Лекция 8
Стандарт OpenMP (продолжение)

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Курс «Высокопроизводительные вычислительные системы»
Сибирский государственный университет телекоммуникаций и информатики (Новосибирск)
Осенний семестр, 2015

Параллельное рекурсивное суммирование (nested sections)

```
double sum_omp(double *v, int low, int high)
{
    if (low == high)
        return v[low];

    double sum_left, sum_right;
    int mid = (low + high) / 2;

    if (omp_get_active_level() >= omp_get_max_active_levels())
        return sum(v, low, mid);

    #pragma omp parallel num_threads(2)
    {
        #pragma omp sections
        {
            #pragma omp section
            sum_left = sum_omp(v, low, mid);

            #pragma omp section
            sum_right = sum_omp(v, mid + 1, high);
        }
    }
    return sum_left + sum_right;
}

omp_set_nested(1);
omp_set_max_active_levels(ilog2(nthreads));
```

Переключение на последовательную версию при достижении предельной глубины вложенных параллельных регионов

Параллельное рекурсивное суммирование (nested sections)

```
double sum_omp(double *v, int low, int high)
{
    if (low == high)
        return v[low];

    double sum_left, sum_right;
    int mid = (low + high) / 2;

    if (omp_get_active_level() >= omp_get_max_active_levels())
        return sum(v, low, mid);

    #pragma omp parallel num_threads(2)
    {
        #pragma omp sections
        {
            #pragma omp section
            sum_left = sum_omp(v, low, mid);

            #pragma omp section
            sum_right = sum_omp(v, mid + 1, high);
        }
    }
    return sum_left + sum_right;
}

omp_set_nested(1);
omp_set_max_active_levels(ilog2(nthreads));
```

Переключение на последовательную версию при достижении предельной глубины вложенных параллельных регионов

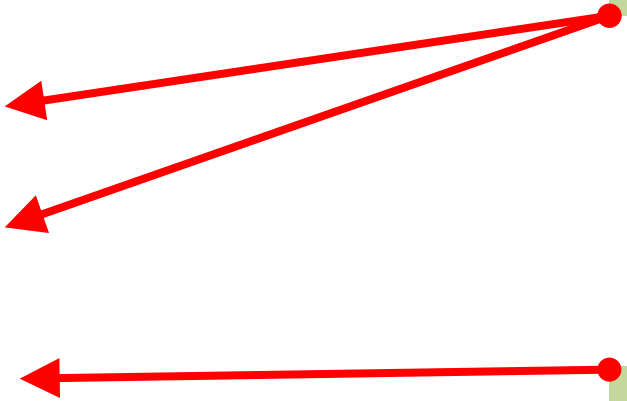
Параллелизм задач (task parallelism, >= OpenMP 3.0)

```
int fib(int n)
{
    if (n < 2)
        return n;
    return fib(n - 1) + fib(n - 2);
}
```

- Директива task создает задачу (легковесный поток)
- Задачи из пула динамически выполняются группой потоков
- Динамическое распределение задача по потокам осуществляется алгоритмами планирования типа work stealing
- Задач может быть намного больше количества потоков

Параллелизм задач (task parallelism, >= OpenMP 3.0)

```
int fib(int n)
{
    int x, y;

    if (n < 2)
        return n;
    #pragma omp task shared(x, n)
    x = fib(n - 1);
    #pragma omp task shared(y, n)
    y = fib(n - 2);
    #pragma omp taskwait
    return x + y;
}

#pragma omp parallel
#pragma omp single
    val = fib(n);
```

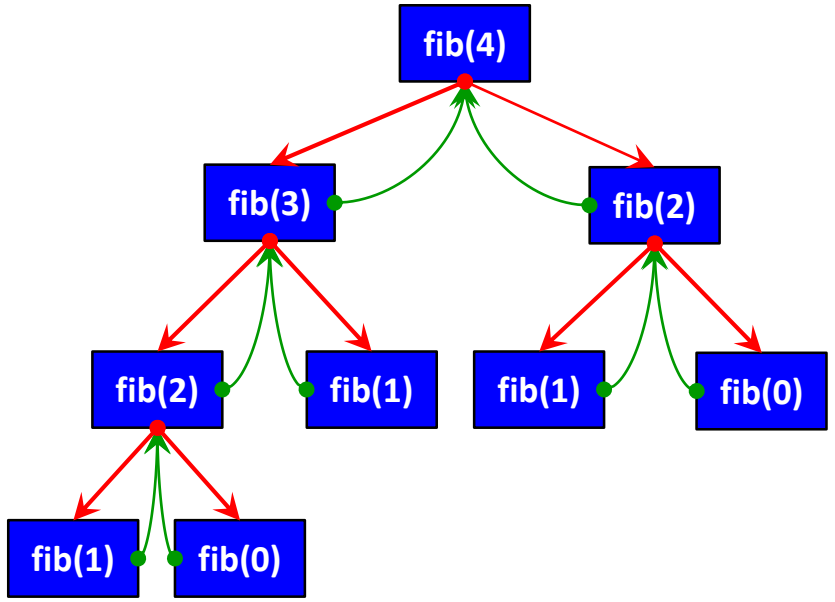


Каждый рекурсивный вызов – это задача

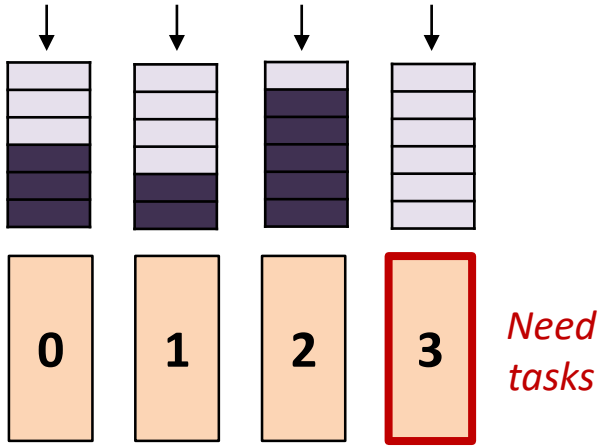
Ожидаем завершение дочерних задач

Параллелизм задач (task parallelism, >= OpenMP 3.0)

Легковесные задачи (tasks)

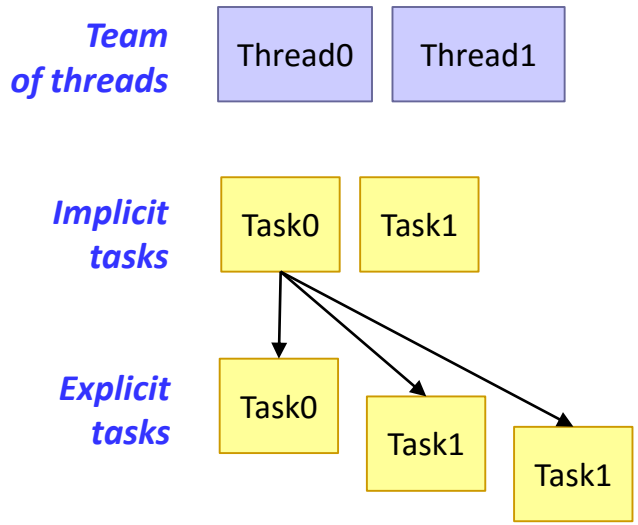


Очереди (деки) задач + пул потоков операционной системы



Параллелизм задач (task parallelism, >= OpenMP 3.0)

```
void fun()
{
    int a, b;
    #pragma omp parallel num_threads(2) shared(a) private(b)
    {
        #pragma omp single nowait
        {
            for (int i = 0; i < 3; i++) {
                #pragma omp task default(firstprivate)
                {
                    int c;
                    // A – shared, B – firstprivate, C - private
                }
            }
        }
    }
}

int main(int argc, char **argv)
{
    fun();
    return 0;
}
```



Параллелизм задач (task parallelism, >= OpenMP 3.0)

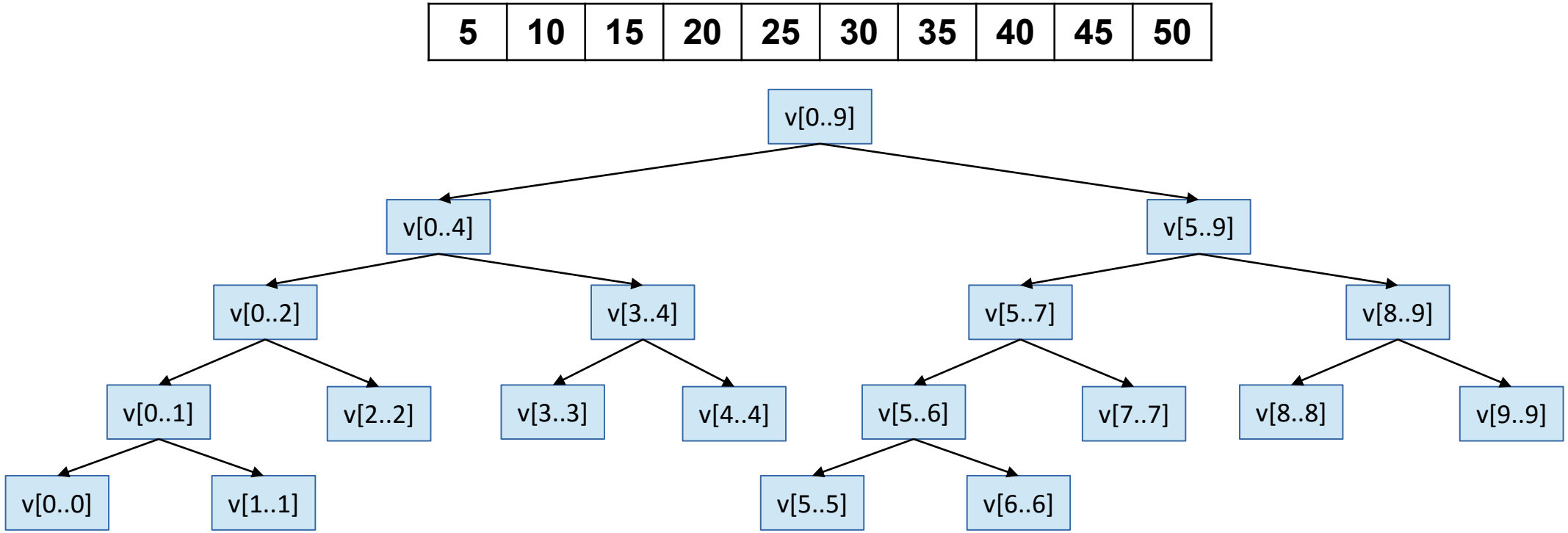
Параллельная обработка динамических структур данных
(связные списки, деревья, ...)

```
// Проход по связному списку
#pragma omp parallel
{

    #pragma omp single nowait
    {
        for (; list != NULL; list = list->next) {
            #pragma omp task firstprivate(list)
            {
                process_node(list);
            }
        }
    }
}
```

```
// Обход дерева
void postorder(node *p)
{
    if (p->left) {
        #pragma omp task
        postorder(p->left);
    }

    if (p->right) {
        #pragma omp task
        postorder(p->right);
    }

    #pragma omp taskwait
    process(p->data);
}
```

Рекурсивное суммирование

```
double sum(double *v, int low, int high)
{
    if (low == high)
        return v[low];

    int mid = (low + high) / 2;
    return sum(v, low, mid) + sum(v, mid + 1, high);
}
```

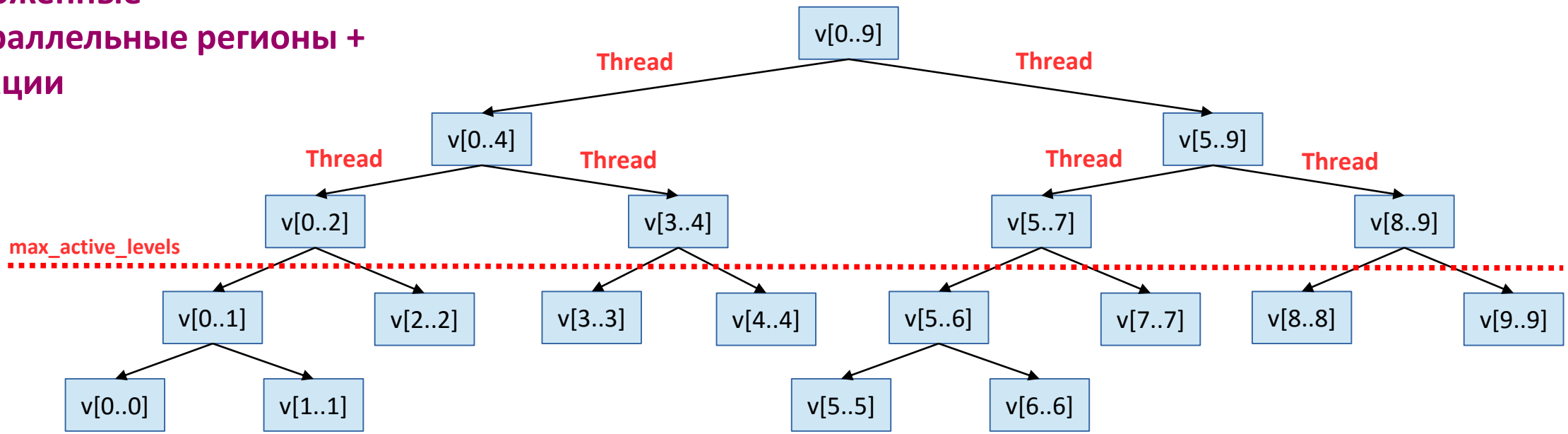


Рекурсивное суммирование

```
double sum(double *v, int low, int high)
{
    if (low == high)
        return v[low];

    int mid = (low + high) / 2;
    return sum(v, low, mid) + sum(v, mid + 1, high);
}
```

5	10	15	20	25	30	35	40	45	50

Вложенные параллельные регионы + секции



Параллельное рекурсивное суммирование (tasks v1)

```
double sum_omp_tasks(double *v, int low, int high)
{
    if (low == high) return v[low];

    double sum_left, sum_right;
    int mid = (low + high) / 2;

    #pragma omp task shared(sum_left)
    sum_left = sum_omp_tasks(v, low, mid);

    #pragma omp task shared(sum_right)
    sum_right = sum_omp_tasks(v, mid + 1, high);

    #pragma omp taskwait
    return sum_left + sum_right;
}

double sum_omp(double *v, int low, int high)
{
    double s = 0;
    #pragma omp parallel
    {
        #pragma omp single nowait
        s = sum_omp_tasks(v, low, high);
    }
    return s;
}
```

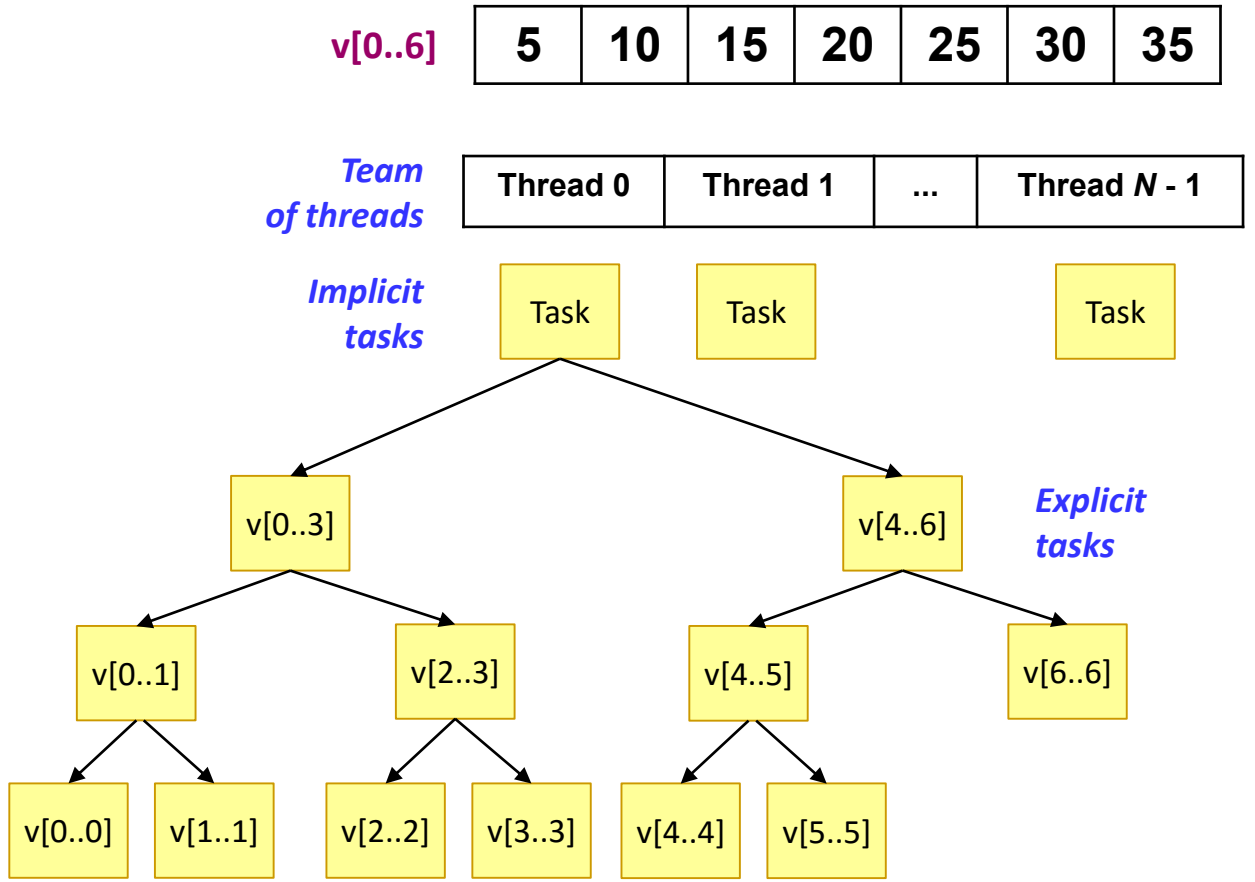
Отдельная задача для каждого рекурсивного вызова

Ожидание завершения дочерних задач

Пул из *N* потоков *+ N* задач (implicit tasks)

Параллельное рекурсивное суммирование (tasks v1)

```
double sum_omp_tasks(double *v, int low, int high)
{
    if (low == high) return v[low];

    double sum_left, sum_right;
    int mid = (low + high) / 2;

    #pragma omp task shared(sum_left)
    sum_left = sum_omp_tasks(v, low, mid);

    #pragma omp task shared(sum_right)
    sum_right = sum_omp_tasks(v, mid + 1, high);

    #pragma omp taskwait
    return sum_left + sum_right;
}

double sum_omp(double *v, int low, int high)
{
    double s = 0;
    #pragma omp parallel
    {
        #pragma omp single nowait
        s = sum_omp_tasks(v, low, high);
    }
    return s;
}
```

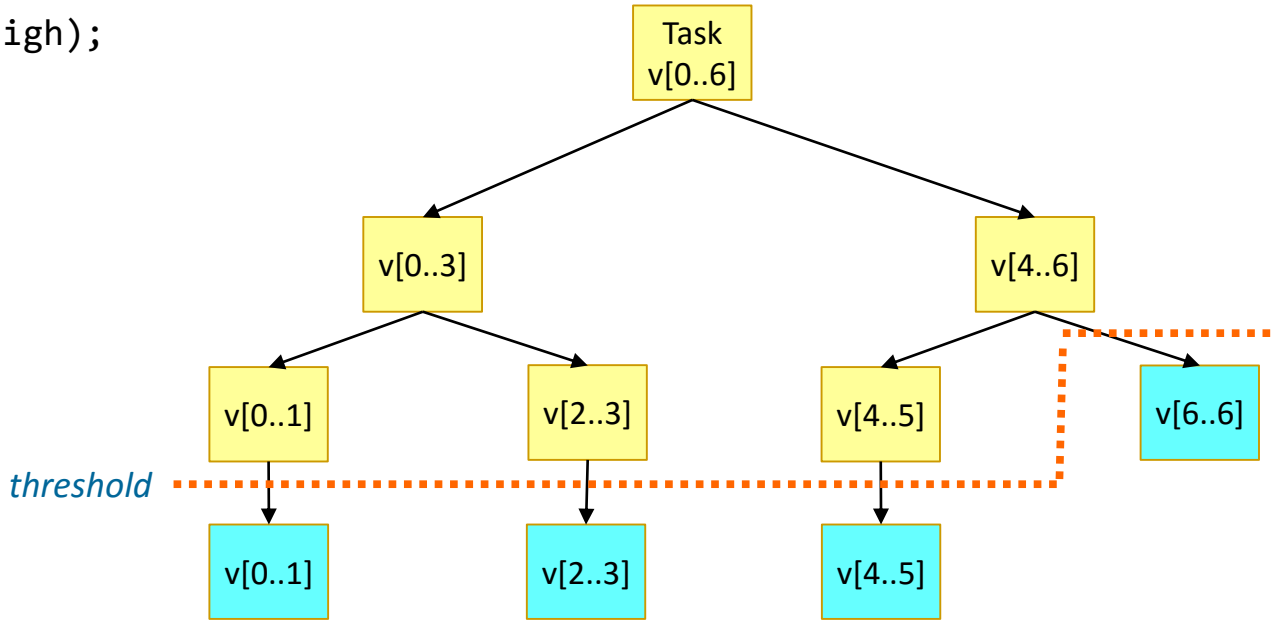


Создается большое количество задач
Значительные накладные расходы

Параллельное рекурсивное суммирование (tasks v2)

```
double sum_omp_tasks_threshold(double *v, int low, int high)
{
    if (low == high) return v[low];

    if (high - low < SUM_OMP_ARRAY_MIN_SIZE)
        return sum(v, low, high);

    double sum_left, sum_right;
    int mid = (low + high) / 2;

    #pragma omp task shared(sum_left)
    sum_left = sum_omp_tasks_threshold(v, low, mid);
    #pragma omp task shared(sum_right)
    sum_right = sum_omp_tasks_threshold(v, mid + 1, high);
    #pragma omp taskwait
    return sum_left + sum_right;
}

double sum_omp(double *v, int low, int high)
{
    double s = 0;
    #pragma omp parallel
    {
        #pragma omp single nowait
        s = sum_omp_tasks_threshold(v, low, high);
    }
    return s;
}
```

Переключение на последовательную версию при достижении предельного размера подмассива

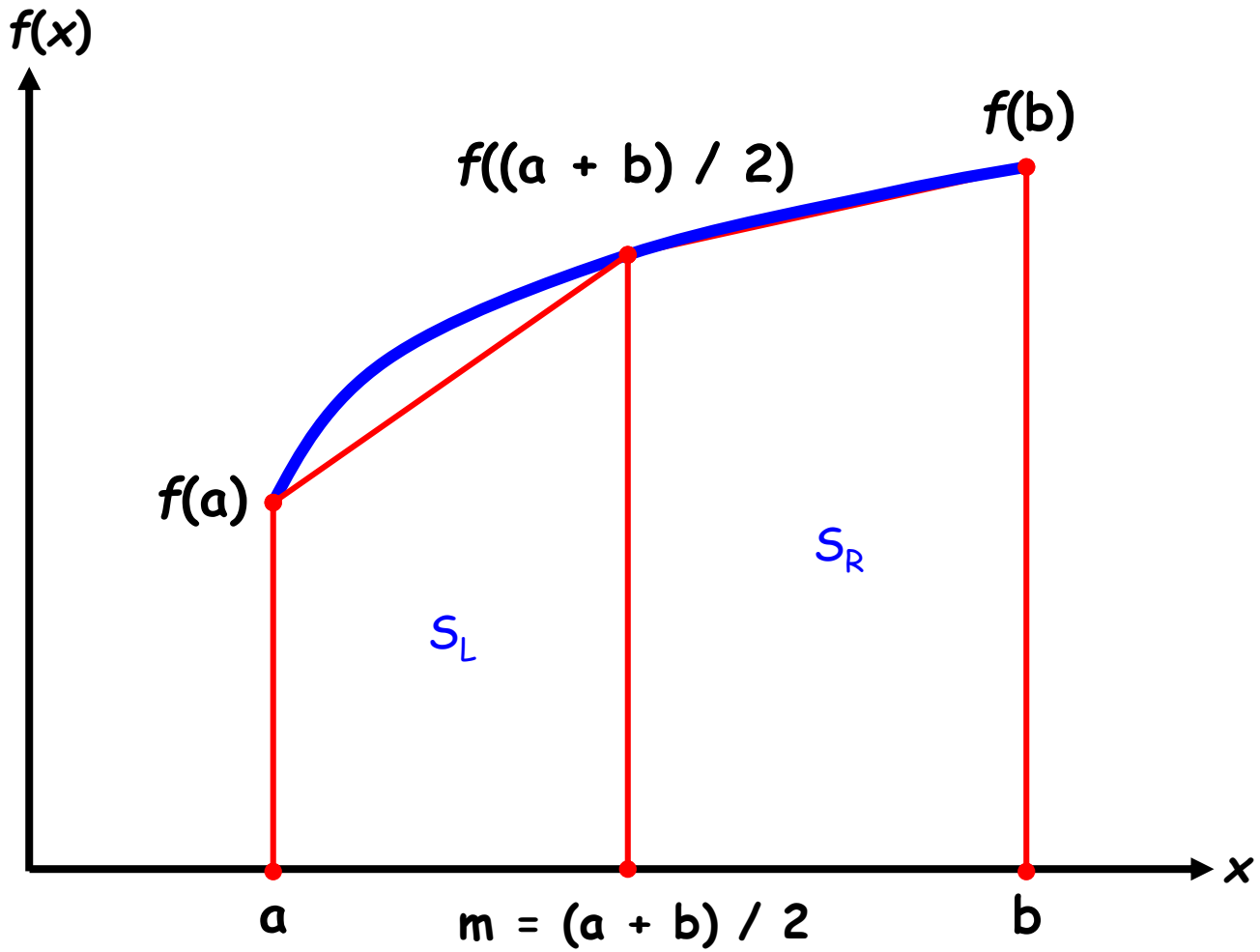


Параллельное рекурсивное суммирование (tasks v3)

```
double sum_omp_tasks_maxthreads(double *v, int low, int high, int nthreads)
{
    if (low == high) return v[low];

    if (nthreads <= 1) return sum(v, low, high);

    double sum_left, sum_right;
    int mid = (low + high) / 2;

    #pragma omp task shared(sum_left)
    sum_left = sum_omp_tasks_maxthreads(v, low, mid, nthreads / 2);
    #pragma omp task shared(sum_right)
    sum_right = sum_omp_tasks_maxthreads(v, mid + 1, high, nthreads - nthreads / 2);
    #pragma omp taskwait
    return sum_left + sum_right;
}

double sum_omp(double *v, int low, int high)
{
    double s = 0;
    #pragma omp parallel
    {
        #pragma omp single nowait
        s = sum_omp_tasks_maxthreads(v, low, high, omp_get_num_procs());
    }
    return s;
}
```

Переключение на последовательную версию при достижении предельного числа запущенных задач

Численное интегрирование (метод трапеций)

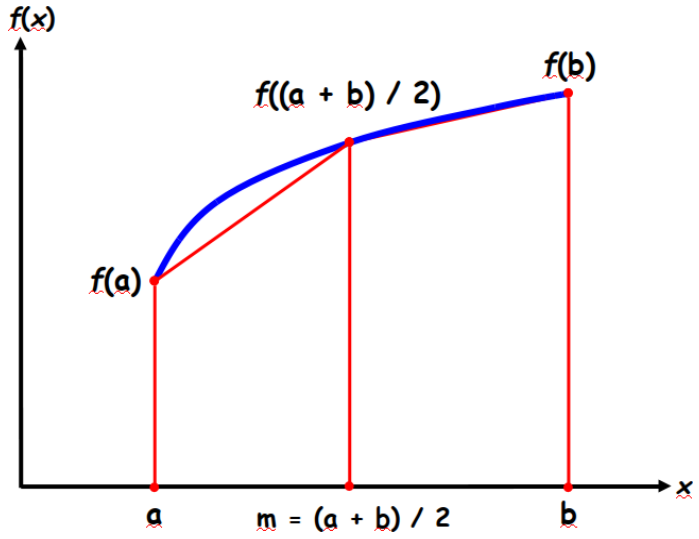


- Площадь левой трапеции:
  $S_L = (f(a) + f(m)) * (m - a) / 2$
- Площадь правой трапеции:
  $S_R = (f(m) + f(b)) * (b - m) / 2$

$$S = S_L + S_R$$

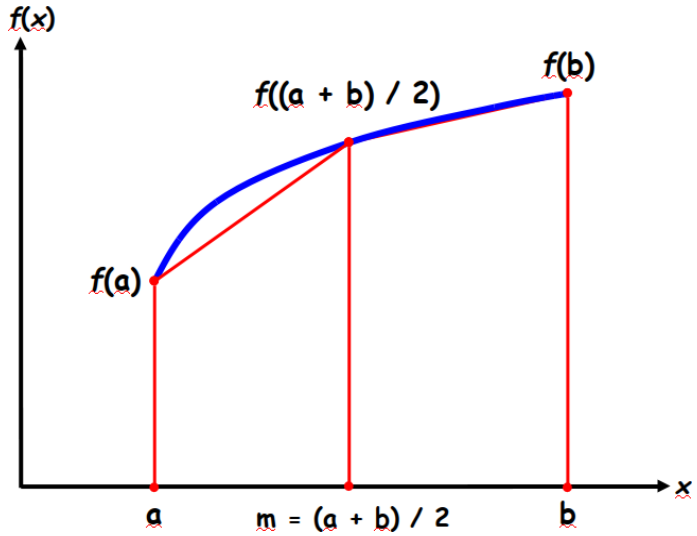
Численное интегрирование (метод трапеций)

```
const double eps = 1E-24;

double f(double x)
{
    return 4.0 / (1.0 + x * x);
}

double integrate(double left, double right, double f_left,
                 double f_right, double leftright_area)
{
    double mid = (left + right) / 2;
    double f_mid = f(mid);
    double left_area = (f_left + f_mid) * (mid - left) / 2;
    double right_area = (f_mid + f_right) * (right - mid) / 2;
    if (fabs((left_area + right_area) - leftright_area) > eps) {
        left_area = integrate(left, mid, f_left, f_mid, left_area);
        right_area = integrate(mid, right, f_mid, f_right, right_area);
    }
    return left_area + right_area;
}


double run_serial()
{
    double pi = integrate(0.0, 1.0, f(0), f(1), (f(0) + f(1)) / 2);
}
```



Численное интегрирование (метод трапеций)

```
const double threshold = 0.05;
double integrate_omp(double left, double right, double f_left,
                     double f_right, double leftright_area)
{
    double mid = (left + right) / 2;
    double f_mid = f(mid);
    double left_area = (f_left + f_mid) * (mid - left) / 2;
    double right_area = (f_mid + f_right) * (right - mid) / 2;
    if (fabs((left_area + right_area) - leftright_area) > eps) {
        if (right - left < threshold)
            return integrate(left, right, f_left, f_right, leftright_area);
        #pragma omp task shared(left_area)
        left_area = integrate_omp(left, mid, f_left, f_mid, left_area);

        right_area = integrate_omp(mid, right, f_mid, f_right, right_area);
        #pragma omp taskwait
    }
    return left_area + right_area;
}
double run_parallel()
{
    double pi;
    #pragma omp parallel
    {
        #pragma omp single nowait
        pi = integrate_omp(0.0, 1.0, f(0), f(1), (f(0) + f(1)) / 2);
    }
}
```



Быстрая сортировка (QuickSort)

```
int partition(double *v, int low, int high)
{
    double pivot = v[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (v[j] <= pivot) {
            i++;
            swap(v, i, j);
        }
    }
    swap(v, i + 1, high);
    return i + 1;
}

void quicksort(double *v, int low, int high)
{
    if (low < high) {
        int k = partition(v, low, high);
        quicksort(v, low, k - 1);
        quicksort(v, k + 1, high);
    }
}

double run_serial()
{
    quicksort(v, 0, N - 1);
}
```

Многопоточная быстрая сортировка (QuickSort)

```
const int threshold = 1000;

void quicksort_omp_tasks(double *v, int low, int high)
{
    if (low < high) {
        if (high - low < threshold) {
            quicksort(v, low, high);
        } else {
            int k = partition(v, low, high);

            #pragma omp task
            quicksort_omp_tasks(v, low, k - 1);

            quicksort_omp_tasks(v, k + 1, high);
        }
    }
}

double run_parallel()
{
    #pragma omp parallel
    {
        #pragma omp single nowait
        quicksort_omp_tasks(v, 0, N - 1);
    }
}
```

Блокировки (locks)

```
#include <omp.h>

int main()
{
    std::vector<int> vec(1000);

    std::fill(vec.begin(), vec.end(), 1);
    int counter = 0;
    omp_lock_t lock;
    omp_init_lock(&lock);

    #pragma omp parallel for
    for (std::vector<int>::size_type i = 0; i < vec.size(); i++) {
        if (vec[i] > 0) {
            omp_set_lock(&lock);
            counter++;
            omp_unset_lock(&lock);
        }
    }
    omp_destroy_lock(&lock);
    std::cout << "Counter = " << counter << std::endl;
    return 0;
}
```

OpenMP 4.0: Поддержка ускорителей (GPU)

```
sum = 0;
#pragma omp target device(acc0) in(B,C)
#pragma omp parallel for reduction(+:sum)
for (i = 0; i < N; i++)
    sum += B[i] * C[i]
```

- **omp_set_default_device()**
- **omp_get_default_device()**
- **omp_get_num_devices()**

OpenMP 4.0: SIMD-конструкции

- SIMD-конструкции для векторизации циклов (SSE, AVX2, AVX-512, AltiVec, …)

```
void minex(float *a, float *b, float *c, float *d)
{
    #pragma omp parallel for simd
    for (i = 0; i < N; i++)
        d[i] = min(distsq(a[i], b[i]), c[i]);
}
```

OpenMP 4.0: Thread Affinity

- **Thread affinity – привязка потоков к процессорным ядрам**

```
#pragma omp parallel proc_bind(master | close | spread)
omp_proc_bind_t omp_get_proc_bind(void)
Env. variable OMP_PLACES
```

```
export OMP_NUM_THREADS=16
export OMP_PLACES=0,8,1,9,2,10,3,11,4,12,5,13,6,14,7,15
export OMP_PROC_BIND=spread,close
```

OpenMP 4.0: user defined reductions

```
#pragma omp declare reduction (reduction-identifier :
            typename-list : combiner) [identity(identity-expr)]
```

```
#pragma omp declare reduction (merge : std::vector<int> :
    omp_out.insert(omp_out.end(),
                   omp_in.begin(), omp_in.end()
                   ))

void schedule(std::vector<int> &v, std::vector<int> &filtered) {
    #pragma omp parallel for reduction (merge : filtered)
    for (std:vector<int>::iterator it = v.begin();
         it < v.end(); it++)
    {
        if (filter(*it))
            filtered.push_back(*it);
    }
}
```

Литература

- Эхтер Ш., Робертс Дж. **Многоядерное программирование**. – СПб.: Питер, 2010. – 316 с.
- Эндрюс Г.Р. **Основы многопоточного, параллельного и распределенного программирования**. – М.: Вильямс, 2003. – 512 с.
- Darryl Gove. **Multicore Application Programming: for Windows, Linux, and Oracle Solaris**. – Addison-Wesley, 2010. – 480 p.
- Maurice Herlihy, Nir Shavit. **The Art of Multiprocessor Programming**. – Morgan Kaufmann, 2008. – 528 p.
- Richard H. Carver, Kuo-Chung Tai. **Modern Multithreading : Implementing, Testing, and Debugging Multithreaded Java and C++/Pthreads/Win32 Programs**. – Wiley-Interscience, 2005. – 480 p.
- Anthony Williams. **C++ Concurrency in Action: Practical Multithreading**. – Manning Publications, 2012. – 528 p.
- Träff J.L. **Introduction to Parallel Computing** // http://www.par.tuwien.ac.at/teach/WS12/ParComp.html