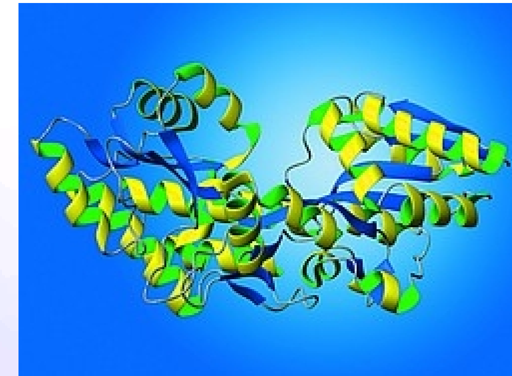
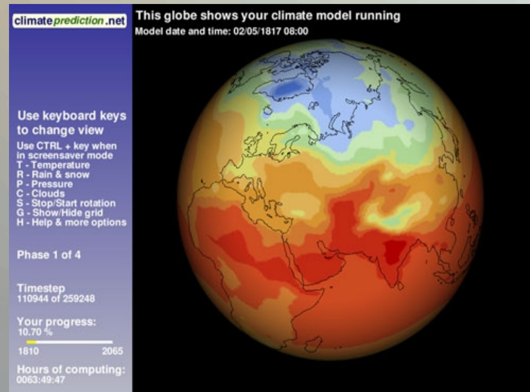
Высокопроизводительные вычисления

Калишенко Е.Л.
Академический Университет

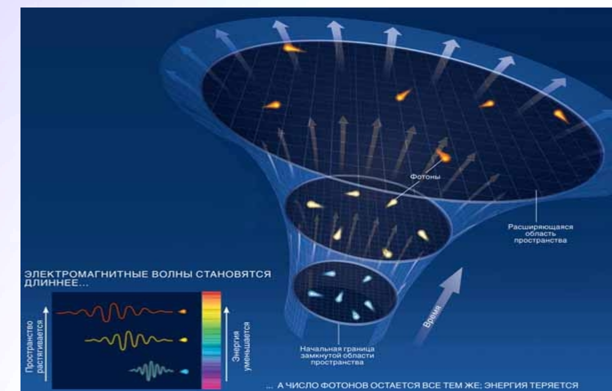
Мотивация

- *Генетика и протеомика*





- *Климатология*



- *Физика высоких энергий*
- *Астрономия, банковские транзакции...*

История (кратко)

Год	*Событие*
1958	IBM 709 – независимые процессоры ввода/вывода
...	
2002	Intel Hyper-Trading
2005	Intel Pentium 4D / AMD Athlon X2
2006	Intel Core 2 Duo
2008	Intel Core Quad, Nehalem (8/16) + SSE 4.2
...	

Содержание

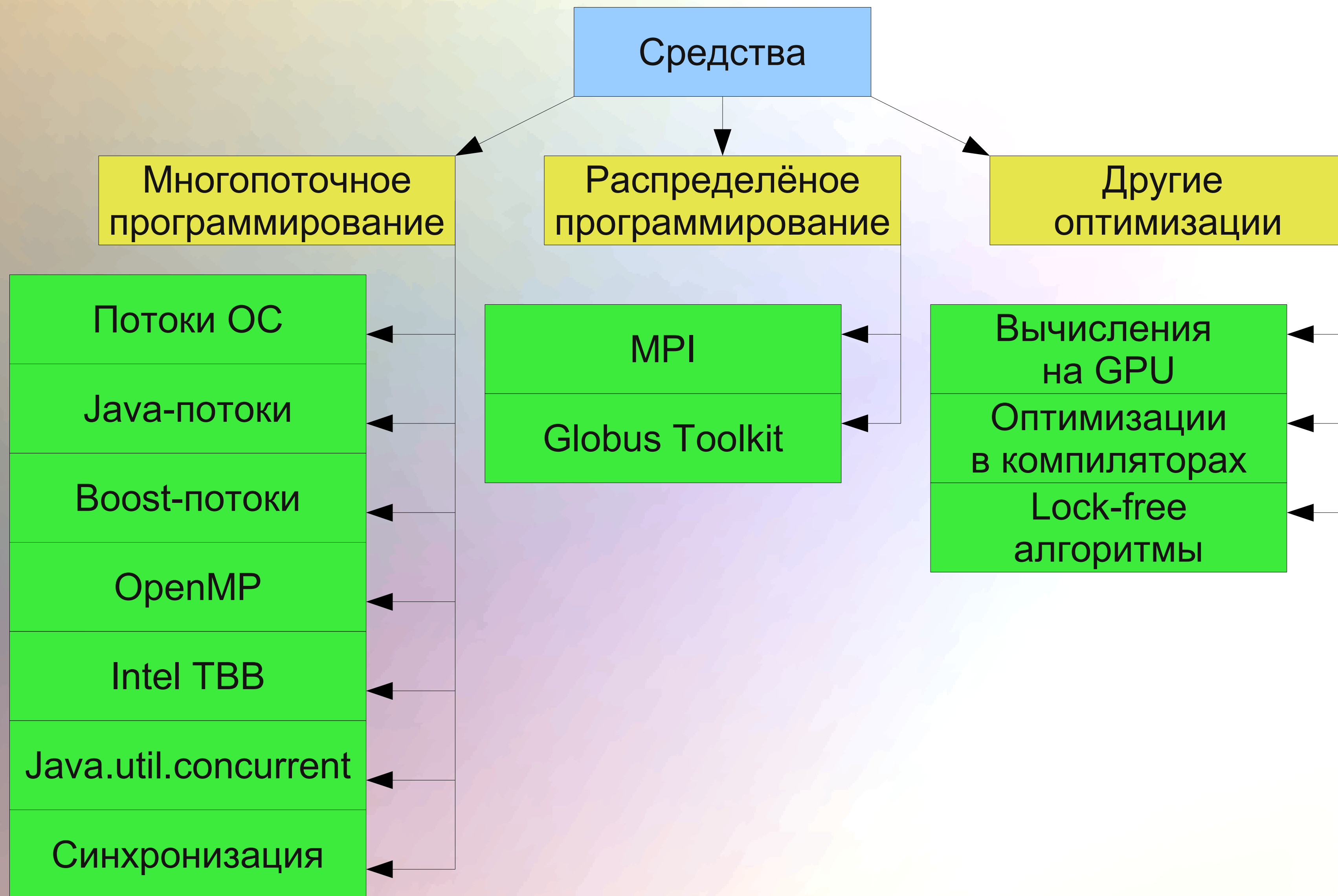


Схема средств

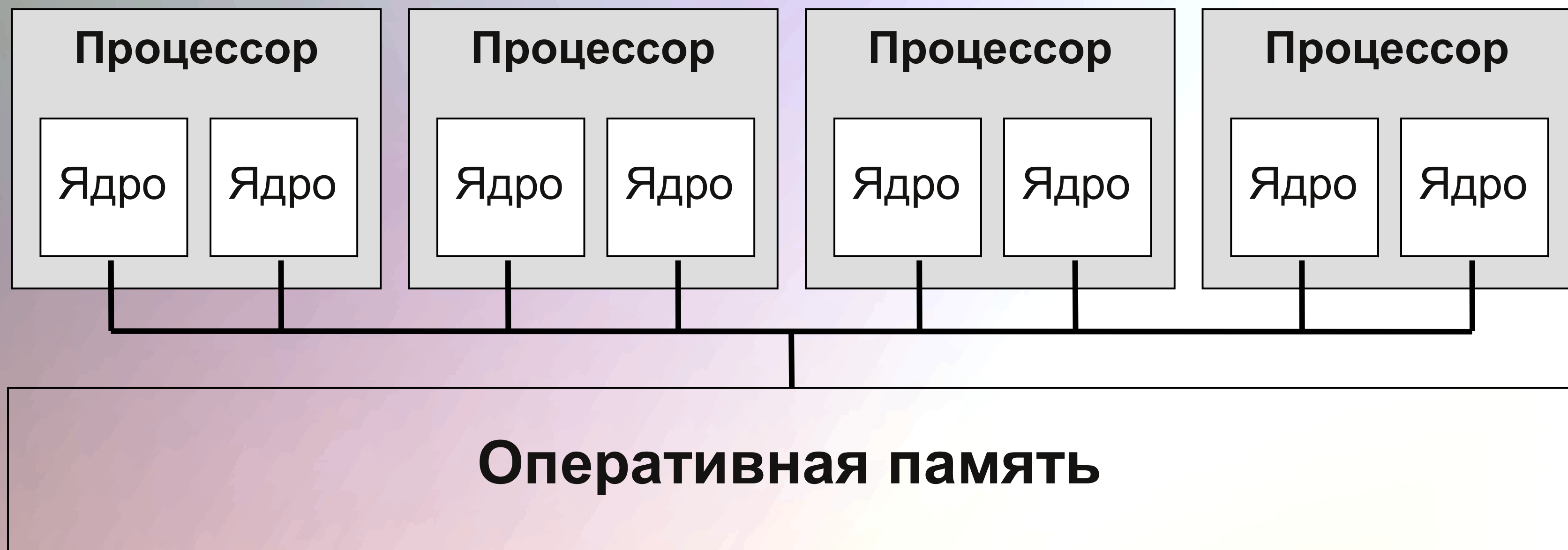
// todo нарисовать нормальную схему

- Пока смотрим на доску :)

Обращаем внимание...

1. Ускорение и Масштабируемость
2. Закон Амдала:
   - a – доля последовательного кода,
   - p – число процессоров

$$S = \frac{1}{a + \frac{(1-a)}{p}};$$



SSE (Streaming SIMD Extensions)

Версия	Возможности
1	Восемь 128-битных регистров для 4 чисел по 32 бит (с плавающей точкой)
2	Теперь два 64-битных числа в регистре
3	Уже 13 инструкций (+горизонтальная работа с регистрами)
4.1	47 инструкций (ускорение видео)
4.2	54 инструкции (операции со строками)

```
float a[4] = { 300.0, 4.0, 4.0, 12.0 };
float b[4] = {   1.5, 2.5, 3.5,  4.5 };

__asm {
  movups xmm0, a    ; // поместить из a в регистр xmm0
  movups xmm1, b    ; // поместить из b в регистр xmm1
  mulps xmm1, xmm0 ; // перемножить пакеты плавающих точек
  movups a, xmm1; // выгрузить результаты из xmm1 по адресам a
};
```

Процессы и потоки

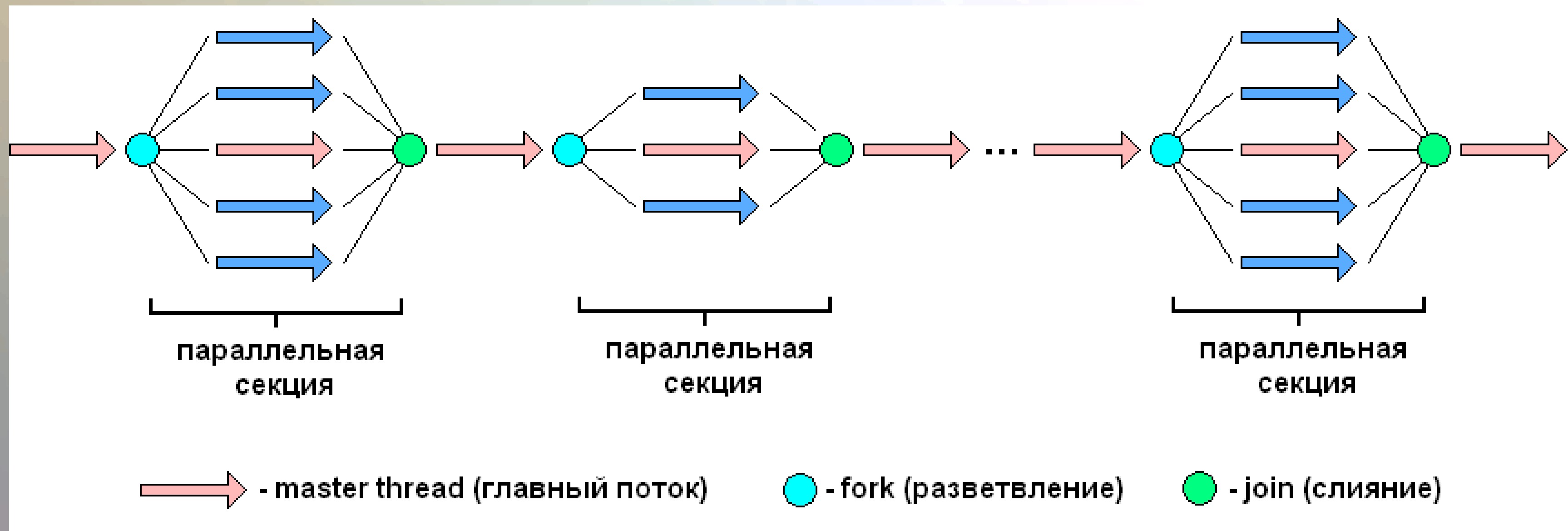
Процесс – исполнение последовательности действий

- В начале выполнения процесс представляет собой один поток
- Потоки могут создавать новые потоки в пределах одного процесса

- Все потоки данного процесса имеют общие сегмент кода и сегмент данных

- каждый поток имеет свой стэк выполнения

Вилочный параллелизм



Потоки ОС

Операция	*Posix Threads*
Создание	pthread_create()
Ожидание завершения	pthread_join()
Захват мьютекса	pthread_mutex_lock()
Освобождение мьютекса	pthread_mutex_unlock()

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine)(void*), void *arg);
```

Пример Posix Threads (1)

```
double integrate(double from, double to, double step) {
    double sum = 0;
    for (double x = from; x < to - 1E-13*to; x += step)
        sum += x * step;

    return sum;
}

struct IntegrateTask {
    double from, to, step, res;
};

void* integrateThread(void* data) {
    IntegrateTask* task = (IntegrateTask*)data;
    task->res = integrate(task->from, task->to, task->step);
    pthread_exit(NULL);
}

#define NUM_THREADS 10
```

Пример Posix Threads (2)

```
int main() {
    pthread_t threads[NUM_THREADS] ;
    IntegrateTask tasks[NUM_THREADS];

    IntegrateTask mainTask = {0, 10, 0.01};
    double distance = (mainTask.to - mainTask.from) / NUM_THREADS;

    for (int i=0; i<NUM_THREADS; ++i) {
        tasks[i].from = mainTask.from + i*distance;
        tasks[i].to = mainTask.from + (i + 1)*distance;
        tasks[i].step = mainTask.step;

        pthread_create(&threads[i], NULL, integrateThread, (void*)&tasks[i]);
    }

    double res = 0;

    for (int i=0; i<NUM_THREADS; ++i) {
        pthread_join(threads[i], NULL);
        res += tasks[i].res;
    }

    return 0;
}
```

Java-потоки

1. Наследование от Thread
2. Реализация Runnable
3. Остальное привычно: start(), join()

```
public class IntegrateRunnable implements Runnable {

        public IntegrateTask task;

        @Override
        public void run() {
                task.res = 0;
            for (double x = task.from; x < task.to - 1E-13*task.to; x += task.step) {
                task.res += task.f(x) * task.step;
            }
        }
}
```

Boost-потоки

Обёртка над потоками ОС (Posix или Win threads)

- Конструктор из указателя на функцию
- joinable()
- join()
- timed_join()
- detach()
- get_id()
- interrupt()
- sleep()

Пример boost::thread

```
boost::thread_group th_group;
for(...i)
{
  boost::thread* th = new
boost::thread(boost::bind(&Класс::<функция>, this, i));
  th_group.add_thread(th);
//th_group.create_new(boost::bind(&Класс::<функция>,this, i));
}
th_group.join_all();
```

OpenMP

1. Стандарт интерфейса для многопоточного программирования над общей памятью
2. Набор средств для языков C/C++ и Fortran:
   - Директивы компилятора *(#pragma omp ...)*
   - Библиотечные подпрограммы *(get_num_threads())*
   - Переменные окружения *(OMP_NUM_THREADS)*

Общий вид OpenMP

```
#include <omp.h>
int main()
{
  // последовательный код
  #pragma omp parallel
  {
    // параллельный код
  }
  // последовательный код

  return 0;
}
```

Пример OpenMP

```
#include <stdio.h>
#include <omp.h>
int main()
{ int i;
  #pragma omp parallel
  {
    #pragma omp for
    for (i=0;i<1000;i++)
      printf("%d ",i);
  }
  return 0;
}
```

Директивы OpenMP

1. Управление задачами:

    For

    Master

    Section...

2. Синхронизация:

    Private

    Shared

    Reduction...

Intel TBB (Threading Building Blocks)

Параллельные алгоритмы
parallel_for
parallel_reduce
parallel_scan
parallel_do
pipeline
parallel_sort

Планировщик задач
task
task_scheduler_init
task_scheduler_observer

Примитивы синхронизации
atomic, mutex, recursive_mutex
queuing_mutex, queuing_rw_mutex
spin_mutex, spin_rw_mutex

Контейнеры
concurrent_hash_map
concurrent_queue
concurrent_vector

Аллокаторы
tbb_allocator
cache_aligned_allocator
scalable_allocator

Работа с потоками
tbb_thread

Прочее
tick_count, task_group_context
blocked ranges, partitioners

Пример Intel TBB

```
void SortExample( ) {
   for( int i = 0; i < N; i++ ) {
      a[i] = sin((double)i);
      b[i] = cos((double)i);
   }
   parallel_sort(a, a + N);
   parallel_sort(b, b + N, std::greater<float>( ));
}
```

```
void ParallelApplyFoo(float a[], size_t n ) {
   parallel_for( blocked_range<size_t>( 0, n ),
    [&](const blocked_range<size_t>& range) {
      for(int i= range.begin(); i!=range.end(); i++)
            Foo(a[i]);
    },
    auto_partitioner() );
}
```

Intel TBB с функтором

Тело цикла – объект-функция

```
class ApplyFoo {
  float* const my_a;
public:
  ApplyFoo( float* a ) : my_a(a) {}
  void operator()(const blocked_range<size_t>& range) const {
      float* a = my_a;
      for( size_t i=range.begin(); i!=range.end(); i++ )
          Foo(a[i]);
  }
};

void ParallelApplyFoo( float a[], size_t n ) {
    parallel_for( blocked_range<size_t>( 0, n ),
              ApplyFoo(a), auto_partitioner());
}
```

Пространство итераций

Параллельный алгоритм

Разбиение области

Java.util.concurrent

1. Пулы потоков:
   - FixedThreadPool
   - CahcedThreadPool
   - SingleThreadPool...
2. Атомики (AtomicBoolean, AtomicLong...)
3. Потокобезопасные контейнеры (ConcurrentHashMap, ConcurrentLinkedQueue...)
4. Свои примитивы (ReentrantLock...)
5. Будущее (Future<>)

```java
private ExecutorService thread_pool;
private int[] array;

private class quick_sort_call implements Callable<Boolean>
{
...
  public Boolean call()
  {
    ...
    thread_pool.submit(...);
  }
}

public array_sorter() {
   thread_pool = Executors.newCachedThreadPool();
}

public void start_sorting()
{
   Future<Boolean> main_future=null;

   main_future = thread_pool.submit(new quick_sort_call(0, array.length-1, array));

   //waiting for the main future to be obtained...
   if(main_future!=null){
     try
     {
       if(main_future.get())System.out.println("Future obtained.");
     }catch(Exception ex)
     {
       System.out.println("Something wrong with multithread sorting! " + ex.getMessage());
     }
   }

   thread_pool.shutdown();
}
```

Сравнение средств

	Intel® TBB	OpenMP	Threads
Параллелизм на уровне задач	+	+	-
Поддержка декомпозиции данных	+	+	-
Сложные параллельные алгоритмы	+	-	-
Вложенный параллелизм	+	-	-
Динамическая балансировка загрузки	+	+	-
Поддержка назначения задач на потоки	-	+	+
Статическое планирование	-	+	-
Структуры данных для многопоточности	+	-	-
Масштабируемое выделение памяти	+	-	-
Задачи ввода/вывода	+	-	+
Примитивы синхронизации	+	+	-
Не требует поддержки компилятора	+	-	+
Кроссплаформенность	+	+	-