

Семинар 7

Стандарт OpenMP (часть 7)

Михаил Курносов

E-mail: mkurnosov@gmail.com

WWW: www.mkurnosov.net

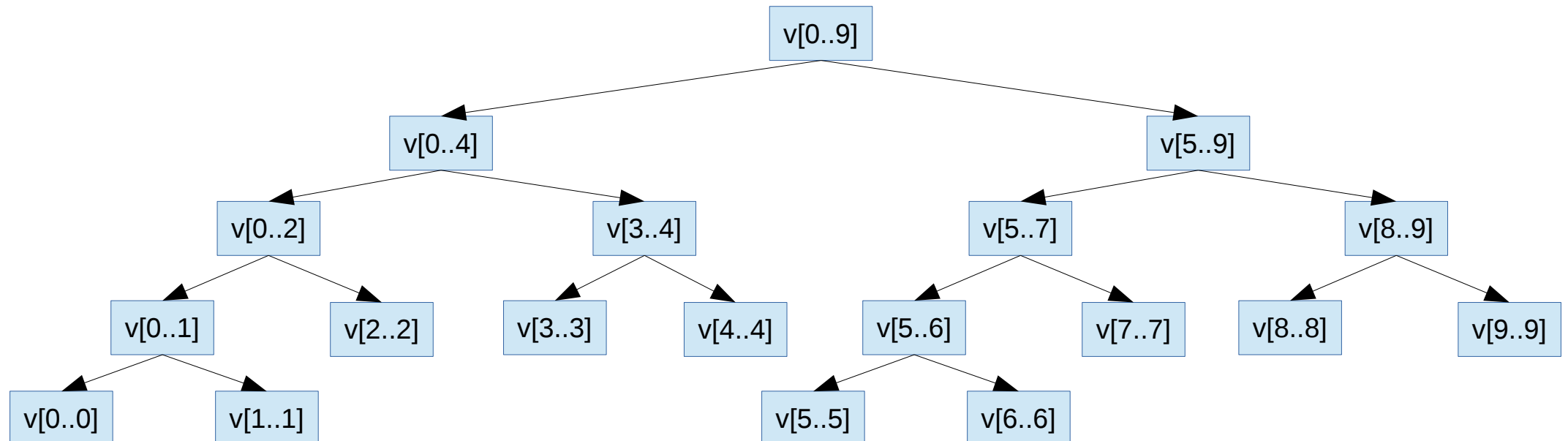
Цикл семинаров «Основы параллельного программирования»
Институт физики полупроводников им. А. В. Ржанова СО РАН
Новосибирск, 2015

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
---	----	----	----	----	----	----	----	----	----



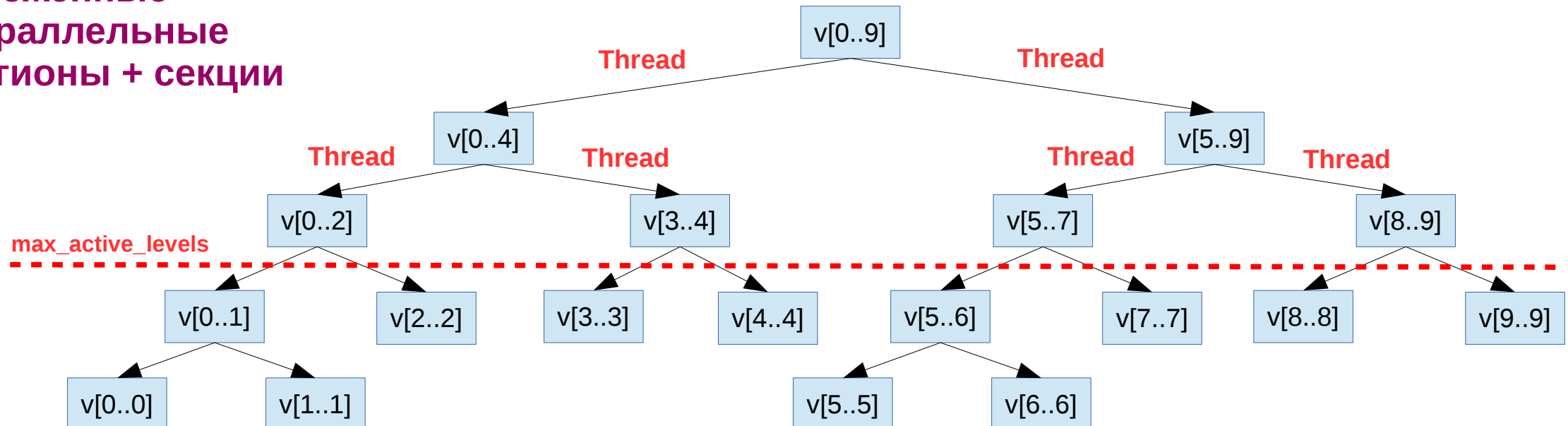
Параллельное рекурсивное суммирование

```
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 (nested sections)

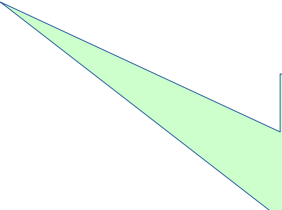
```
double sum_omp(double *v, int low, int high)
{
    if (low == high)
        return v[low];

    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;
}
```



**Переключение на
последовательную версию
при достижении предельной
глубины вложенных
параллельных регионов**

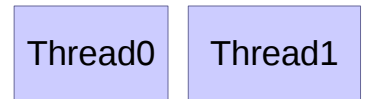
```
omp_set_nested(1);
omp_set_max_active_levels(ilog2(nthreads));
```

Параллелизм задач (task parallelism, \geq 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;
}
```

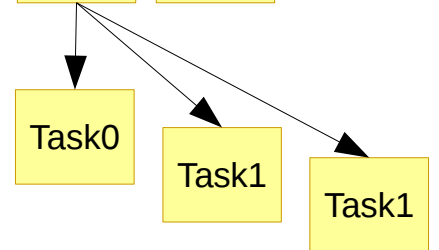
*Team
of threads*



*Implicit
tasks*



*Explicit
tasks*



Параллелизм задач (task parallelism, \geq OpenMP 3.0)

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

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

```
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);
}
```

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;
}
```

Отдельная задача для каждого
рекурсивного вызова

Ожидание завершения
дочерних задач

```
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;
}
```

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;
}
```

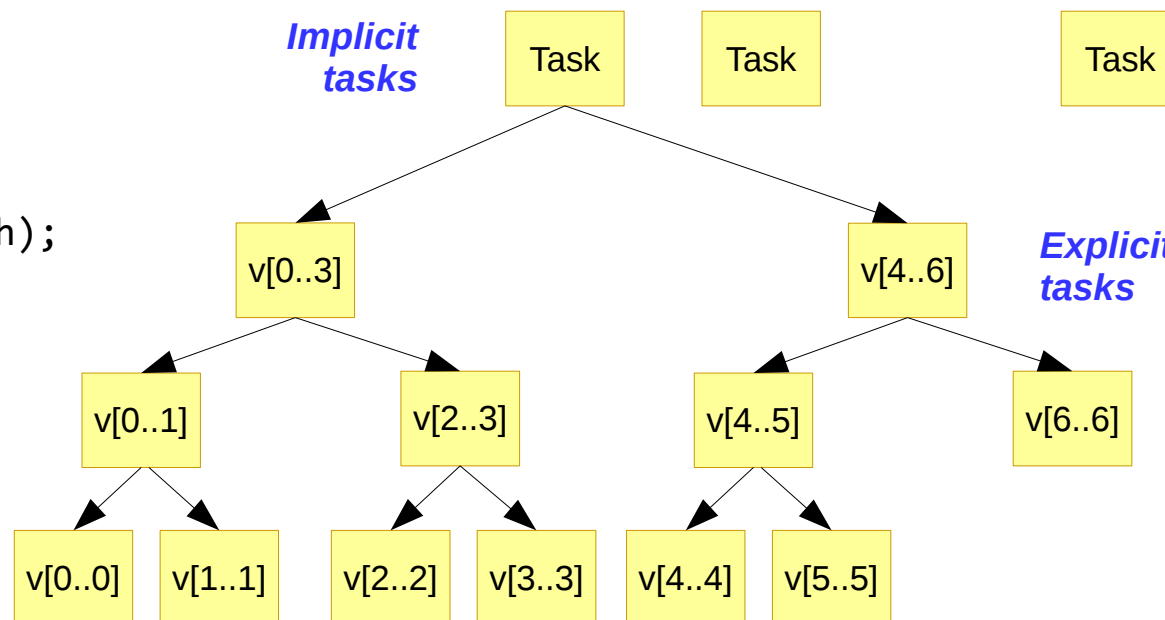
v[0..6]

5	10	15	20	25	30	35
---	----	----	----	----	----	----

**Team
of threads**

Thread 0	Thread 1	...	Thread N - 1
----------	----------	-----	--------------

**Implicit
tasks**



**Explicit
tasks**

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;

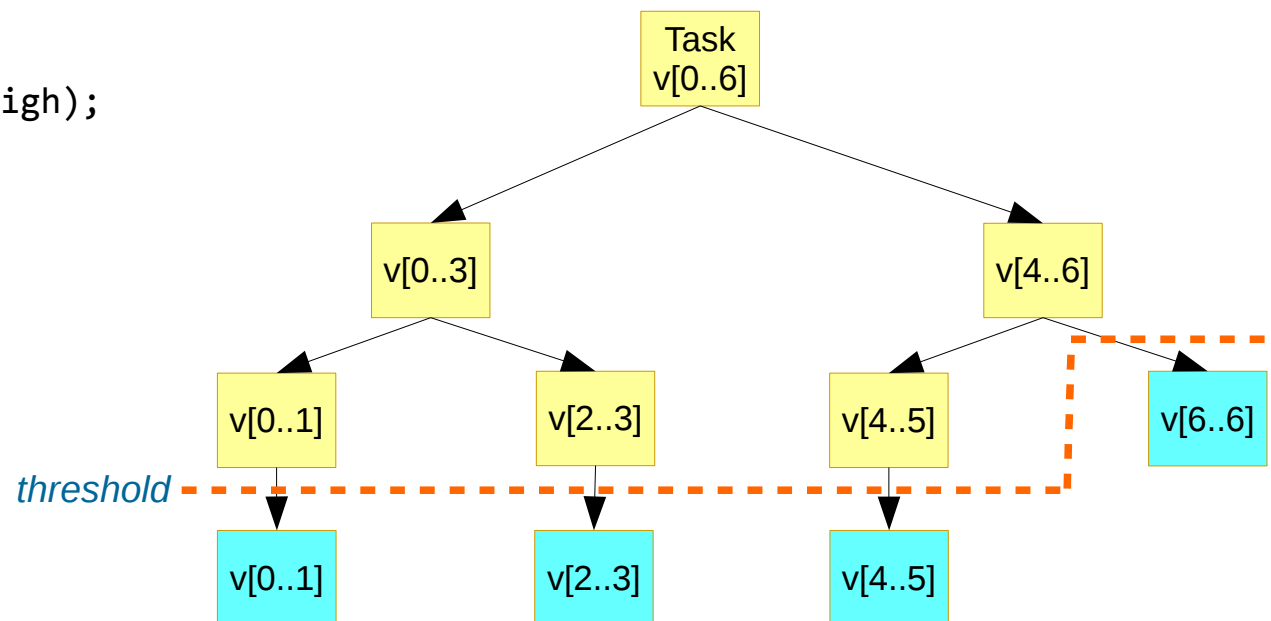
    #pragma omp task shared(sum_left)
    sum_left = sum_omp_tasks_threshold(v, low, mid);

    #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);

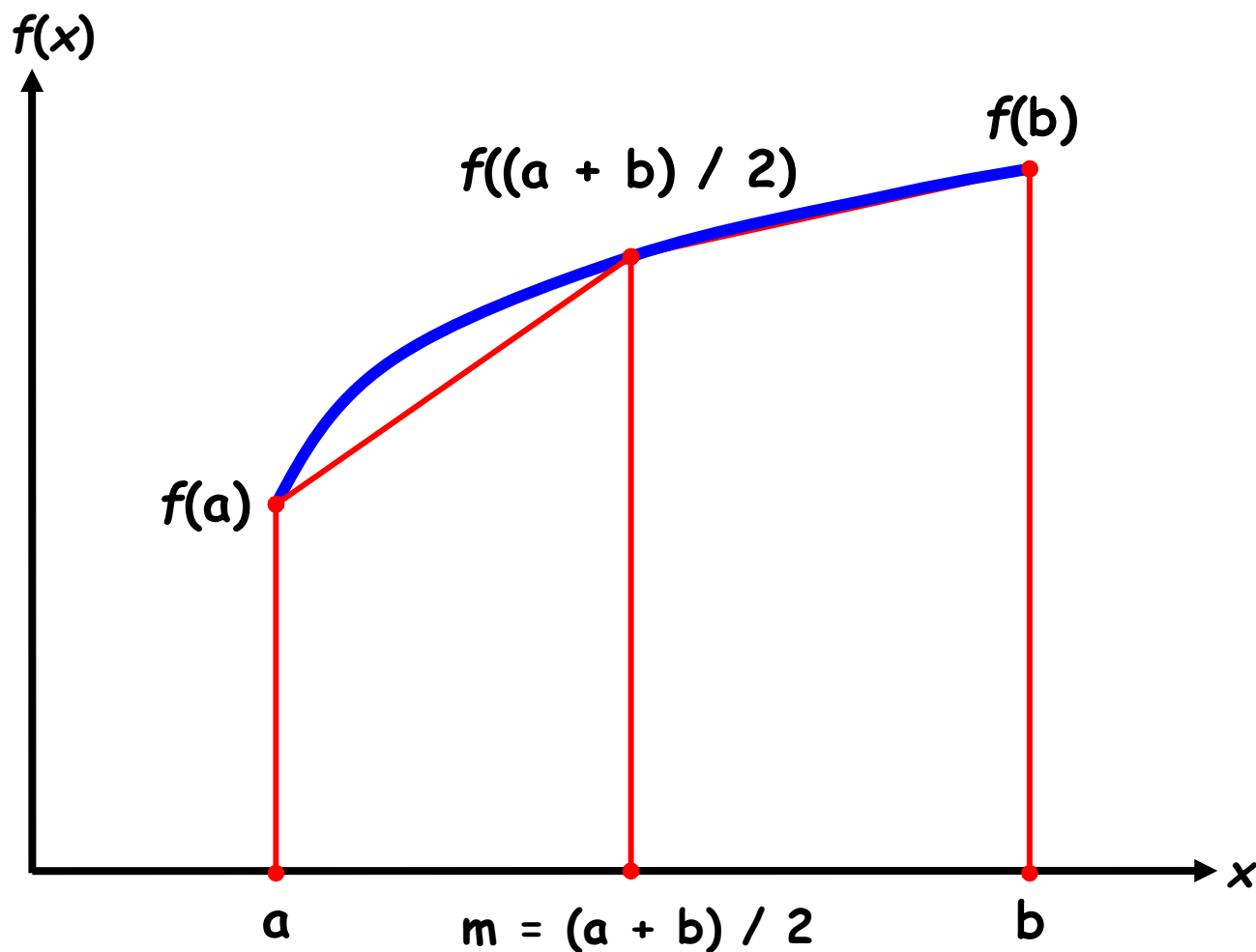
    #pragma omp task shared(sum_right)
    sum_right = sum_omp_tasks_maxthreads(v, mid + 1, high, nthreads - nthreads / 2);

    #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:

$$S_R = (f(m) + f(b)) * (b - m) / 2$$

$$S = S_L + S_R$$

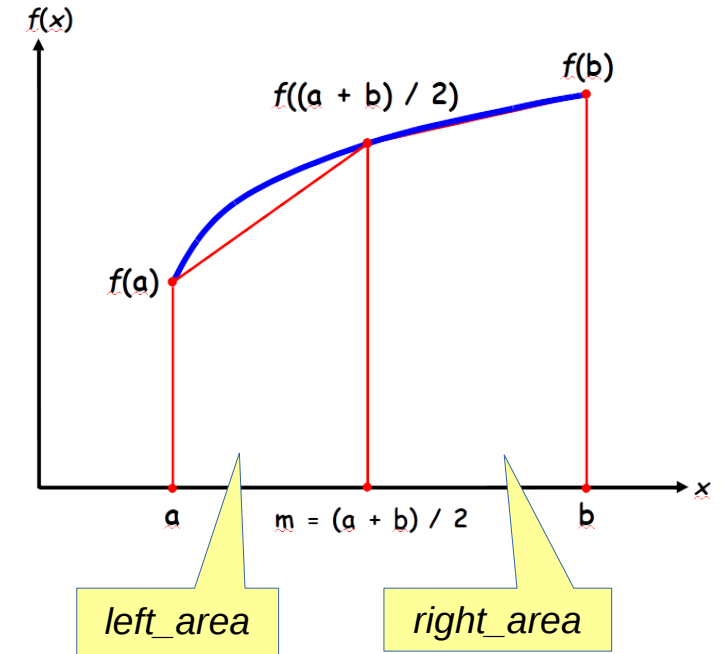
Численное интегрирование (метод трапеций)

```
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);
}
```



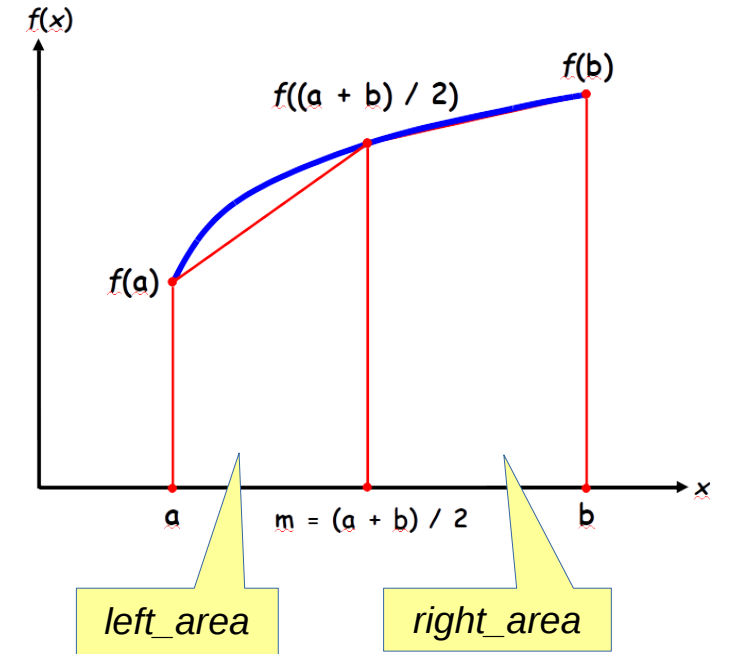
Численное интегрирование (метод трапеций)

```
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);
    }
}
```

Задание

- Разработать на OpenMP многопоточную версию функции быстрой сортировки — написать код функции `quicksort_omp_tasks`
- Провести анализ масштабируемости параллельной программы
- Шаблон программы находится в каталоге `_task`