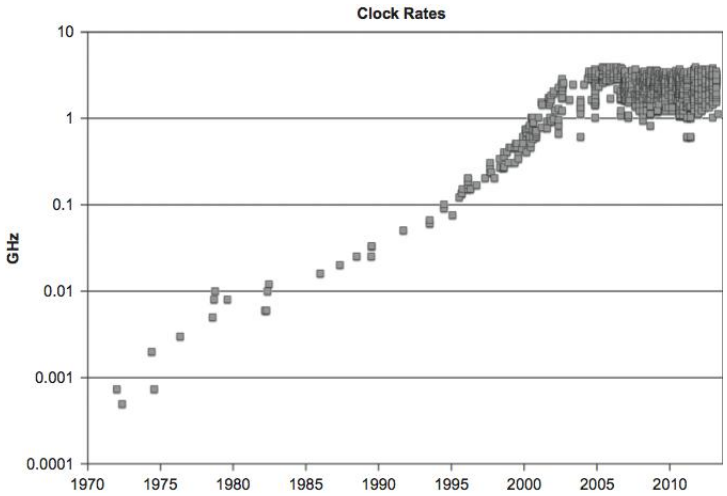
МНОГОПОТОЧНОСТЬ И ПАРАЛЛЕЛИЗМ В C++

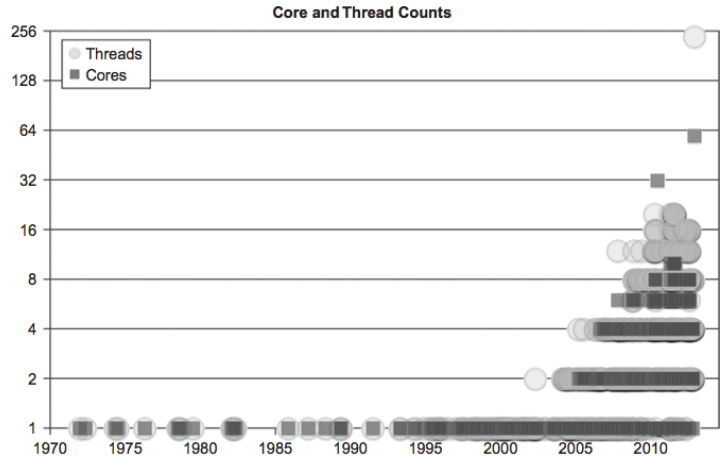
Курсы Intel Delta-3

Алексей Куканов, Intel Corporation

Нижний Новгород, 2015

Параллелизм – норма жизни





- Мультиядерные и многоядерные процессоры
- SIMD, SIMT, SMT
- Степень аппаратного ||| продолжает расти
- Требуется явный ||| на уровне софта

- Здесь и далее:
  ||| означает параллелизм

C++ до 2011: ||| на свой страх и риск

Стандарт языка C++ (1998)

- Строго последовательная модель, ||| не рассматривается вовсе

Разнообразные библиотеки потоков /*threads*/

- POSIX threads, WinAPI threads, Boost, …
- Чаще всего, кросс-платформенные обёртки над средствами ОС

OpenMP

- Языковое расширение на основе прагм, развиваемое консорциумом производителей
- Ориентировано на HPC (Fortran,C); слабо интегрировано с C++

«Молодая шпана»

- TBB (Intel), PPL (Microsoft), GCD (Apple), CUDA (NVidia), Cilk++ (CilkArts => Intel), академические разработки, ...
- Потоки, если и используются, скрыты от программиста

Часть 1. Многопоточность в C++11

C++11: необходимые основы

- Многопоточная модель исполнения программы
- Отношение порядка исполнения /**happens-before**/
- Синхронизированные операции
- «Гонки данных» /**data races**/ объявлены UB
- Переменные, локальные для потока: тип хранения `thread_local`

Поддержка многопоточности на уровне языка

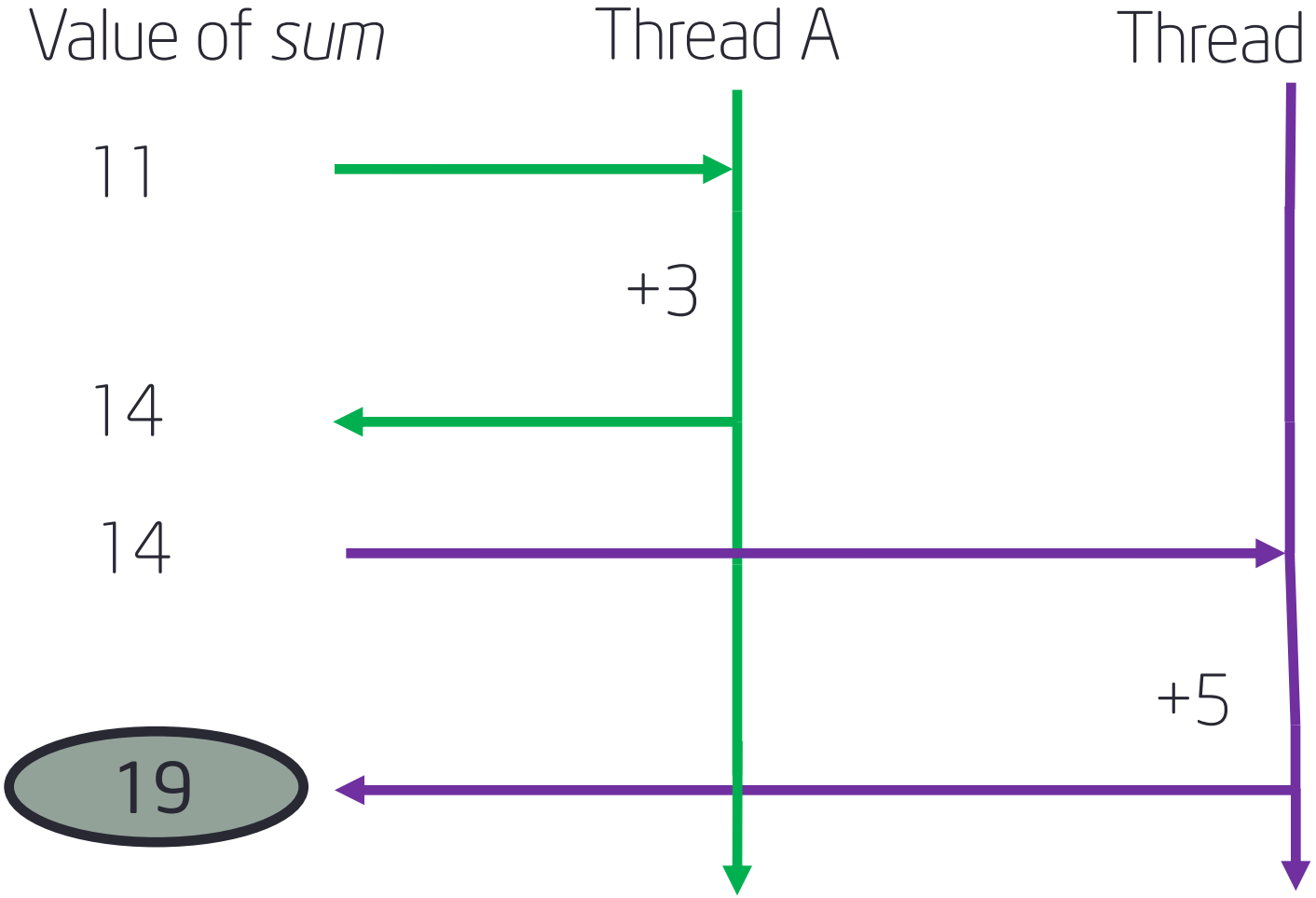
«Гонки данных» /*data races*/

- Два вычисления *конфликтуют*, если они используют одну и ту же переменную, и как минимум одно из них меняет эту переменную.
- Если конфликтующие вычисления в разных потоках *не упорядочены*, поведение программы не определено.

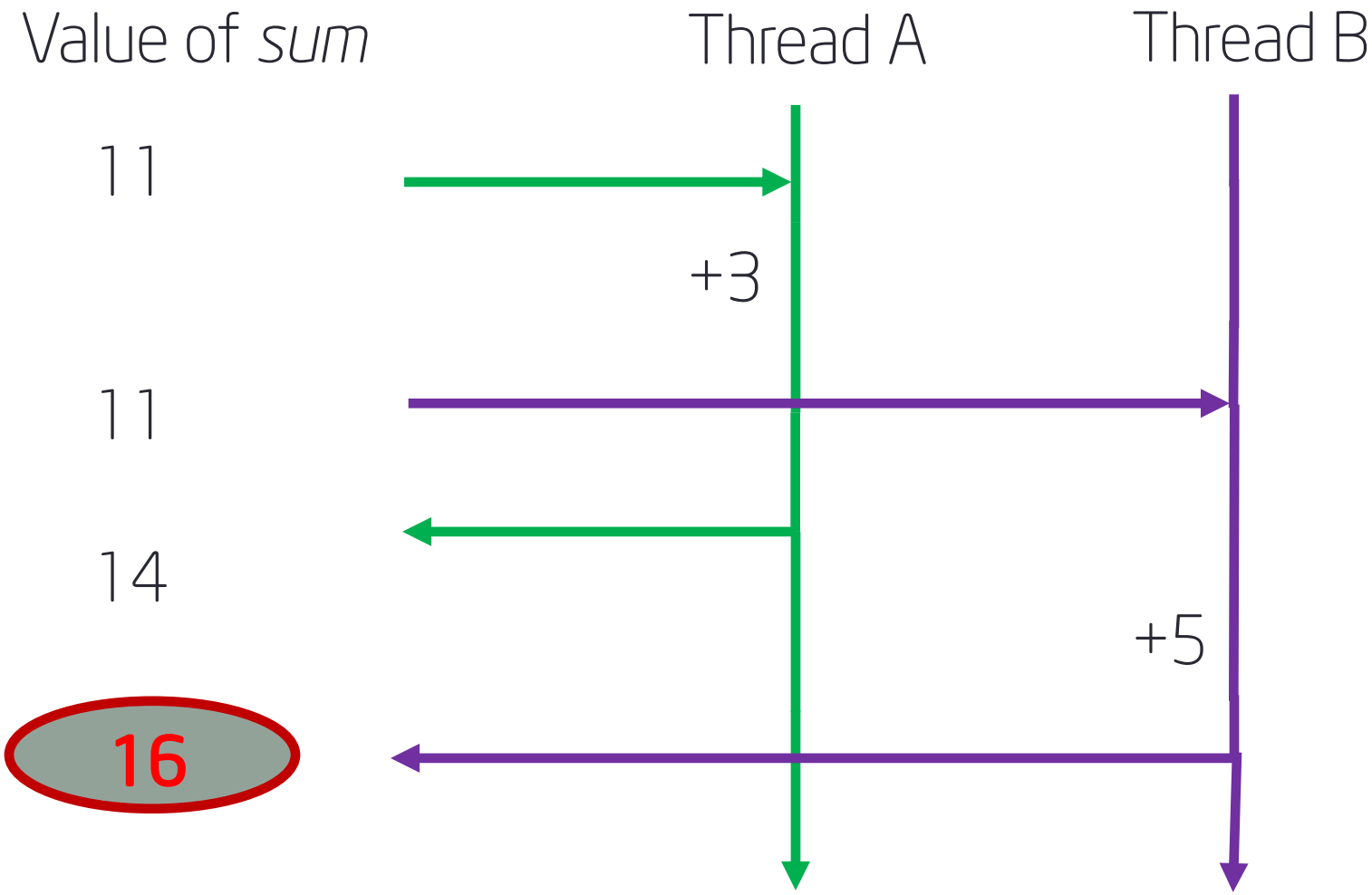
```
int sum, n;
...
for (int i = 0; i < n; i++) {
   sum += func(i);
}

// Что произойдёт, если цикл распараллелить?
```

Результат может быть корректным



... или некорректным



Поддержка в библиотеке C++11

- Потоки исполнения: `std::thread`
- Синхронизация: `std::mutex`, `std::lock_guard<>`, `std::condition_variable`
- Атомарные переменные: `std::atomic<>`
- Асинхронное исполнение: `std::future<>`, `std::async<>`

Базовые блоки для многопоточных программ

Основные операции с std::thread

Создать поток для исполнения функции

```
std::thread t( function, arguments... );
```

Дождаться завершения потока /*«присоединить»*/

```
t.join();
```

ВНИМАНИЕ:
нельзя вызывать деструктор для «неприсоединённого» потока

Пример с std::thread

```
#include <thread>
#include <iostream>

int main( int argc, const char* argv[] )
{
  std::thread t( [=]{
    for( int i=0; i<argc; ++i )
      std::cout << argv[i] << “\n”;
  });
  t.join();
  return 0;
}
```

Больше возможностей std::thread

Узнать доступное количество аппаратных потоков

```
std::thread::hardware_concurrency();
```

Приостановить исполнение текущего потока /*sleep*/

```
std::this_thread::sleep_for( duration );
std::this_thread::sleep_until( time_point );
```

Получить платформно-зависимый дескриптор

```
some_thread.native_handle();
```

Пример: пул потоков

```
#include <thread>
#include <vector>
#include <iostream>

void report( int i ) {
  std::cout << “Thread “ << i << “ was here\n”;
}

int main() {
  unsigned nthreads = std::thread::hardware_concurrency();
  std::vector<std::thread> threads;
  for( unsigned i=0; i<nthreads; ++i )
    threads.push_back( std::thread(report, i) );
  for( auto& t: threads )
    t.join();
  return 0;
}
```

Синхронизация доступа: std::mutex

Создать мьютекс

```
std::mutex m;
```

Получить исключительный доступ /*«захватить» мьютекс*/

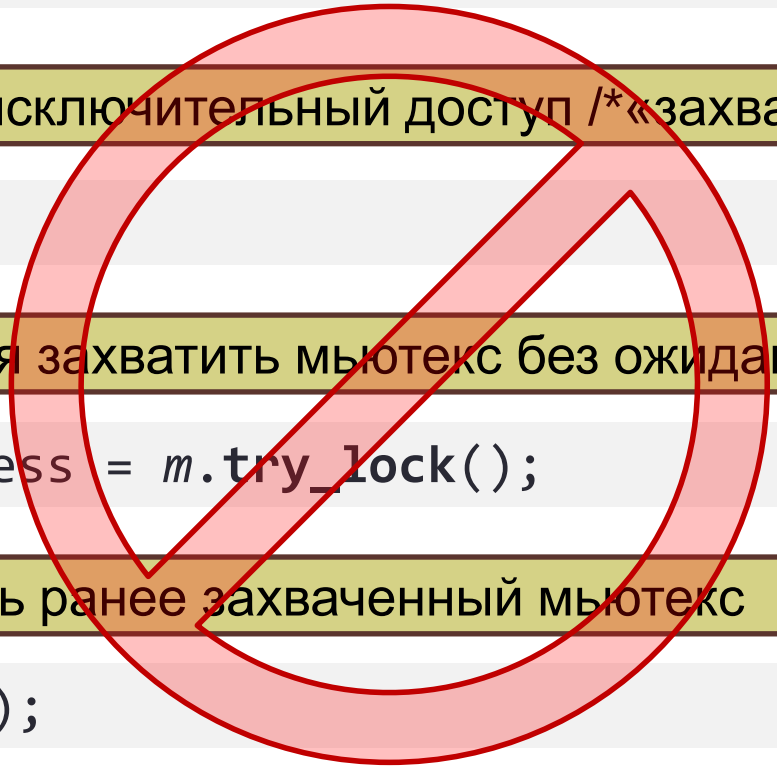
```
m.lock();
```

Попытаться захватить мьютекс без ожидания

```
bool success = m.try_lock();
```

Освободить ранее захваченный мьютекс

```
m.unlock();
```



Правильно: std::lock_guard<>

Создать мьютекс

```
std::mutex m;
```

Получить исключительный доступ /*«захватить» мьютекс*/

```
{ // критическая секция
    std::lock_guard<std::mutex> locked(m);
```

Освободить ранее захваченный мьютекс

```
} // автоматически при разрушении lock_guard
```

RAII: безопасно и просто!

Пример: синхронизация вывода

```
#include <mutex>
// ...

std::mutex report_mutex;
void report( int i ) {
  std::lock_guard<std::mutex> locked(report_mutex);
  std::cout << “Thread “ << i << “ was here\n”;
}

int main() {
  //...
}
```

Другие классы для синхронизации

- `std::recursive_mutex`
  - Можно захватывать повторно в том же потоке
- `std::timed_mutex`, `std::recursive_timed_mutex`
  - Поддерживают таймаут: `try_lock_{for,until}`
- `std::unique_lock<>`
  - Поддерживает RAII, как `std::lock_guard<>`
  - Предоставляет явные методы, как `std::mutex`
  - Можно использовать со всеми типами мьютексов
  - Требуется при использовании `std::condition_variable`

std::condition_variable

Создать условную переменную

```
std::condition_variable cvar;
```

Заблокировать поток в ожидании сигнала

```
std::unique_lock<std::mutex> locked(m);
cvar.wait(locked, condition_predicate);
```

Разблокировать один из ожидающих потоков

```
cvar.notify_one();
```

Разблокировать все ожидающие потоки

```
cvar.notify_all();
```

Образец: ожидание события

```
#include <mutex>
#include <condition_variable>

std::mutex mtx;
std::condition_variable condvar;
//...
std::unique_lock<std::mutex> locked(report_mutex);
while (!condition)
  condvar.wait(locked); // mtx свободен
// mtx захвачен


// Ещё проще
condvar.wait(locked, []{ return condition; });
```

ВНИМАНИЕ: возможен «случайный выход» /*spurious wakeup*/ из ожидания; поэтому нужен цикл while

Как это всё использовать для |||?

Необходимо разработать

- Управление потоками исполнения
- Распределение работы между потоками
- Синхронизацию при использовании разделяемых данных

Необходимо учесть

- Накладные расходы
- Балансировку нагрузки
- Масштабируемость
- Компонуемость составных частей программы

Общепринятое мнение:
Параллельные программы – это сложно

Questions | Tags | Users | Badges | Unanswered

C++ - Questions about multithreading

1. Do I need to query or even consider the number of cores on a machine or when the threads are running, they are automatically sent to free cores?

2. Can anyone sh

Assume we have an array or vector of length 256(can be more or less) and the number of pthreads to generate to be 4(can be more or less).

I need to figure out how to assign each pthread to a process a section

I have a large `for` loop, in which I want each item to be passed to a function on a thread. I have a thread pool of a certain size, and I want to reuse the thr

Basically, I want all the threads to wait for the READY signal before continuing. num_thread is set to 0, and READY is false before threads are created. Once in a while, deadlock occurs. Can anyone help

When should I allocate a new thread to the task?

I have one task to co
one array, and the se

In this simplified code, all the threads may try to write the exact same value to the same memomry location in `vec`. Is this a data race likely to trigger undefined behavior, or is it safe since the values are never read before all the threads are joined again?

Concurrency != Parallelism

Concurrency

- **Цель:** эффективная *организация программы* через взаимодействие отдельных компонент
- **Способ:** взаимодействующие *потоки*, синхронизация, ожидание и обработка событий

Параллелизм

- **Цель:** эффективное *использование «железа»* и масштабируемая *производительность*
- **Способ:** независимые *задачи,* которые можно исполнять одновременно на разных потоках

Потоки можно использовать по-разному

Аналогия из спорта

Photo credit JJ Harrison (CC) BY-SA 3.0

Concurrency

Photo credit André Zehetbauer (CC) BY-SA 2.0

Параллелизм

C++11 поддерживает concurrency, но не параллелизм

```
// GOOD IDEA
std::thread work_thread(computeFunc);
event_loop();
work_thread.join();
```

```
// BAD IDEA
std::thread child([=]{ parallel_qsort(begin, middle, comp); });
parallel_qsort(middle+1, end, comp);
child.join();
```

```
// BAD IDEA
auto fut = std::async([=]{ parallel_qsort(begin, middle, comp); });
parallel_qsort(middle+1, end, comp);
fut.wait();
```

Многопоточность – это сложно

«Кто виноват?»	«Что делать?»
☑ Слишком низкий уровень абстракции	☐
☑ Отсутствие необходимых знаний и опыта	☐
☑ Некоторые концепции **действительно** сложны!	☐

Источники информации

Working Draft, Standard for Programming Language C++ (N3337, 2012-01-16)

http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2012/n3337.pdf

Anthony Williams et al:

серия интернет-статей *Multithreading and Concurrency*
https://www.justsoftwaresolutions.co.uk/threading/

Материалы конференции CppCon 2014

https://github.com/CppCon/CppCon2014/tree/master/Presentations
(в частности, доклад Пабло Халперна “Overview of Parallel Programming in C++”)

Домашнее задание

1. Напишите многопоточную программу для подсчёта кол-ва слов в тексте.
   - Используйте подход «производитель-потребитель» /*producer-consumer*/.
   - Используйте классы стандартной библиотеки: `queue, thread, mutex, condition_variable`

2. Напишите параллельную программу для вычисления скалярного произведения двух векторов большого размера.
   - ```
     double scalar_product( const double* v1, const double* v2, int n )
     {
         double result = 0.0;
         for( int i=0; i<n; ++i ) result += v1[i]*v2[i];
         return result;
     }
     ```
     ```

Продолжение следует...