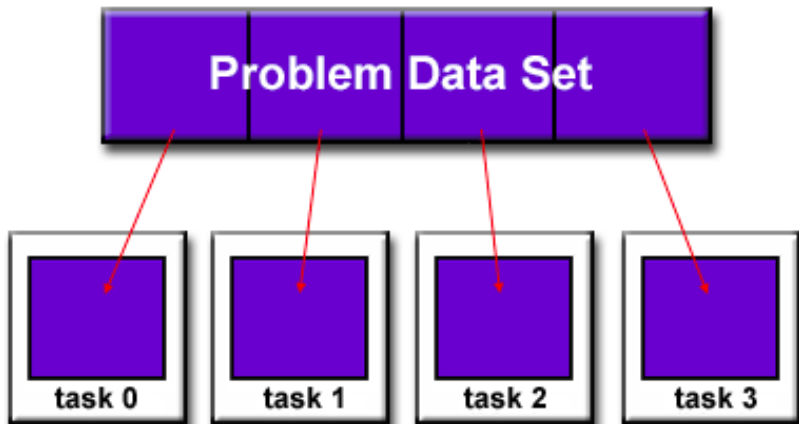
Мьютекс

Ссылки

# Декомпозиция

- ▶ По данным
- ▶ По инструкциям

## Декомпозиция по данным



# Декомпозиция по данным

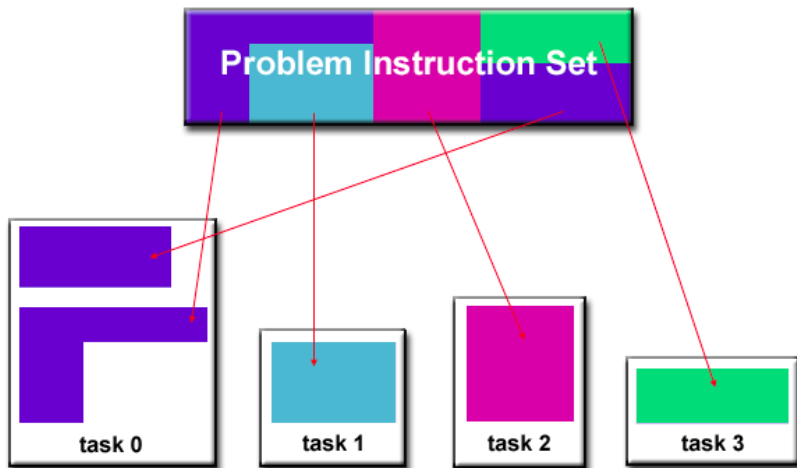
Пронумеруем потоки от 1 до N.

Каждый поток  $P_i$  получит на обработку количество частей данных:

$$P_i = M/N + \begin{cases} 1, & \text{если } i \leq M \% N \\ 0, & \text{в обратном случае} \end{cases}$$

- ▶ N – число потоков
- ▶ M – число частей данных

# Функциональная декомпозиция





# Outline

Введение

Классификация

Закон Амдала

Декомпозиция

Проблемы параллельных вычислений

Потоки в C++

Синхронизация

Мьютекс

Ссылки

# Атомарная операция

**Атомарная операция** — операция, которая либо выполняется целиком, либо не выполняется вовсе; операция, которая не может быть частично выполнена и частично не выполнена.

В C++ существуют отдельные типы данных `atomicxxx`

# Атомарная операция

- ▶ Атомарность операции можно проверить записав её в виде ассемблерного кода
- ▶ Если операция будет представлена одной ассемблерной операцией, то она атомарна

# Атомарная операция

- ▶ Атомарность операции можно проверить записав её в виде ассемблерного кода
- ▶ Если операция будет представлена одной ассемблерной операцией, то она атомарна
- ▶ Является ли операция присваивания атомарной?
- ▶ Является ли операция инкремента атомарной?
- ▶ Является ли операция добавления (для числовой переменной)?
- ▶ Атомарность операции может быть гарантирована и программным способом.

# Потоковая безопасность

**Потоковая безопасность** (thread-safety) — это концепция программирования, применимая к многопоточным программам. Код потокобезопасен, если он функционирует исправно при использовании его из нескольких потоков одновременно. В частности, он должен обеспечивать правильный доступ нескольких потоков к разделяемым данным.

# Состояние гонки (race condition)

- ▶ Состояние гонки = конкуренция
- ▶ Однако академически правильный термин для этой проблемы: **неопределённость параллелизма**.
- ▶ Проблема: после проверки данных одним потоком, они изменяются другим потоком. Первый поток не замечает этого изменения.

см также проблемы [Therac-25](#)

## Состояние гонки (race condition)

```
volatile int x;  
// Поток 1:  
while (!stop) {  
    x++;  
    ... }  
  
// Поток 2:  
while (!stop) {  
    if (x%2 == 0)  
        System.out.println("x=" + x);  
    ... }
```

Пусть  $x=0$ . Предположим, что выполнение программы происходит в таком порядке:

1. Оператор `if` в потоке 2 проверяет  $x$  на чётность.
2. Оператор « $x++$ » в потоке 1 увеличивает  $x$  на единицу.
3. Оператор вывода в потоке 2 выводит « $x=1$ », хотя, казалось бы, переменная проверена на чётность.

# Состояние гонки (race condition)

Решение 1: Локальная копия

```
// Поток 2:  
while (!stop)  
{  
    int cached_x = x;  
    if (cached_x%2 == 0)  
        System.out.println("x=" + cached_x);  
    ...  
}
```

Операция копирования переменной должна быть атомарной



# Состояние гонки (race condition)

Решение 1: Локальная копия

```
int x;
```

```
// Поток 1:
```

```
while (!stop){  
    synchronized(someObject) {  
        x++;}  
    ...  
}
```

```
// Поток 2:
```

```
while (!stop){  
    synchronized(someObject){  
        if (x%2 == 0)  
            System.out.println("x=" + x);}  
    ...  
}
```

Блок synchronized может выполняться в один момент времени только одним потоком

# Взаимная блокировка

Взаимная блокировка (deadlock) – ситуация в многозадачной среде или СУБД, при которой несколько процессов находятся в состоянии ожидания ресурсов, занятых друг другом, и ни один из них не может продолжать свое выполнение.

Шаг	Процесс 1	Процесс 2
0	Хочет захватить А и В, начинает с А	Хочет захватить А и В, начинает с В
1	Захватывает ресурс А	Захватывает ресурс В
2	Ожидает освобождения ресурса В	Ожидает освобождения ресурса А
3	Взаимная блокировка	

# Взаимная блокировка

## Решение

- ▶ Классический способ борьбы с проблемой — разработка иерархии блокировок
- ▶ между блокировками устанавливается отношение сравнения и вводится правило о запрете захвата «большой» блокировки в состоянии, когда уже захвачена «меньшая».
- ▶ Таким образом, если процессу нужно несколько блокировок, ему нужно всегда начинать с самой «большой» — предварительно освободив все захваченные «меньшие», если такие есть — и затем в нисходящем порядке.

# Проблема АВА

- ▶ Процесс  $P_1$  читает значение  $A$  из разделяемой памяти,
- ▶  $P_1$  вытесняется, позволяя выполняться  $P_2$ ,
- ▶  $P_2$  меняет значение  $A$  на  $B$  и обратно на  $A$  перед вытеснением,
- ▶  $P_1$  возобновляет работу, видит, что значение не изменилось, и продолжает...
- ▶ Хотя  $P_1$  может продолжать работу, возможно, что его поведение будет неправильным из-за других, скрытых изменений общей памяти (которые он не отслеживал).

# Проблема АВА

## Решения

- ▶ Использование счётчика числа изменений переменной
- ▶ атомарная инструкция – сравнение с обменом (compare and swap, CAS)

# Outline

Введение

Классификация

Закон Амдала

Декомпозиция

Проблемы параллельных вычислений

**Потоки в C++**

Синхронизация

Мьютекс

Ссылки

# Процессы и потоки

- ▶ <https://en.cppreference.com/w/cpp/thread/thread>
- ▶ Обёртка над потоками ОС
- ▶ [github.com/ivtipm/ProcessCalculus/tree/master/examples/example\\_thread](https://github.com/ivtipm/ProcessCalculus/tree/master/examples/example_thread)

# volatile

- ▶ Компилятор оптимизирует код
- ▶ В некоторых случаях код может быть существенно переписан
- ▶ Например, компилятор может проанализировать код и подставить везде вместо переменной её значение, полагая что переменная не изменяет своего значения
- ▶ Однако компилятор не учитывает случая, когда переменная изменяется в отдельном потоке
- ▶ Чтобы избежать таких вредных оптимизаций переменную помечают как `volatile`
- ▶ см. также [https://en.wikipedia.org/wiki/Volatile\(\*computer\\_programming\*\)](https://en.wikipedia.org/wiki/Volatile_(computer_programming))



# volatile

```
bool cancel = false;  
while( !cancel ) {  
    ;  
}
```

Если cancel не меняется, то ее и проверять каждый раз незачем, компилятор и не будет ее проверять.

Зато если вы укажете перед переменной `volatile`, то оптимизации не будет. Предполагается, что переменная `cancel` могла измениться каким-то волшебным образом.

```
volatile bool cancel = false;  
while( !cancel ) {  
    ;  
}
```

# Outline

Введение

Классификация

Закон Амдала

Декомпозиция

Проблемы параллельных вычислений

Потоки в C++

**Синхронизация**

Мьютекс

Ссылки

## Критическая секция (critical section)

- ▶ **Критическая секция** (critical section) – участок исполняемого кода программы, в котором производится доступ к общему ресурсу (данным или устройству), который не должен быть одновременно использован более чем одним потоком выполнения.
- ▶ При нахождении в критической секции двух (или более) потоков возникает состояние «гонки» («состязания»). Для избежания данной ситуации необходимо выполнение четырех условий:
  - ▶ Два потока не должны одновременно находиться в критических областях. В программе не должно быть предположений о скорости или количестве процессоров. Поток, находящийся вне критической области, не может блокировать другие потоки. Невозможна ситуация, в которой поток вечно ждет попадания в критическую область.

## Критическая секция (critical section)

- ▶ При нахождении в критической секции двух (или более) потоков возникает состояние «гонки» («состязания»). Для избежания данной ситуации необходимо выполнение четырех условий:
  - ▶ Два потока не должны одновременно находиться в критических областях.
  - ▶ В программе не должно быть предположений о скорости или количестве процессоров.
  - ▶ Поток, находящийся вне критической области, не может блокировать другие потоки.
  - ▶ Невозможна ситуация, в которой поток вечно ждет попадания в критическую область.

# Outline

Введение

Классификация

Закон Амдала

Декомпозиция

Проблемы параллельных вычислений

Потоки в C++

Синхронизация

Мьютекс

Ссылки

Мьютекс (mutex, от mutual exclusion — «взаимное исключение»)

# Outline

Введение

Классификация

Закон Амдала

Декомпозиция

Проблемы параллельных вычислений

Потоки в C++

Синхронизация

Мьютекс

Ссылки

Материалы дисциплины  
[github.com/ivtipm/ProcessCalculus](https://github.com/ivtipm/ProcessCalculus)