Семинар 4
Стандарт OpenMP (часть 4)

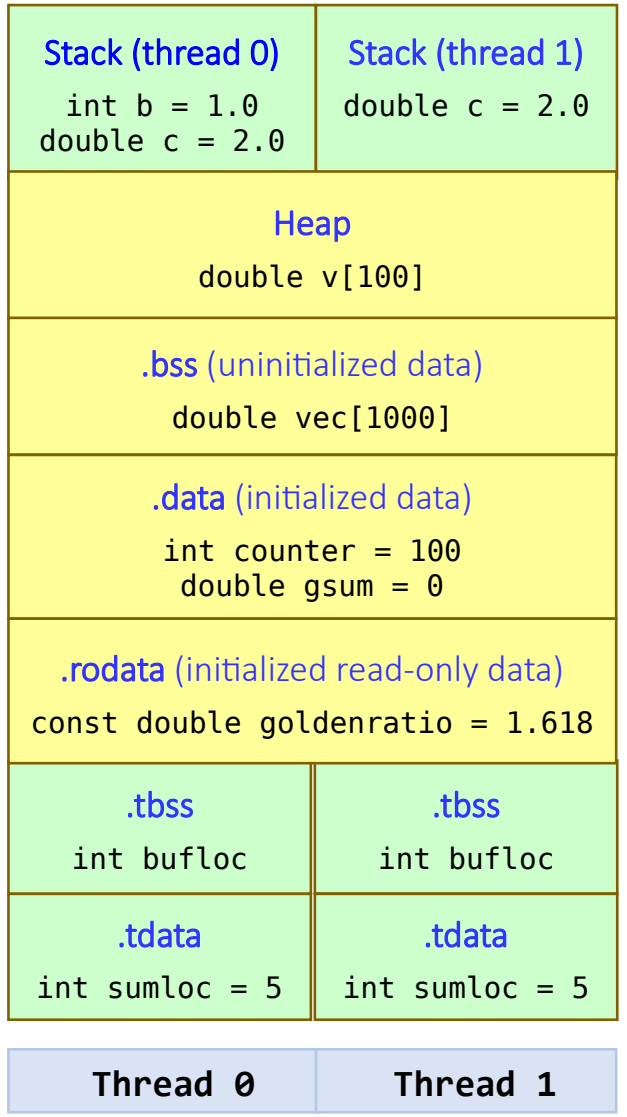
Михаил Курносов

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

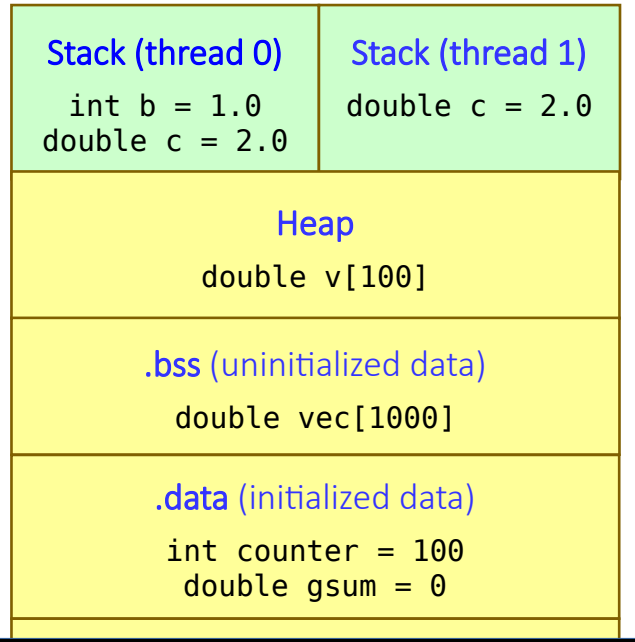
Видимость данных (C11 storage duration)

```c
const double goldenratio = 1.618;           /* Static (.rodata) */
double vec[1000];                           /* Static (.bss) */
int counter = 100;                          /* Static (.data) */

double fun(int a)
{
    double b = 1.0;                         /* Automatic (stack, register) */

    static double gsum = 0;                 /* Static (.data) */

    _Thread_local static double sumloc = 5; /* Thread (.tdata) */
    _Thread_local static double bufloc;     /* Thread (.tbbs) */

    double *v = malloc(sizeof(*v) * 100);   /* Allocated (Heap) */

    #pragma omp parallel num_threads(2)
    {
        double c = 2.0;                     /* Automatic (stack, register) */

        /* Shared: goldenratio, vec[], counter, b, gsum, v[] */
        /* Private: sumloc, bufloc, c */
    }

    free(v);
}
```



Видимость данных (C11 storage duration)

```c
const double goldenratio = 1.618;           /* Static (.rodata) */
double vec[1000];                           /* Static (.bss) */
int counter = 100;                          /* Static (.data) */

double fun(int a)
{
    double b = 1.0;                         /* Automatic (stack, register) */

    static double gsum = 0;                 /* Static (.data) */

    _Thread_local static double sumloc = 5; /* Thread (.tdata) */
    _Thread_local static double bufloc;     /* Thread (.tbbs) */

    double *v = malloc(sizeof(*v) * 100);   /* Allocated (Heap) */

    #pragma omp parallel num_threads(2)
    {
        double c = 2.0;

        /* Shared: goldenratio, vec[], c
        /* Private: sumloc, bufloc, c */
    }

    free(v);
}
```




```
$ objdump --syms ./datasharing

./datasharing:     file format elf64-x86-64
SYMBOL TABLE:

0000000000601088 l     O .bss     0000000000000008    gsum.2231
0000000000000000 l       .tdata   0000000000000008    sumloc.2232
0000000000000008 l       .tbss    0000000000000008    bufloc.2233
00000000006010c0 g     O .bss     0000000000001f40    vec
000000000060104c g     O .data    0000000000000004    counter
00000000004008e0 g     O .rodata  0000000000000008    goldenratio
```

Атрибуты видимости данных

```
#pragma omp parallel shared(a, b, c) private(x, y, z) firsprivate(i, j, k)
{
    #pragma omp for lastprivate(v)
    for (int i = 0; i < 100; i++)
}
```

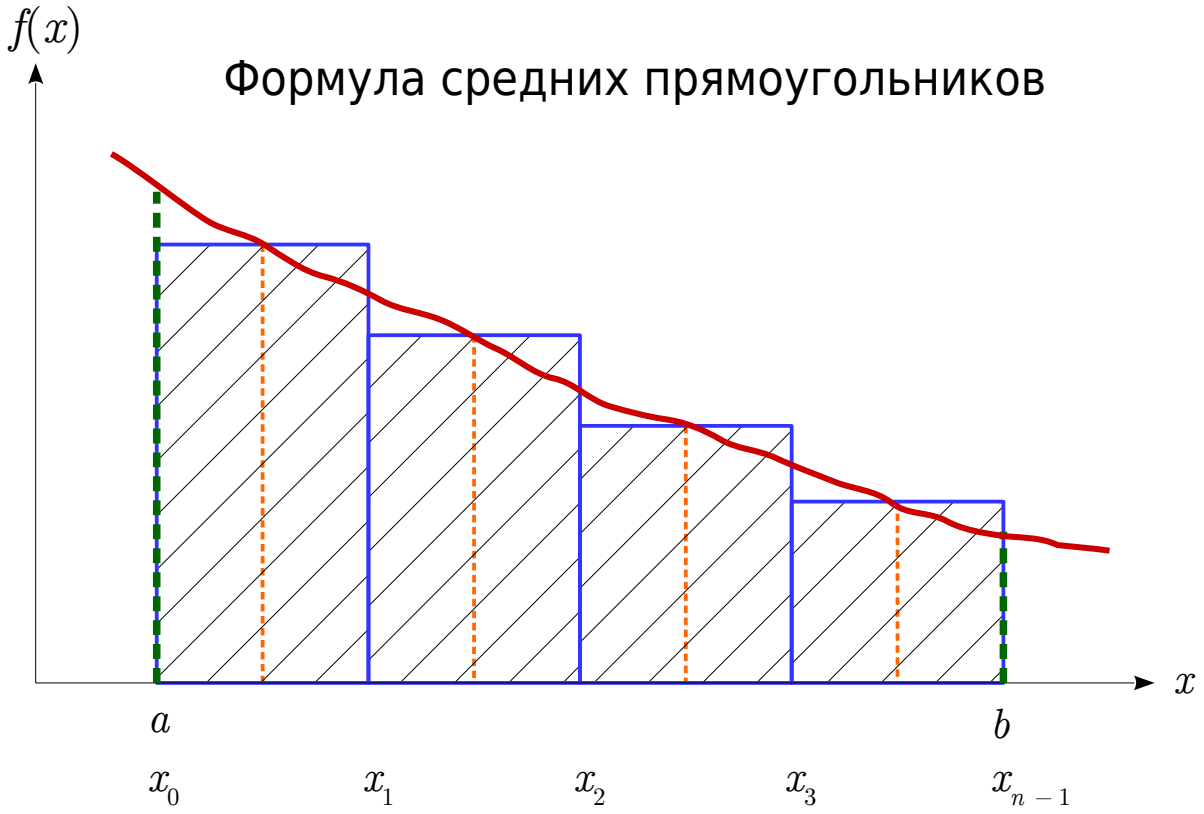
- **shared** (list) – указанные переменные сохраняют исходный класс памяти (auto, static, thread_local), все переменные кроме thread_local будут разделяемыми
- **private** (list) – для каждого потока создаются локальные копии указанных переменных (automatic storage duration)
- **firstprivate** (list) – для каждого потока создаются локальные копии переменных (automatic storage duration), они инициализируются значениями, которые имели соответствующие переменные до входа в параллельный регион
- **lastprivate** (list) – для каждого потока создаются локальные копии переменных (automatic storage duration), в переменные копируются значения последней итерации цикла, либо значения последней параллельной секции в коде (#pragma omp section)
- **#pragma omp threadprivate(list)** — делает указанные статические переменные локальными (TLS)

Атрибуты видимости данных

```
void fun()
{
    int a = 100;
    int b = 200;
    int c = 300;
    int d = 400;
    static int sum = 0;
    printf("Before parallel: a = %d, b = %d, c = %d, d = %d\n", a, b, c, d);

    #pragma omp parallel private(a) firstprivate(b) num_threads(2)
    {
        int tid = omp_get_thread_num();
        printf("Thread %d: a = %d, b = %d, c = %d, d = %d\n", tid, a, b, c, d);
        a = 1;
        b = 2;

        #pragma omp threadprivate(sum)
        sum++;

        #pragma omp for lastprivate(c)
        for (int i = 0; i < 100; i++)
            c = i;
        /* c=99 - has the value from last iteration */

    }

    // a = 100, b = 200, c = 99, d = 400, sum = 1
    printf("After parallel: a = %d, b = %d, c = %d, d = %d\n", a, b, c, d);
}
```

```
Before parallel: a = 100, b = 200, c = 300, d = 400
Thread 0: a = 0, b = 200, c = 300, d = 400
Thread 1: a = 0, b = 200, c = 300, d = 400
After parallel: a = 100, b = 200, c = 99, d = 400
```

Численное интегрирование (метод прямоугольников)




```
const double a = -4.0;          /* [a, b]  */
const double b = 4.0;
const int nsteps = 40000000;    /* n */

double func(double x)
{
    return exp(-x * x);
}

double integrate(double a, double b, int n)
{
    double h = (b - a) / n;
    double sum = 0.0;

    for (int i = 0; i < n; i++)
        sum += func(a + h * (i + 0.5));

    sum *= h;
    return sum;
}
```

$$\int_a^b f(x)\,dx \approx h\sum_{i=1}^{n} f\left(x_{i-1}+\frac{h}{2}\right) = h\sum_{i=1}^{n} f\left(x_i-\frac{h}{2}\right), \qquad h=\frac{b-a}{n}$$

Параллельная версия

```
int count_prime_numbers_omp(int a, int b)
{
    int nprimes = 0;

    if (a <= 2) {
        nprimes = 1;     /* Count '2' as a prime number */
        a = 2;
    }
    if (a % 2 == 0)      /* Shift 'a' to odd number */
        a++;

    #pragma omp parallel
    {
        int nloc = 0;

        /* Loop over odd numbers: a, a + 2, a + 4, ... , b */
        #pragma omp for schedule(dynamic,100) nowait
        for (int i = a; i <= b; i += 2) {
            if (is_prime_number(i))
                nloc++;
        } /* 'nowait' disables barrier after for */

        #pragma omp atomic
        nprimes += nloc;
    }
    return nprimes;
}
```

Суммируем результаты всех потоков

Редукция (reduction, reduce)

```
int count_prime_numbers_omp(int a, int b)
{
    int nprimes = 0;

    /* Count '2' as a prime number */
    if (a <= 2) {
        nprimes = 1;
        a = 2;
    }

    /* Shift 'a' to odd number */
    if (a % 2 == 0)
        a++;

    #pragma omp parallel
    {
        #pragma omp for schedule(dynamic,100) reduction(+:nprimes)
        for (int i = a; i <= b; i += 2) {
            if (is_prime_number(i))
                nprimes++;
        }
    }
    return nprimes;
}
```

- В каждом потоке создает private-переменная nprimes
- После завершения параллельного региона к локальным копиям применяется операция «+»
- Результат редукции записывается в переменную nprimes
- Допустимые операции: +, -, *, &, |, ^, &&, ||

Начальные значения переменных редукции

Identifier	Initializer	Combiner
+	omp_priv = 0	omp_out += omp_in
*	omp_priv = 1	omp_out *= omp_in
-	omp_priv = 0	omp_out += omp_in
&	omp_priv = ~0	omp_out &= omp_in
|	omp_priv = 0	omp_out |= omp_in
^	omp_priv = 0	omp_out ^= omp_in
&&	omp_priv = 1	omp_out = omp_in && omp_out
||	omp_priv = 0	omp_out = omp_in || omp_out
max	omp_priv = *Least representable number in the reduction list item type*	omp_out = omp_in > omp_out ? omp_in : omp_out
min	omp_priv = *Largest representable number in the reduction list item type*	omp_out = omp_in < omp_out ? omp_in : omp_out

Объединение пространств итераций циклов

```
enum {
    M = 4,
    N = 1000000
};

float m[M * N];

void fun()
{

    #pragma omp parallel
    {
        #pragma omp for
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                m[i * N + j] = i * j;
            }
        }
    }
}
```

- Число потоков может быть больше числа итераций цикла
- Часть потоков будет простаивать (не хватит итераций)

Ручное объединение пространств итераций циклов

```
void fun()
{
    #pragma omp parallel
    {

        #pragma omp for
        for (int ij = 0; ij < M * N; ij++) {
            int i = ij / N;
            int j = ij % N;

            m[i * N + j] = i * j;
        }
    }
}
```

Пример

Строк M = 2
Столбцов N = 5
Потоков p = 4

```
for ij = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Стандартное распределение итераций:
n = p * q – r,  q = ceil(n / p) = 3

0: 0, 1, 2     -- (0, 0), (0, 1), (0, 2)
1: 3, 4, 5     -- (0, 3), (0, 4), (1, 0)
2: 6, 7        -- (1, 1), (1, 2)
3: 8, 9        -- (1, 3), (1, 4)
```

Объединение пространств итераций циклов

```
void fun()
{
    #pragma omp parallel
    {

        #pragma omp for collapse(2)
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                m[i * N + j] = i * j;
            }
        }
    }
}
```

Пример

Строк M = 2
Столбцов N = 5
Потоков p = 4

```
for ij = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Стандартное распределение итераций:
n = p * q – r,  q = ceil(n / p) = 3

0: 0, 1, 2     -- (0, 0), (0, 1), (0, 2)
1: 3, 4, 5     -- (0, 3), (0, 4), (1, 0)
2: 6, 7        -- (1, 1), (1, 2)
3: 8, 9        -- (1, 3), (1, 4)
```

Директивы master и single

```
void fun()
{
    #pragma omp parallel
    {
        #pragma omp master
        {
            printf("Thread in master %d\n", omp_get_thread_num());
        }


        #pragma omp single
        {
            printf("Thread in single %d\n", omp_get_thread_num());
        }
    }
}
```

Выполняется потоком с номером 0

Выполняется один раз, любым потоком

Барьерная синхронизация

```
void fun()
{
    #pragma omp parallel
    {
        // Parallel code
        #pragma omp for nowait
        for (int i = 0; i < n; i++)
            x[i] = f(i);

        // Serial code
        #pragma omp single
        do_stuff();

        #pragma omp barrier
        // Ждем готовности x[0:n-1]

        // Parallel code
        #pragma omp for nowait
        for (int i = 0; i < n; i++)
            y[i] = x[i] + 2.0 * f(i);

        // Serial code
        #pragma omp master
        do_stuff_last();
    }
}
```

#pragma omp barrier

Потоки ждут пока все не достигнут этого места в программе

Нормализация яркости изображения

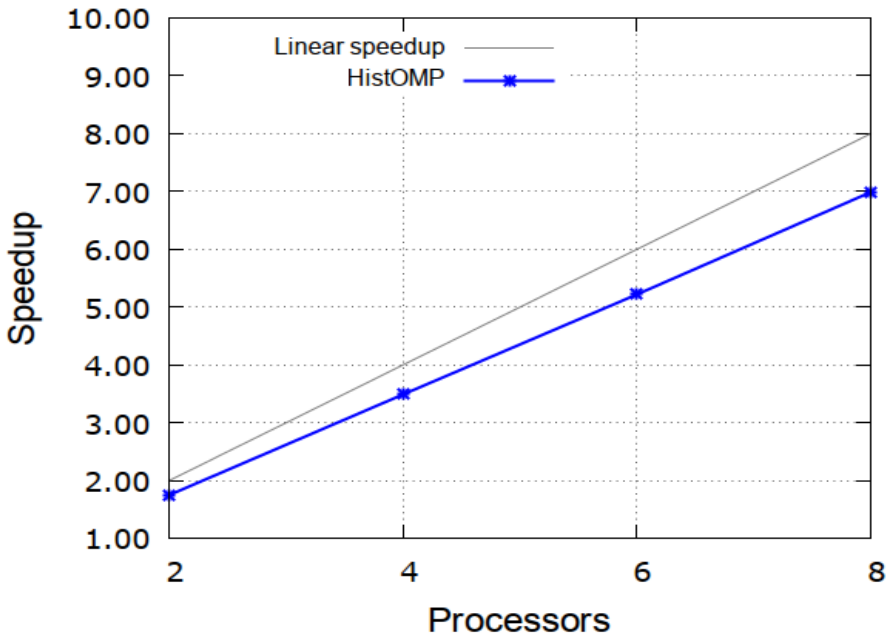
```
const uint64_t width = 32 * 1024;  const uint64_t height = 32 * 1024;

void hist_serial(uint8_t *pixels, int height, int width)
{
    uint64_t npixels = height * width;

    int *h = xmalloc(sizeof(*h) * 256);
    for (int i = 0; i < 256; i++)
        h[i] = 0;

    for (int i = 0; i < npixels; i++)
        h[pixels[i]]++;

    int mini, maxi;
    for (mini = 0; mini < 256 && h[mini] == 0; mini++);
    for (maxi = 255; maxi >= 0 && h[maxi] == 0; maxi--);

    int q = 255 / (maxi - mini);
    for (int i = 0; i < npixels; i++)
        pixels[i] = (pixels[i] - mini) * q;
    free(h);
}

int main(int argc, char *argv[])
{
    uint64_t npixels = width * height;

    pixels1 = xmalloc(sizeof(*pixels1) * npixels);

    hist_serial(pixels1, height, width);
    // ...
}
```

h[i] — количество точек цвета i в изображении (гистограмма)

$$\mathrm{LUT}[i] = 255 \cdot \frac{i - I_{\min}}{I_{\max} - I_{\min}}.$$

Задание

- Разработать на OpenMP параллельную версию программы — написать код функции hist_omp
- Шаблон программы находится в каталоге _task



Вариант решения находиться в каталоге 7_hist