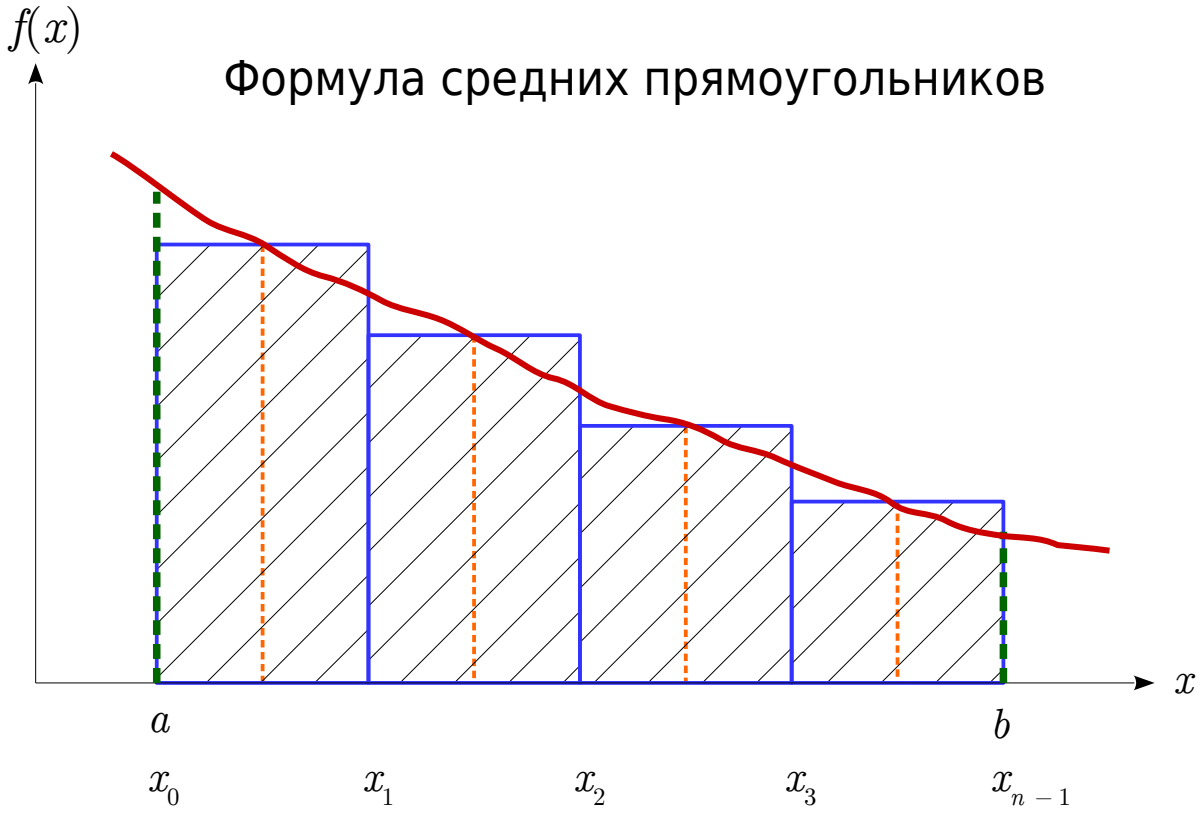
Семинар 3
Стандарт OpenMP (часть 3)

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

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;
}
```

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

#pragma omp atomic + локальная переменная

```
double integrate_omp(double (*func)(double), double a, double b, int n)
{
    double h = (b - a) / n;
    double sum = 0.0;

    #pragma omp parallel
    {
        int nthreads = omp_get_num_threads();
        int threadid = omp_get_thread_num();
        int items_per_thread = n / nthreads;
        int lb = threadid * items_per_thread;
        int ub = (threadid == nthreads - 1) ? (n - 1) : (lb + items_per_thread - 1);
        double sumloc = 0.0;

        for (int i = lb; i <= ub; i++)
            sumloc += func(a + h * (i + 0.5));

        #pragma omp atomic
        sum += sumloc;
    }
    sum *= h;
    return sum;
}
```

Вручную определяем диапазон

[lb, ub] значений i для каждого потока

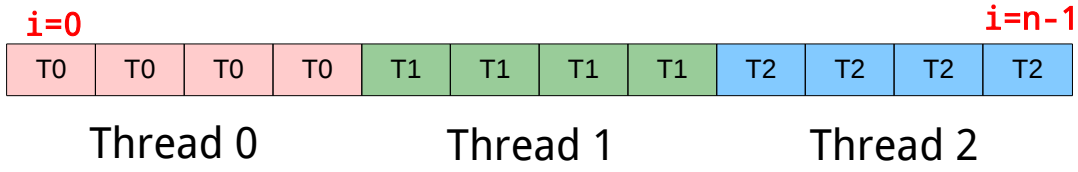
- Каждый поток накапливает сумму в своей локальной переменной sumloc
- Затем атомарной операцией вычисляется итоговая сумма

Распараллеливание циклов (#pragma omp for)

```
double integrate_omp(double (*func)(double), double a, double b, int n)
{
    double h = (b - a) / n;
    double sum = 0.0;

    #pragma omp parallel
    {
        double sumloc = 0.0;

        #pragma omp for
        for (int i = 0; i < n; i++)
            sumloc += func(a + h * (i + 0.5));

        #pragma omp atomic
        sum += sumloc;
    }
    sum *= h;
    return sum;
}
```

Разбивает пространство итераций на nthreads непрерывных смежных интервалов

Разбиение пространства итераций на смежные непрерывные части



Распределение итераций по потокам

Обозначения:

- ***p*** — число потоков в параллельном регионе (nthreads)
- ***q*** = ceil(*n* / *p*)
- ***n*** = *p* * *q* - *r*

```
#pragma omp for
for (int i = 0; i < n; i++)
```

Разбиение на p смежных непрерывных диапазона

- Первым p - r потокам достается по q итераций, остальным r потокам — по q - 1
- Пример для p = 3, n = 10 (n = 3 * 4 - 2):
  0 0 0 0 1 1 1 2 2 2

```
#pragma omp for schedule(static, k)
for (int i = 0; i < n; i++)
```

Циклическое распределение итераций (round-robin)

- Первые k итераций достаются потоку 0, следующие k итераций потоку 1, ..., k итераций потоку p - 1, и заново k итераций потоку 0 и т.д.
- Пример для p = 3, n = 10, k = 1 (n = 3 * 4 - 2):
  0 1 2 0 1 2 0 1 2 0

Распределение итераций по потокам

Обозначения:

- ***p*** — число потоков в параллельном регионе (nthreads)
- ***q*** = ceil(*n* / *p*)
- ***n*** = *p* * *q* - *r*

```
#pragma omp for schedule(dynamic, k)
for (int i = 0; i < n; i++)
```

Динамическое выделение блоков по k итераций

- Потоки получают по k итераций, по окончанию их обработки запрашивают еще k итераций и т.д.
- Заранее неизвестно какие итерации достанутся потокам (зависит от порядка и длительности их выполнения)

```
#pragma omp for schedule(guided, k)
for (int i = 0; i < n; i++)
```

Динамическое выделение уменьшающихся блоков

- Каждый поток получает n / p итераций
- По окончанию их обработки, из оставшихся n' итераций поток запрашивает n' / p

Подсчет количества простых чисел

```
const int a = 1;
const int b = 10000000;

int is_prime_number(int n)
{
    int limit = sqrt(n) + 1;
    for (int i = 2; i <= limit; i++) {
        if (n % i == 0)
            return 0;
    }
    return (n > 1) ? 1 : 0;
}

int count_prime_numbers(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++;

    /* Loop over odd numbers: a, a + 2, a + 4, ... , b */
    for (int i = a; i <= b; i += 2) {
        if (is_prime_number(i))
            nprimes++;
    }
    return nprimes;
}
```

Определят, является ли число n простым

Подсчитывает количество простых числе в интервале [a, b]

Подсчет количества простых чисел (OpenMP)

```
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
        for (int i = a; i <= b; i += 2) {
            if (is_prime_number(i))
                nloc++;
        }

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

Разбили интервал [a, b] проверяемых чисел
на смежные непрерывные отрезки

Неравномерная загрузка потоков (load imbalance)

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

    if (a <= 2) {
        nprimes = 1;
        a = 2;
    }
    if (a % 2 == 0)
        a++;

    #pragma omp parallel
    {
        double t = omp_get_wtime();
        int nloc = 0;

        #pragma omp for 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;

        t = omp_get_wtime() - t;
        printf("Thread %d execution time: %.6f sec.\n",
               omp_get_thread_num(), t);
    }
    return nprimes;
}
```

```
$ OMP_NUM_THREADS=4 ./primes

Count prime numbers on [1, 10000000]
Result (serial): 664579
Thread 0 execution time: 1.789976
Thread 1 execution time: 2.961944
Thread 2 execution time: 3.004635
Thread 3 execution time: 3.588935
Result (parallel): 664579
Execution time (serial): 7.190282
Execution time (parallel): 3.590609
Speedup: 2.00
```

Load imbalance

Потоки загружены не равномерно (load imbalance) — потоки с большими номерами выполняются дольше

Причина?

Неравномерная загрузка потоков (load imbalance)

```
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
        for (int i = a; i <= b; i += 2) {
            if (is_prime_number(i))
                nloc++;
        }

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

Неэффективное распределение итераций по потокам

- Время выполнения функции is_prime_number(i) зависит от значения i
- Потокам с большими номерами достались большие значения i

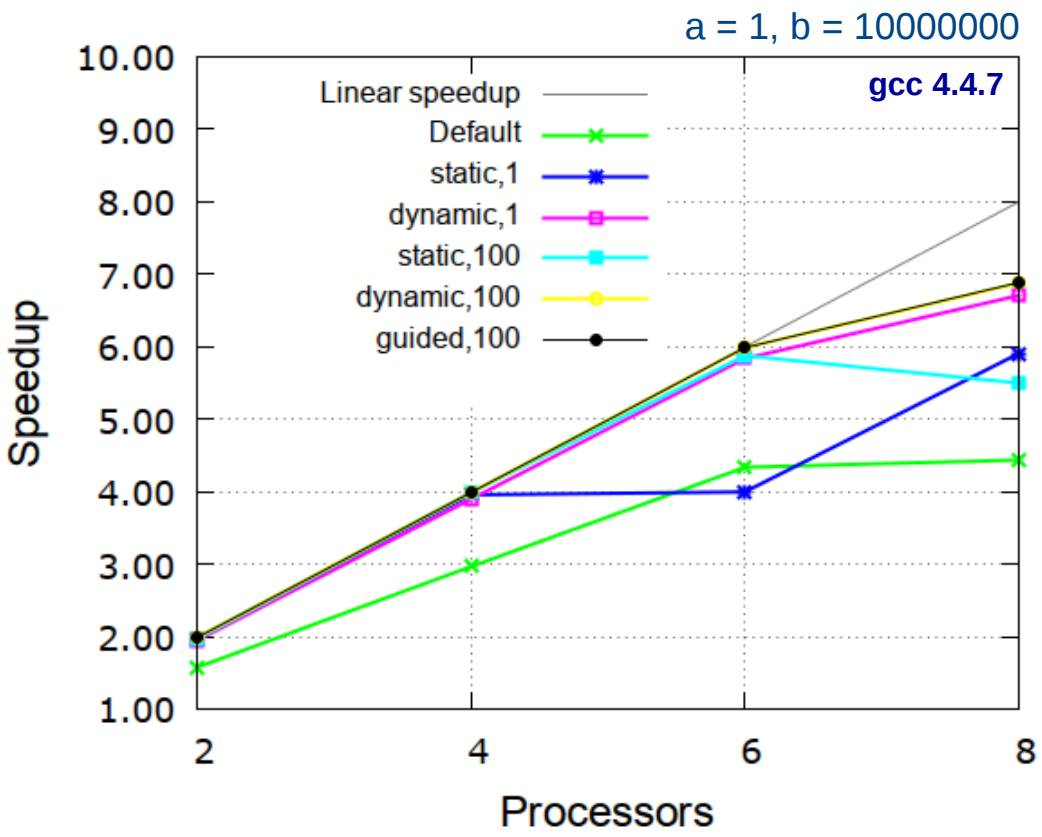
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3

Динамическое распределение итераций

```
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;
}
```

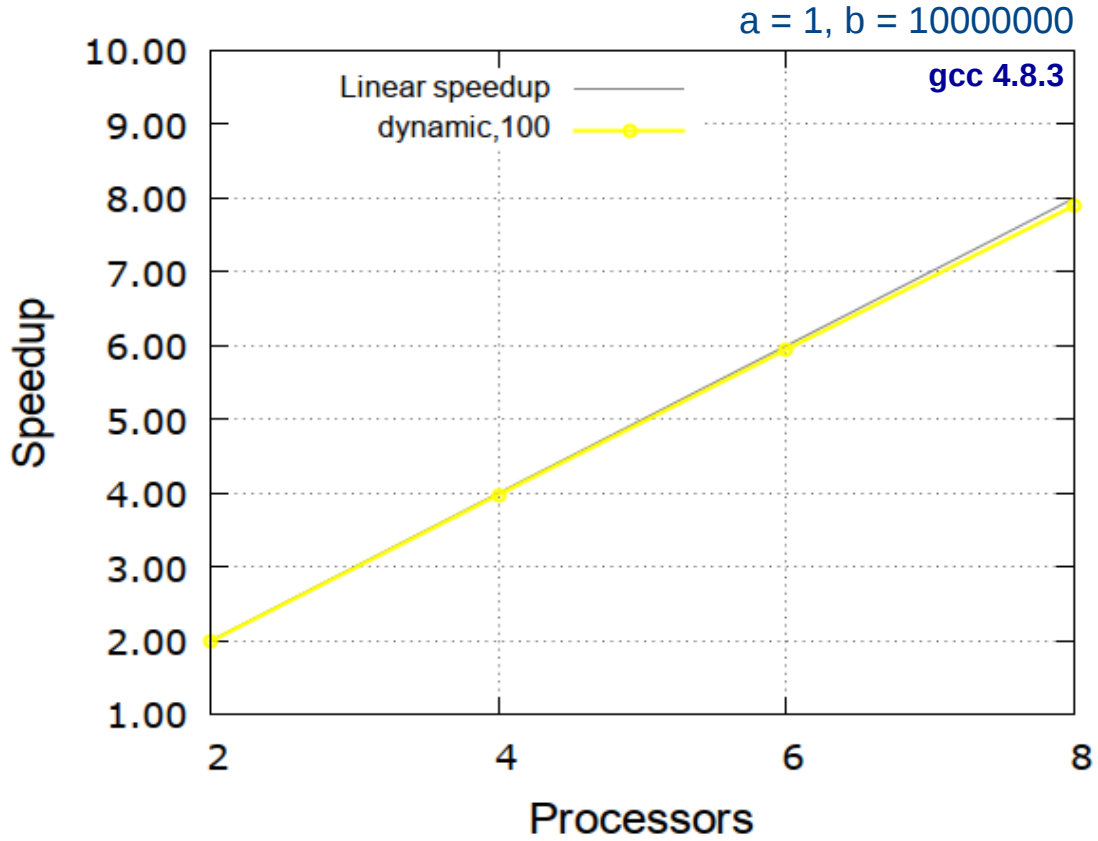
Динамическое распределение итераций блоками по 100 элементов

Анализ эффективности



Вычислительный узел кластера Oak (NUMA)

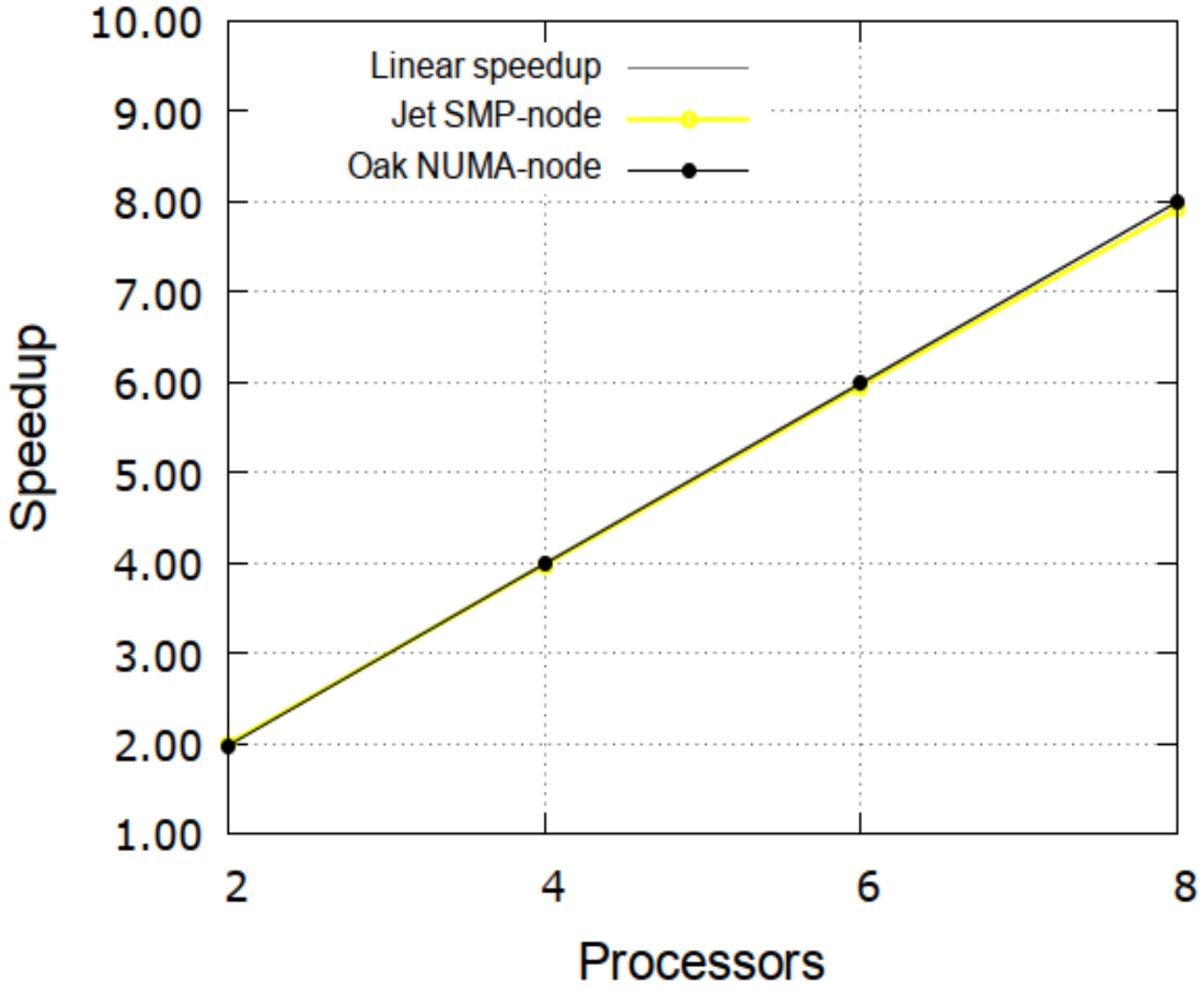
- **8 ядер** (два Intel Quad Xeon E5620)
- **24 GiB RAM** (6 x 4GB DDR3 1067 MHz)
- CentOS 6.5 x86_64 (kernel 2.6.32), **GCC 4.4.7**



Вычислительный узел кластера Jet (SMP)

- **8 ядер** (два Intel Quad Xeon E5420)
- **8 GiB RAM**
- Fedora 20 x86_64 (kernel 3.11.10), **GCC 4.8.3**

Привязка потоков к процессорам




```
export GOMP_CPU_AFFINITY="0-7"

export OMP_NUM_THREADS=8

./primes
```

Задание

Реализовать параллельную версию программы подсчета числа простых чисел в заданном интервале

- написать код функции count_prime_numbers_omp с использованием #pragma omp for и локальной переменной (слайд 11)
- добиться равномерной загрузки потоков, оценить ускорение программы на 2, 4, 6 и 8 потоках

Шаблон программы находится в каталоге _task