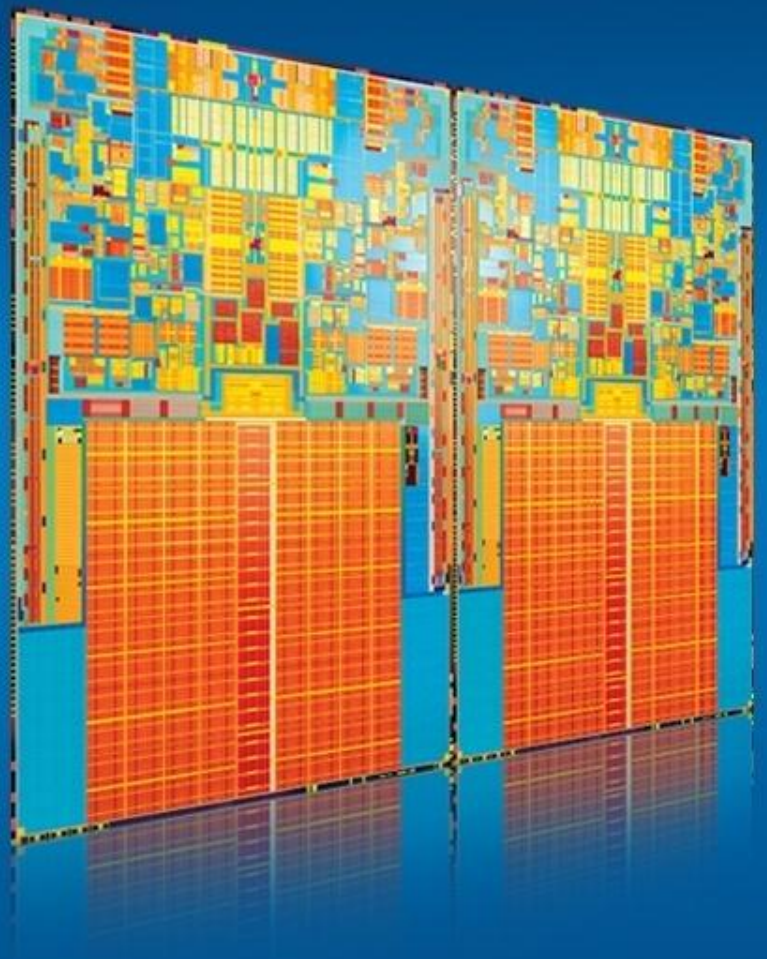


Модели параллельного программирования Intel

Кирилл Рогожин
Технический консультант
kirill.rogozhin@intel.com

intel Software

Содержание

- **Обзор программных моделей**
- **OpenMP***
- **Intel® Threading Building Blocks**
- **Intel® Cilk™ Plus**
  - **cilk_spawn и cilk_sync**
  - **cilk_for**
  - **Reducers**
  - **Векторизация**
- **Заключение**

Модели параллельного программирования

Intel® Cilk™ Plus	Intel® Threading Building Blocks	Специализированные библиотеки	Поддержка стандартов	Экспериментальные технологии
Расширение C/C++	Библиотека шаблонов C++	Intel® Integrated Performance Primitives	Message Passing Interface (MPI)	Intel® Concurrent Collections
Простой параллелизм	Открытый код	Intel® Math Kernel Library	OpenMP*	Offload Extensions
Открытый код	Продукт Интел		Coarray Fortran	Intel® Array Building Blocks
Продукт Интел			OpenCL*	Intel® SPMD Parallel Compiler

Содержание

- Обзор программных моделей
- OpenMP
- Intel® Threading Building Blocks
- Intel® Cilk™ Plus
  - cilk_spawn и cilk_sync
  - cilk_for
  - Reducers
  - Векторизация
- Заключение

OpenMP

http://www.openmp.org

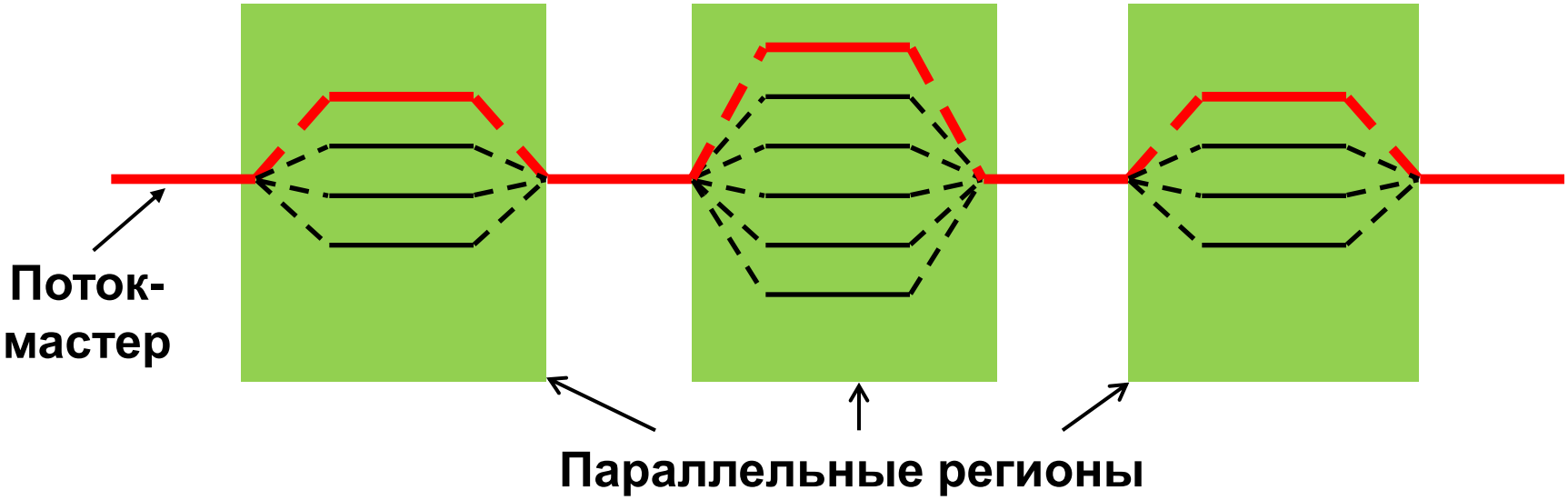
Current spec is OpenMP 3.1

- Переносимый API для систем с общей памятью
  - Fortran, C, и C++
  - Поддерживается многими производителями
- Linux, Windows, Mac OS
- Стандартизирует распараллеливание циклов и по задачам
- Объединяет последовательный и параллельный код в одном исходнике
- Стандартизирует ~ 20 летний опыт многопоточности в компиляторах

Программная модель

Fork-Join параллелизм:

- **Поток-мастер** запускает **группу потоков**
- **Параллелизм добавляется последовательно: последовательная программа эволюционирует в параллельную**



Как выглядит OpenMP

- Опция компилятора /Qopenmp в Windows
  или –openmp в Linux
- Директивы компилятора, pragma
  - Вид директив в C and C++:

```
#pragma omp construct [clause [clause]…]
```

  - Формы директив в Fortran:

```
C$OMP construct [clause [clause]…]
!$OMP construct [clause [clause]…]
*$OMP construct [clause [clause]…]
```

- API функции

```
omp_get_num_threads()
```

- Переменные окружения

```
omp_get_num_threads()
OMP_NUM_THREADS
```

Параллельный регион и структурные блоки (C/C++)

- Большинство конструкций OpenMP применяются к структурному блоку
  - Структурный блок: блок с одной точкой входа и одной точкой выхода
  - Единственное разрешённое «ветвление» - оператор STOP в Fortran и exit() в C/C++

```
#pragma omp parallel
{
  int id = omp_get_thread_num();

more: res[id] = do_big_job (id);

  if (conv (res[id])) goto more;
}
printf ("All done\n");
```

Структурный блок

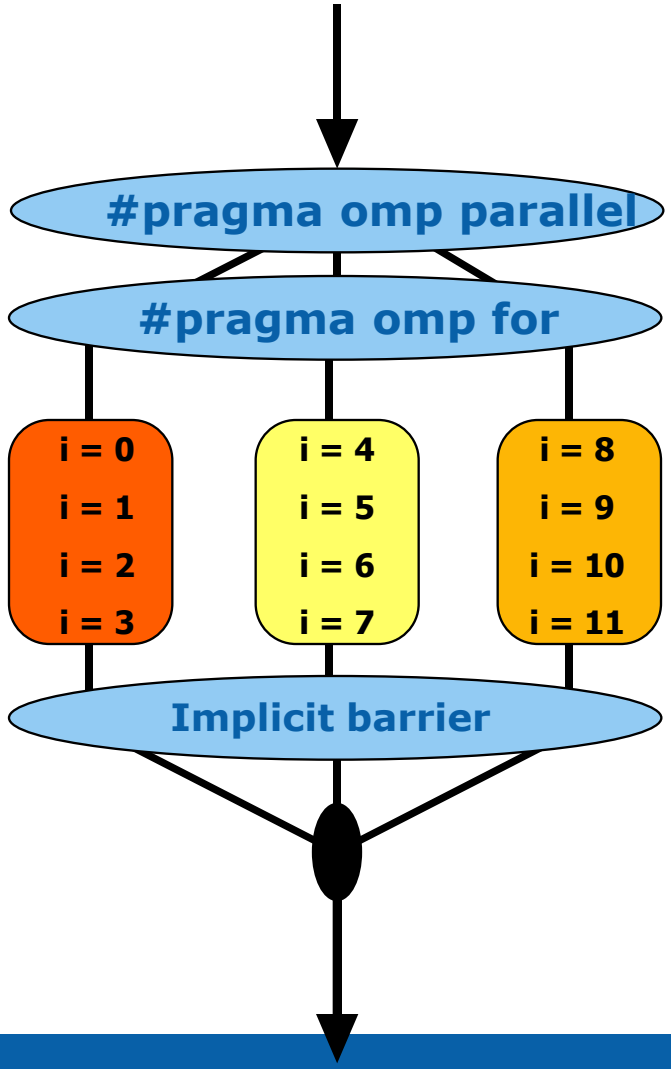
```
if (go_now()) goto more;
#pragma omp parallel
{
  int id = omp_get_thread_num();
more:  res[id] = do_big_job(id);
  if (conv (res[id])) goto done;
  goto more;
}
done: if (!really_done()) goto more;
```

Не структурный блок

Конструкция omp for

```
// assume N=12
#pragma omp parallel for
for(i = 0; i < N; i++)
    c[i] = a[i] + b[i];
```

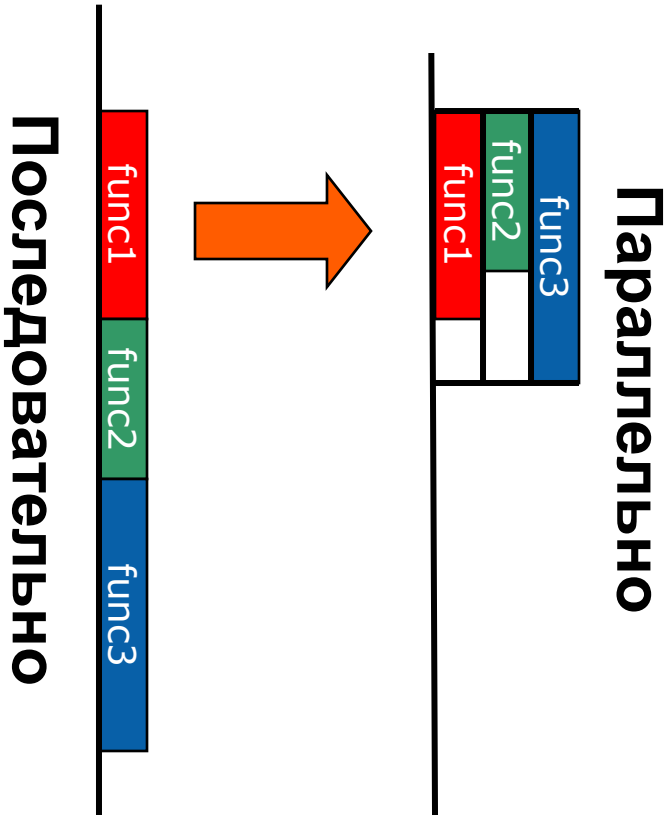
- Потокам назначаются независимые наборы итераций
- Потоки должны ждать в конце параллельной конструкции



Параллельные секции

- Независимые секции кода могут выполняться параллельно – уменьшается время исполнения

```
#pragma omp parallel sections
{
   #pragma omp section
   func1();
   #pragma omp section
   func2();
   #pragma omp section
   func3();
}
```



Что ещё есть в OpenMP

- Вложенный параллелизм
- Организация доступа к общим данным
- Задачи (tasks)
- Средства синхронизации
  - Атомарные операции
  - Замки
  - Критические секции
- Управляющие директивы и API

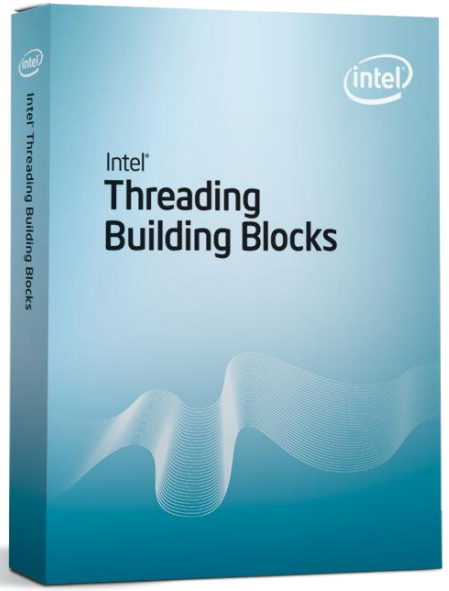
Содержание

- Обзор программных моделей
- OpenMP
- Intel® Threading Building Blocks
- Intel® Cilk™ Plus
  - cilk_for
  - cilk_spawn и cilk_sync
  - Reducers
  - Векторизация
- Заключение

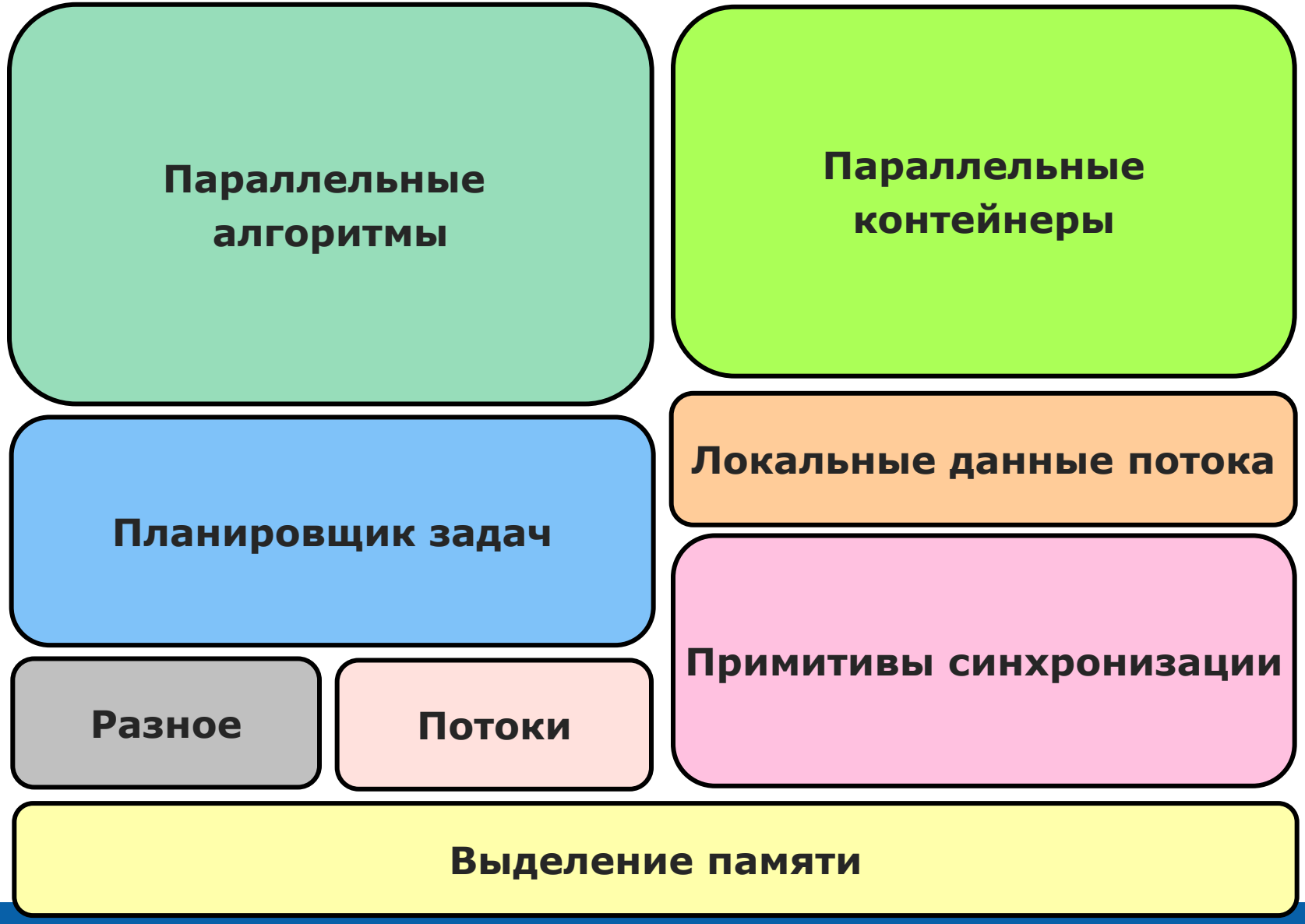
Intel® Threading Building Blocks

- **Библиотека C++ для распараллеливания кода №1**
  - Берёт на себя управление многозадачностью
- **Runtime библиотека**
  - Масштабируемость на доступное число потоков
- **Кросс-платформенность**
  - Windows, Linux, Mac OS* и другие
- **Коммерческая и открытая лицензии**

http://threadingbuildingblocks.org

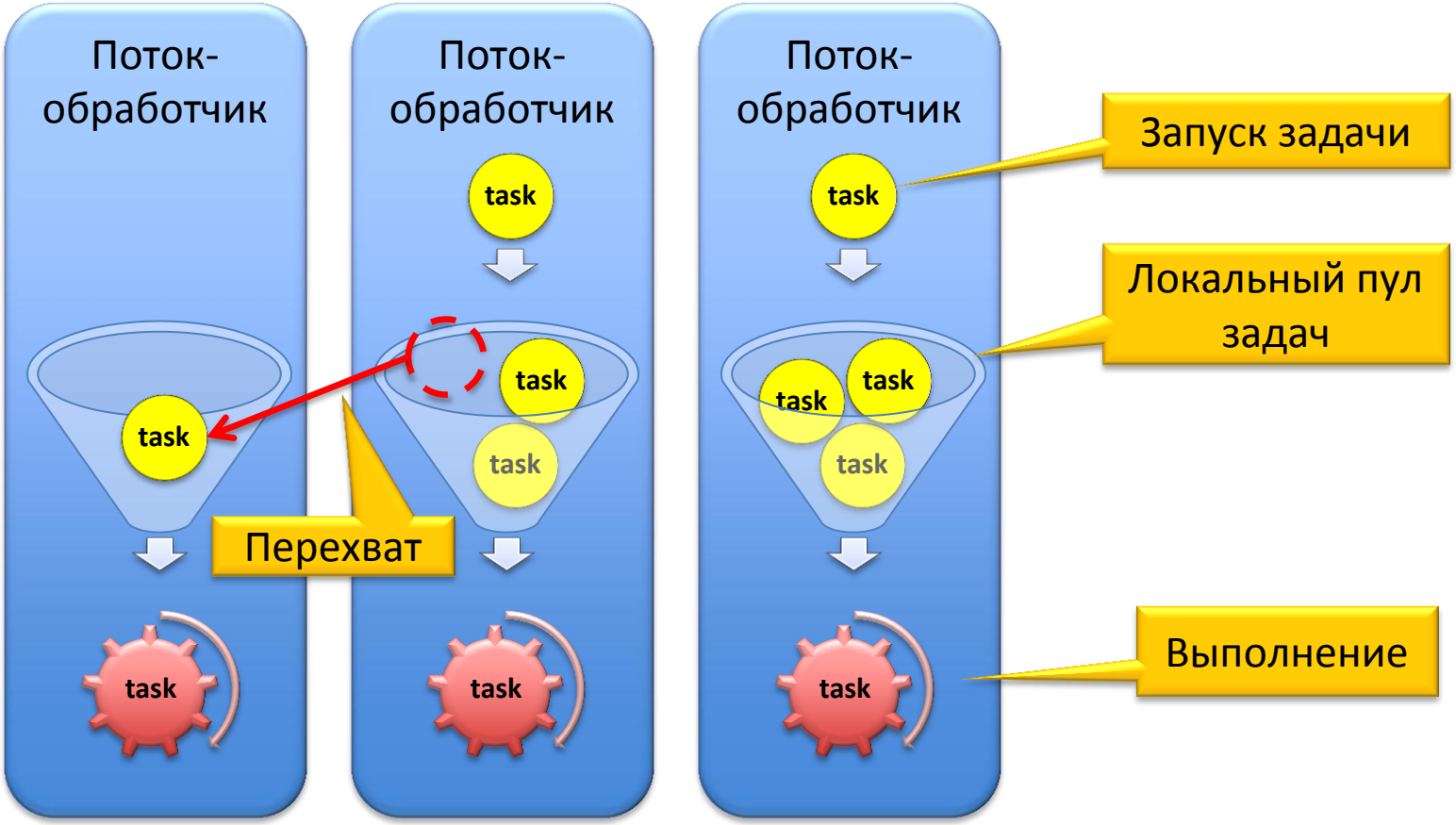


Intel® Threading Building Blocks



Исполнение задач

Задачи позволяют балансировать нагрузку автоматически



Параллельные алгоритмы

Параллельные циклы

parallel_for
parallel_reduce

Алгоритмы для потоков данных

parallel_do
parallel_for_each
pipeline / parallel_pipeline

Параллельная сортировка

parallel_sort

Запуск параллельных функций

parallel_invoke

Вычислительный граф

flow::graph

Пример использования parallel_for

```
class ChangeArray{
    int* array;
public:
    ChangeArray (int* a): array(a) {}
    void operator()( const blocked_range<int>& r ) const{
        for (int i=r.begin(); i!=r.end(); i++ ){
            Foo (array[i]);
        }
    }
};

parallel cycle(){
    parallel_for (0, n, 1), ChangeArray(a));
}

serial_loop(){
     for (int i=0; i<n; i++)
          Foo (array[i]);
}
```

Класс ChangeArray определяет тело цикла для parallel_for

blocked_range – шаблон TBB, представляющий одномерное итерационное пространство

Основная работа находится в операторе «operator()». Это типично для функциональных объектов C++

Вызов шаблонной функции parallel_for<Range, Body>:
Аргументы:
Range → blocked_range
Body → ChangeArray

Поддержка лямбда-функций C++ 11

Предыдущий пример parallel_for можно записать иначе:

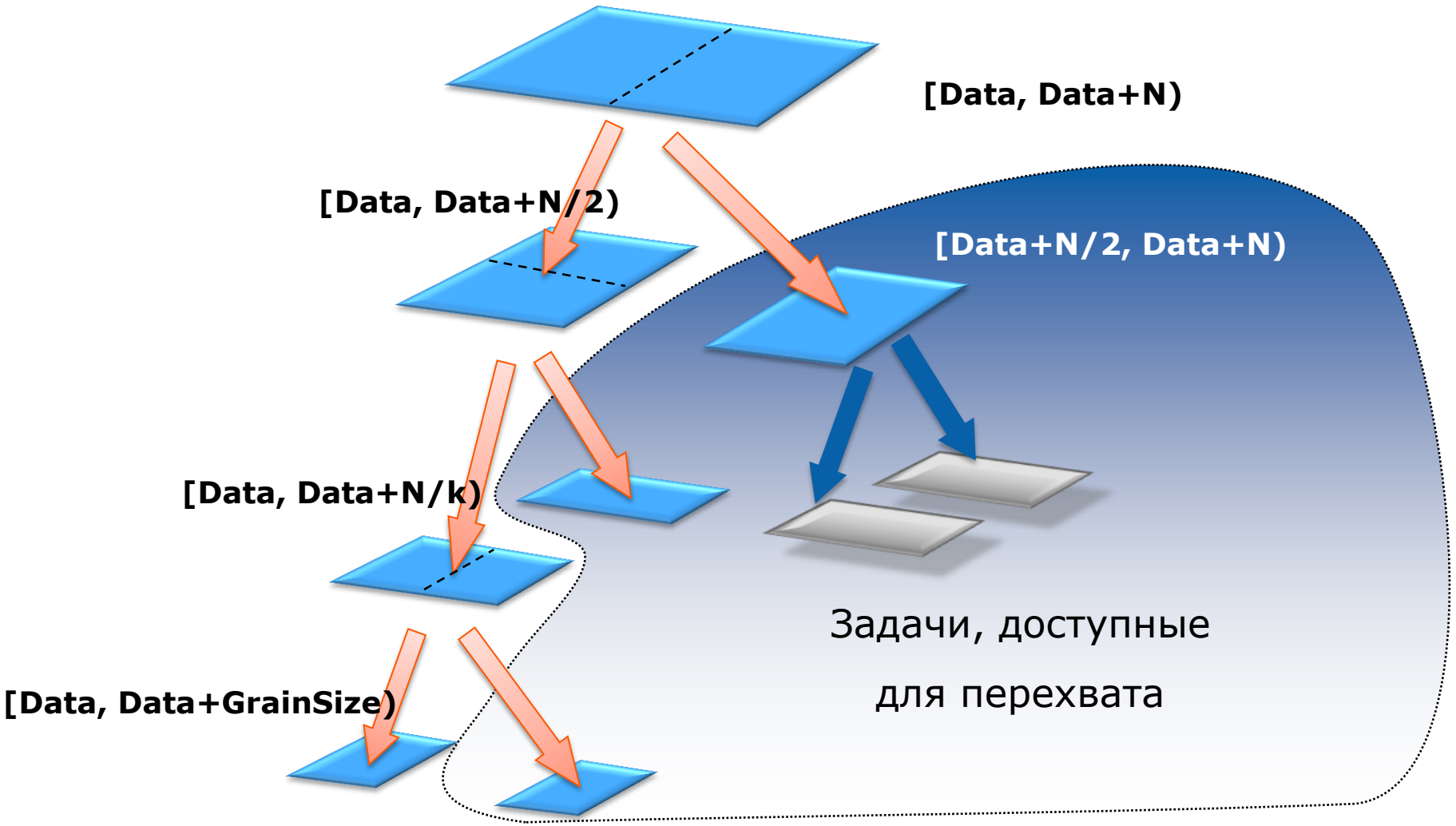
```
#include "tbb/blocked_range.h"
#include "tbb/parallel_for.h"
using namespace tbb;

int main (){
    int a[n];
    // initialize array here…

    parallel_for (0, n, 1,
        [=](int i) {
          Foo (a[i]);
        });
    return 0;
}
```

Перегруженный parallel_for использует аргументы start, stop и step и сам конструирует blocked_range

Передача параметров по значению работает так же, как и при обычном вызове функции. Если использовать [&], параметры передаются по ссылке

С использованием лямбда-функций MyBody::operator() реализуется прямо внутри вызова parallel_for().

parallel_for (Range(Data), Body(), Partitioner());



Основные параллельные контейнеры

concurrent_hash_map <Key, T, Hasher, Allocator>

concurrent_unordered_map<Key, T, Hasher, Equality, Allocator>

concurrent_vector <T, Allocator>

concurrent_queue <T, Allocator>

concurrent_bounded_queue <T, Allocator>

Атомарные операции

```
#include "tbb/blocked_range.h"
#include "tbb/parallel_for.h"
using namespace tbb;

atomic<int> sum;

int main (){
    int a[n];
    // initialize array here…

    parallel_for (0, n, 1,
        [=](int i) {
            Foo (a[i]);
            sum += a[i];
        });
return 0;
}
```

Эта операция выполняется атомарно

Содержание

- Обзор программных моделей
- OpenMP
- Intel® Threading Building Blocks
- **Intel® Cilk™ Plus**
  - cilk_spawn и cilk_sync
  - cilk_for
  - Reducers
  - Векторизация
- Заключение

Что такое Cilk?
Intel® Cilk™ Plus

- Расширение C/C++ для распараллеливания программ

- 3 ключевых слова
  - cilk_for
  - cilk_spawn
  - cilk_sync

- Указание мест для распараллеливания, а не управление потоками

Содержание

- Обзор программных моделей
- OpenMP
- Intel® Threading Building Blocks
- Intel® Cilk™ Plus
  - cilk_spawn и cilk_sync
  - cilk_for
  - Reducers
  - Векторизация
- Заключение

Последовательное выполнение

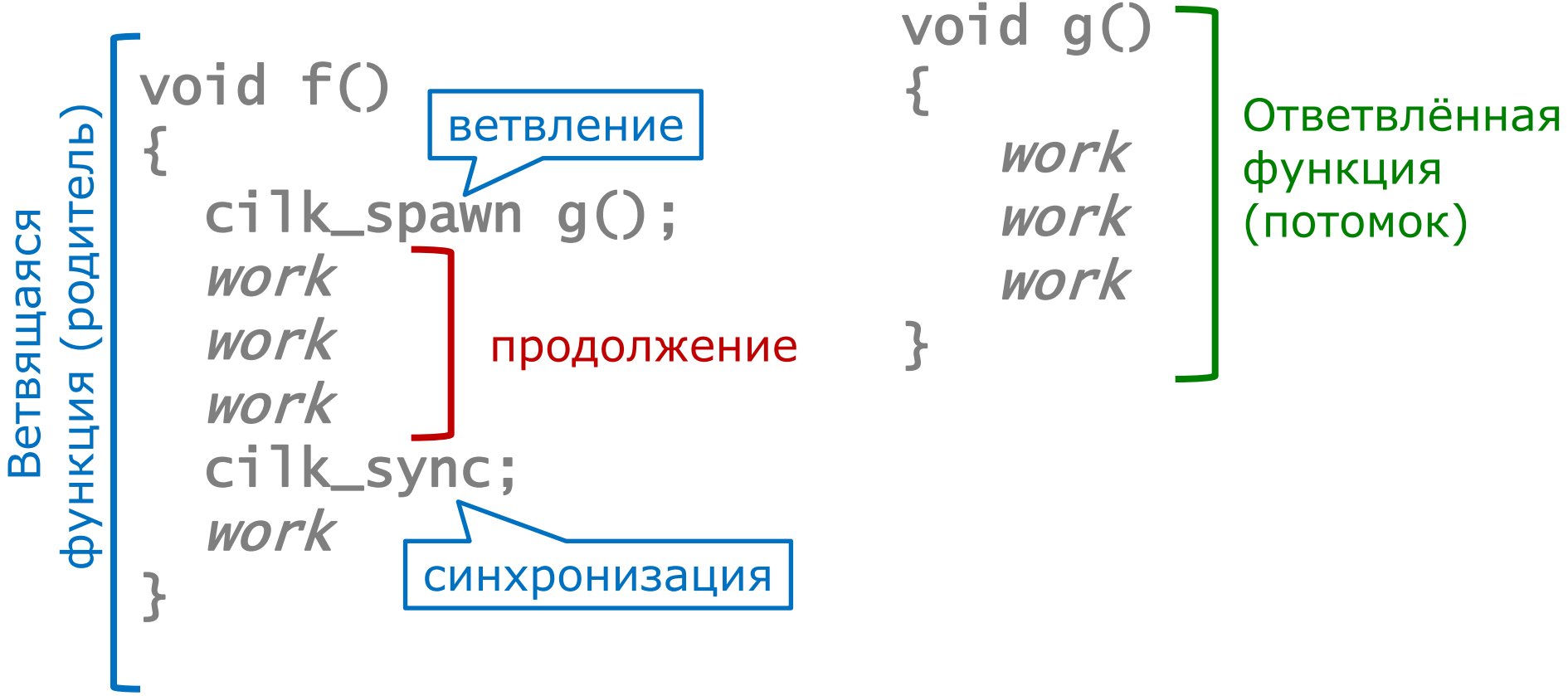
Intel® Cilk™ Plus

```
void f()
{
  g();
  work
  work
  work

  work
}
```

```
void g()
{
   work
   work
   work
}
```

Анатомия ветвления

Intel® Cilk™ Plus



Захват задачи
другие обработчики недоступны

```
void f()
{
  cilk_spawn g();
  work
  work
  work
  cilk_sync;
  work
}
```

```
void g()
{
   work
   work
   work
}
```

Обработчик A

То же поведение, что и при последовательном исполнении!

Захват задачи
доступны другие обработчики

```
void f()
{
  cilk_spawn g();
  work
  work
  work
  cilk_sync;
  work
}
```


```
void g()
{
   work
   work
   work
}
```

Обработчик A

Обработчик B

Обработчик A/B

Накладные расходы при захвате задач

Intel® Cilk™ Plus

- Ветвление не затратно
- Захват гораздо более затратен (требуются блокировки и барьеры памяти)
- Большинство ветвлений не приводит к захватам.
- Сбалансированное распределение нагрузки ➔ меньше захватов ➔меньше накладных расходов

Содержание

- Обзор программных моделей
- OpenMP
- Intel® Threading Building Blocks
- Intel® Cilk™ Plus
  - cilk_spawn и cilk_sync
  - cilk_for
  - Reducers
  - Векторизация
- Заключение

Цикл cilk_for

Intel® Cilk™ Plus

Возможности:

- Выглядит как обычный цикл for:

```
cilk_for (int x = 0; x < 1000000; ++x) { … }
```

- Любая итерация может выполняться параллельно с другими
- Все итерации заканчивают выполнение до выхода из цикла

Требования:

- Только одна контрольная переменная
- Возможность перейти к началу любой итерации
- Итерации должны быть независимы

Примеры cilk_for

Intel® Cilk™ Plus

```
cilk_for (int x; x < 1000000; x += 2) { … }
```

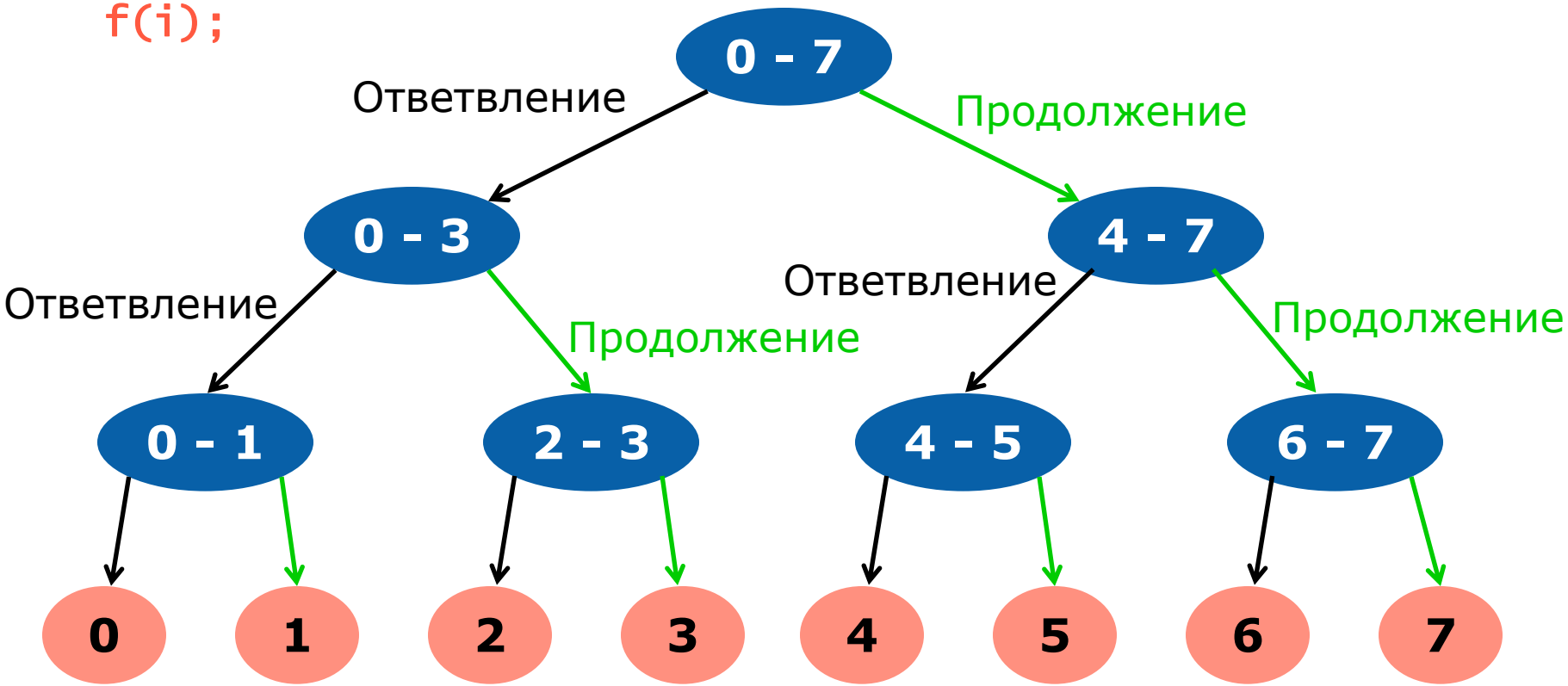
```
cilk_for (vector<int>::iterator x = y.begin();
          x != y.end(); ++x) { … }
```

```
cilk_for (list<int>::iterator x = y.begin();
          x != y.end(); ++x) { … }
```

Реализация cilk_for

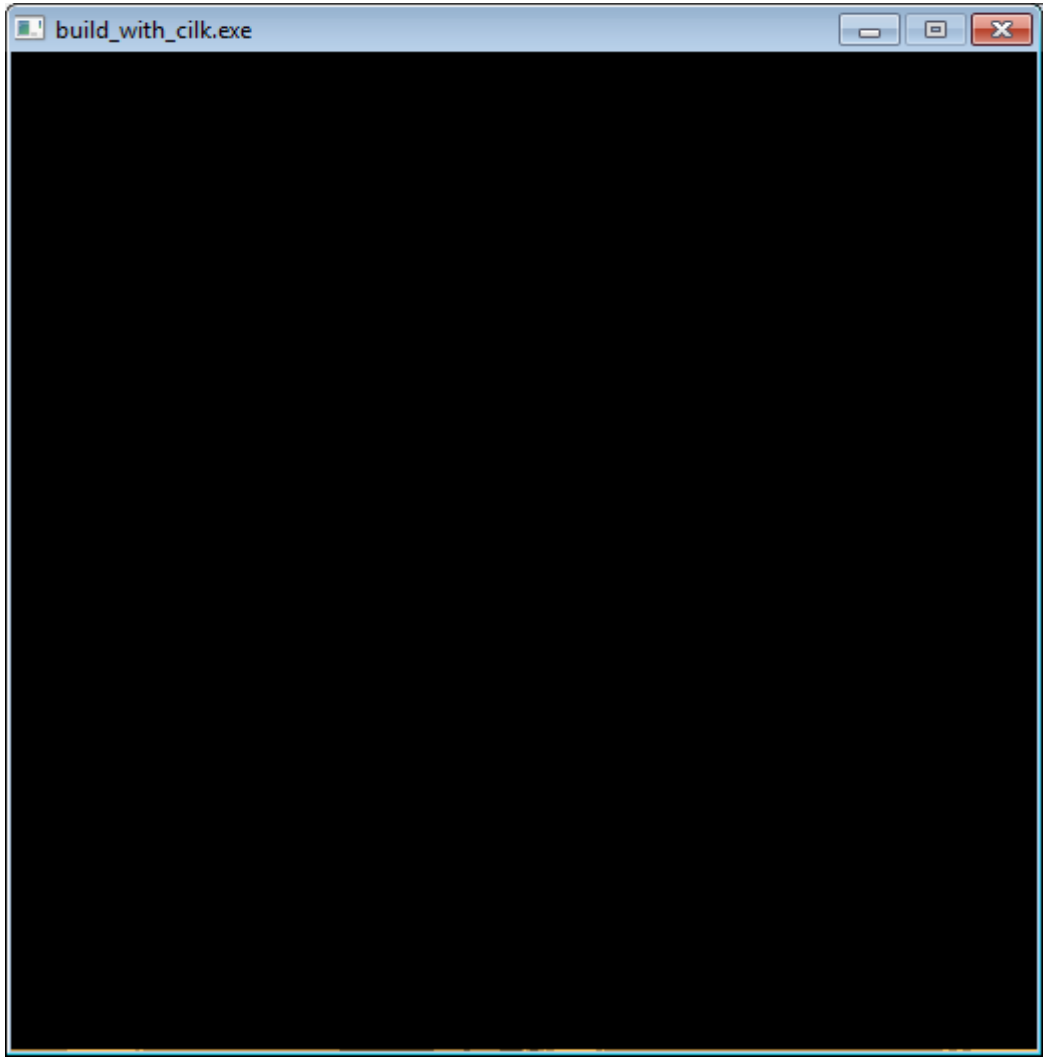
Intel® Cilk™ Plus

```
cilk_for (int i=0; i< 8; ++i)
    f(i);
```



Использование `cilk_for`

Intel® Cilk™ Plus



cilk_for и последовательный for с ветвлением

- Сравните циклы:

```
for (int x = 0; x < n; ++x) { cilk_spawn f(x); }

cilk_for (int x = 0; x < n; ++x) { f(x); }
```

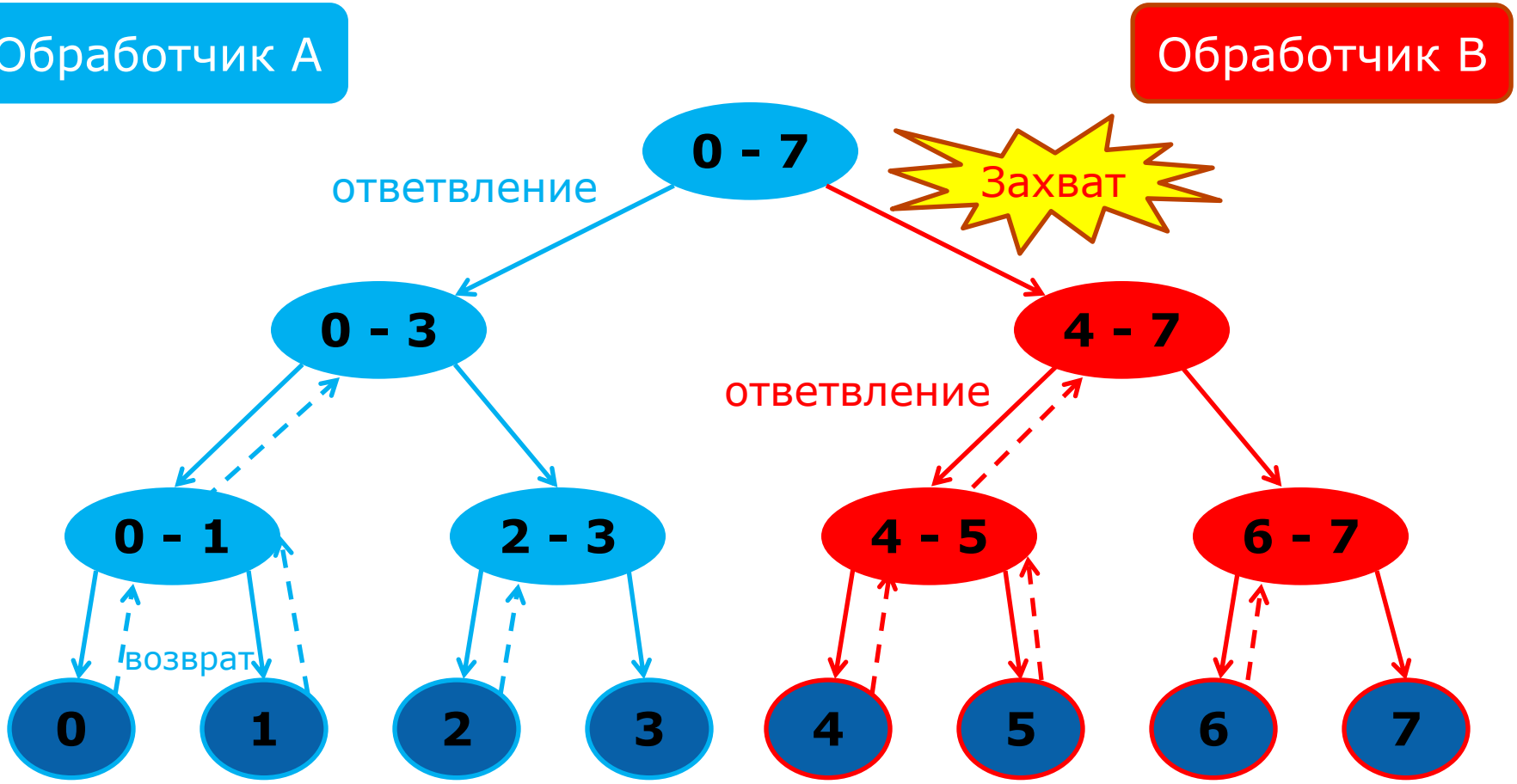
- Два цикла имеют схожую семантику, но…
- их производительность сильно отличается

Последовательный for с ветвлением: не сбалансирован

Если работа
за одну итерацию мала,
накладные расходы
могут быть значительны

cilk_for: Divide and Conquer

Intel® Cilk™ Plus



Divide and conquer results in few steals and less overhead.

Содержание

- Обзор программных моделей
- OpenMP
- Intel® Threading Building Blocks
- Intel® Cilk™ Plus
  - cilk_spawn и cilk_sync
  - cilk_for
  - Reducers
  - Векторизация
- Заключение

Reducers

Без захвата

```
cilk::reducer_opadd<int> sum(3);
```

3

Начальное значение

```
void f()
{
  cilk_spawn g();
  work
  sum += 2;
  work
  cilk_sync;
  work
}
```

```
void g()
{
    work
    sum++;
    work
}
```

4

6

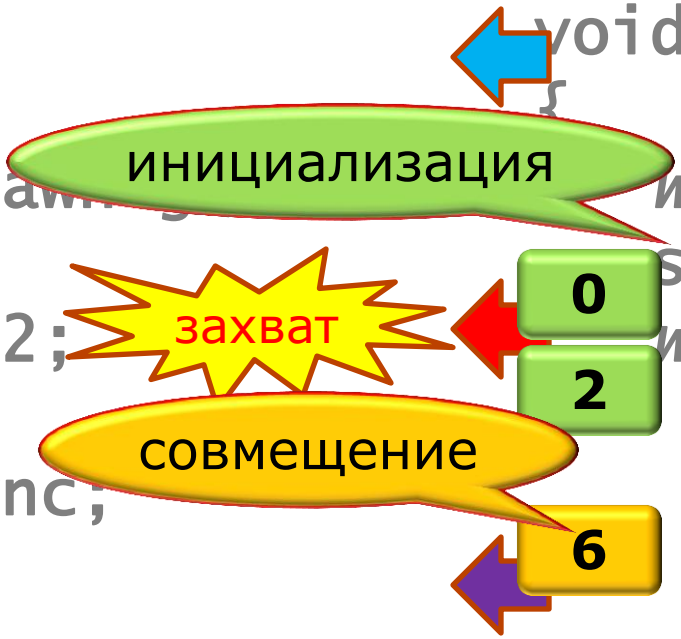
Обработчик A

То же поведение, что и при последовательном исполнении!

Reducers

продолжение захвачено

```
cilk::reducer_opadd<int> sum(3);

void f()
{
  cilk_spawn g();
  work
  sum += 2;
  work
  cilk_sync;
  work
}

void g()
{
  work
  sum++;
  work
}
```



3

Начальное значение

4

0

2

6

Обработчик A

Обработчик B

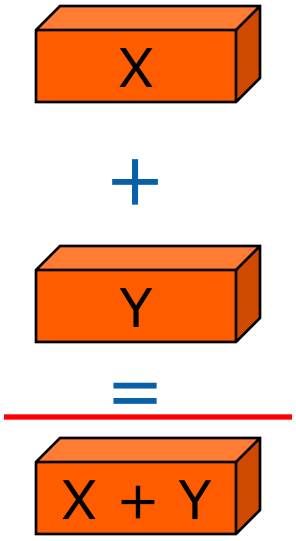
Обработчик A/B

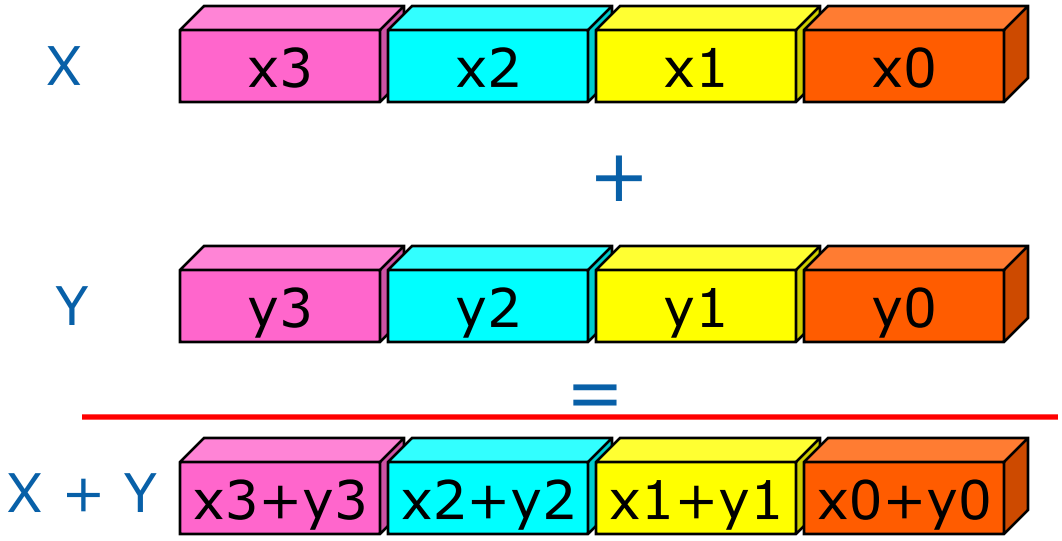
Содержание

- Обзор программных моделей
- OpenMP
- Intel® Threading Building Blocks
- Intel® Cilk™ Plus
  - cilk_spawn и cilk_sync
  - cilk_for
  - Reducers
  - Векторизация
- Заключение

SIMD: Single Instruction Multiple Data

- **Скалярная обработка**
  - Традиционный режим
  - Одна операция, один результат
- **SIMD обработка**
  - Одна операция, множественный результат





C/C++ Extensions for Array Notation (CEAN)

- Обозначение массивов

  <array base> [<lower_bound> : <length> : <stride>]*

  - ‘:<stride>’ опционально (по умолчанию stride=1)
  - отсутствие ‘:<length>:<stride>’ означает length=1
  - Просто ‘:’ выбирает все элементы в данном измерении
  - Обратите внимание на отличие от Фортрана, где:

    lower_bound : upper_bound : [stride]

- Примеры:

```
A[:]          // Все элементы вектора A
B[2:6]        // Элементы со 2 по 7 вектора B
C[:][5]       // Столбец 5 матрицы C
D[0:3:2]      // Элементы 0,2,4 вектора D
E[0:3][0:4]   // 12 элементов с E[0][0] по E[2][3]
```

Условные выражения

- Выражения CEAN могут использоваться в условных выражениях.
  - Блоки if/else должны иметь один вход и один выход. Использование операторов goto, break и return не допускается. Переходы в блок if/else запрещены.

```
if (a[:] < b[:]) {
    mask[:] = -1;
}
else {
    mask[:] = 1;
}
// mask[n] содержит -1 если a[n] < b[n], 1 если
  a[n] >= b[n]
mask[:] = a[:] < b[:] ? -1 : 1;
```

Содержание

- Обзор программных моделей
- OpenMP
- Intel® Threading Building Blocks
- Intel® Cilk™ Plus
  - cilk_spawn и cilk_sync
  - cilk_for
  - Reducers
  - Векторизация
- Заключение

Заключение

- **OpenMP** – признанный стандарт для распараллеливания кода компилятором
- **Intel® TBB** – мощная библиотека C++, автоматическая балансировка нагрузки
- **Intel® Cilk™ Plus** - простое расширение C/C++ для распараллеливания программ

Optimization Notice

Уведомление об оптимизации

Компиляторы Intel могут не обеспечивать для процессоров других производителей такой же уровень оптимизации для оптимизаций, которые не являются присущими только процессорам Intel. В число этих оптимизаций входят наборы команд SSE2, SSE3 и SSSE3, а также другие оптимизации. Корпорация Intel не гарантирует наличие, функциональность или эффективность оптимизаций микропроцессоров других производителей. Содержащиеся в данной продукции оптимизации, зависящие от микропроцессора, предназначены для использования с микропроцессорами Intel. Некоторые оптимизации, не характерные для микроархитектуры Intel, резервируются только для микропроцессоров Intel. Более подробную информацию о конкретных наборах команд, покрываемых настоящим уведомлением, можно получить в соответствующих руководствах пользователя и справочниках на продукт.

Уведомление, редакция № 20110804

Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Cilk, Core Inside, FlashFile, i960, InstantIP, Intel, the Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skoool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

http://intel.com/software/products