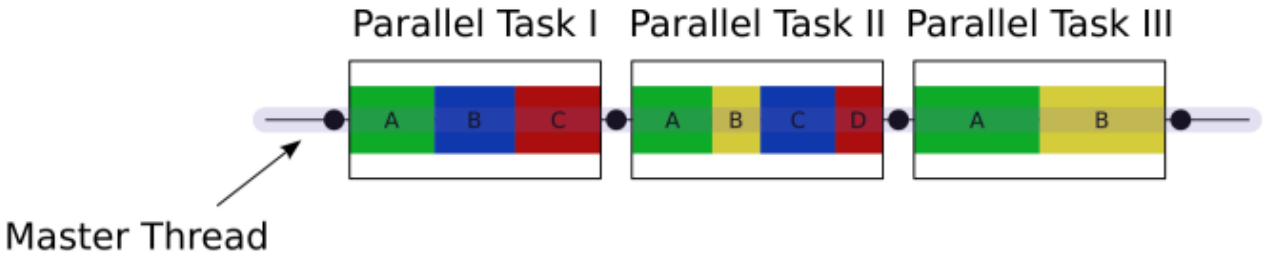
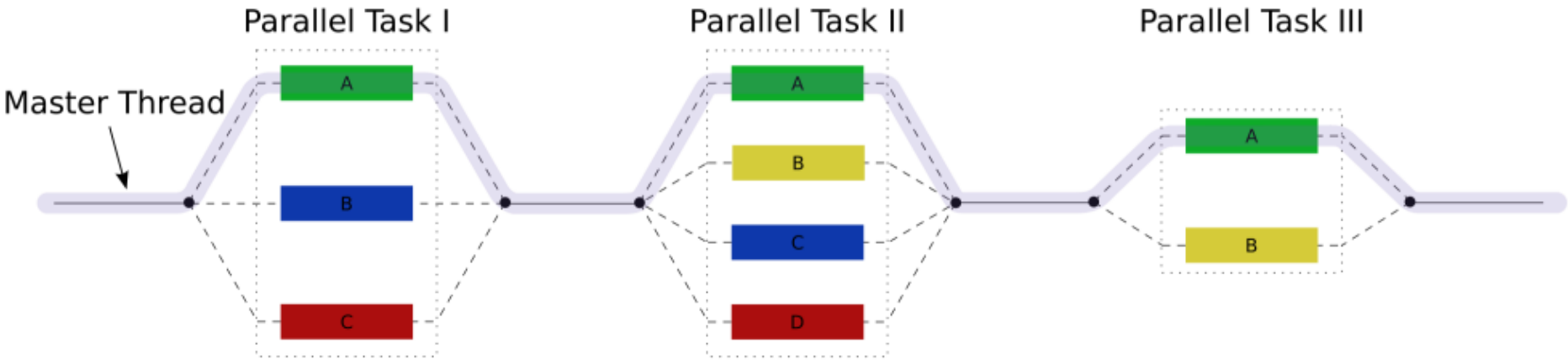
Система с общей памятью OpenMP

Практическое занятие #2

Принцип работы OpenMP





Вычисление числа Пи

$$\pi = \sum_{i=0}^{N} \frac{4}{1 + x_i^2} \qquad x_i = \frac{\left(i + \frac{1}{2}\right)}{N}$$

```
int Pi= 0;
double step = 1.0 / N;
double x, Pi = 0;

#pragma omp parallel for private(x,i) reduction(+:Pi)
for(i = 0; i < N; i++) {
    x = step * (0.5 + i);
    Pi += 4.0 / (1 + x*x);
}
```

Типы переменных

- **`shared`** – доступно всем потокам (по-умолчанию)
- **`private`** – у каждого потока своя копия
- **`firstprivate`** – у каждого потока своя переменная, которая инициализируется значением оригинальной
- **`lastprivate`** – при завершении параллельного блока обновляет переменную значением из последней нити

Firstprivate

```
class Example {

   …

};

Example foo;

1) #pragma omp parallel shared(foo) { … }

2) #pragma omp parallel private(foo) { … }
   ?

3) #pragma omp parallel firstprivate(i,x) { … }
   ?
```

Firstprivate

```
class Example {
  int *a;
  Example() { a = new int[10]; }
  Example(Example &b) { a = copy(b.a); }
};

Example foo;

1) #pragma omp parallel shared(foo) { … }

2) #pragma omp parallel private(foo) { … }
   Копирование памяти

3) #pragma omp parallel firstprivate(i,x) { … }
   Вызов конструктора копирования
```

lastprivate

```
int count = omp_get_max_threads()
double tmp[count];

#pragma omp parallel private(i) lastprivate(x)
{
    int id = omp_get_thread_num();
    for(i = id; i < N; i += count) {
        x = step * (0.5 + i);
        tmp[id] += 4.0 / (1 + x*x);
    }
}

for(i = 0; i < count; i++) {
    Pi += tmp[i];
}
```

Распределение итераций

```
#pragma omp parallel for
for(i = 0; i < N; i++) { … }
```

Schedule – механизм разделения итераций по нитям

static,m – статические блоки по m итераций

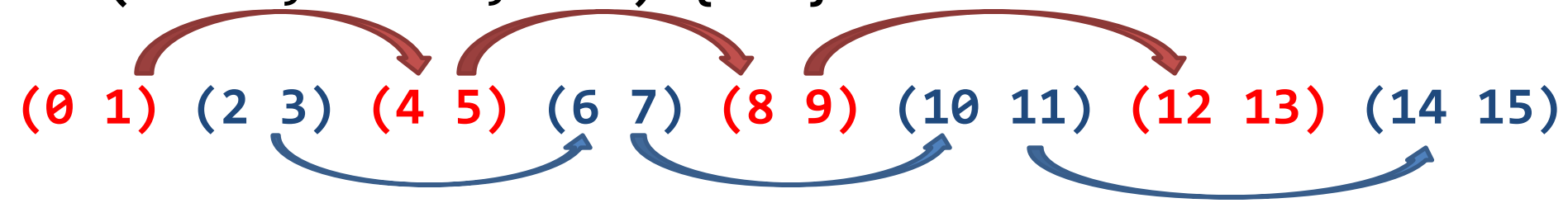
dynamic,m – динамически блоки по m итераций

guided,m – экспоненциальное уменьшение размеров блоков, вплоть до m итераций в блоке

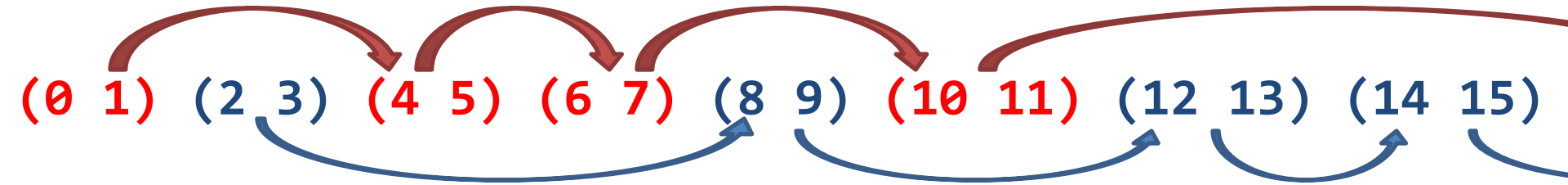
runtime – определяется в процессе работы

Распределение итераций

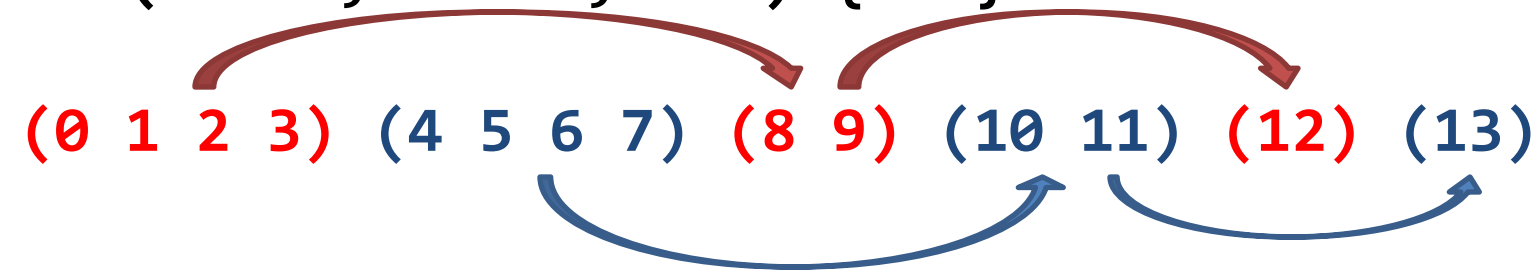
```
#pragma omp parallel for schedule(static,2)
for(i = 0; i < N; i++) { … }
```




```
#pragma omp parallel for schedule(dynamic,2)
for(i = 0; i < N; i++) { … }
```




```
#pragma omp parallel for schedule(guided,1)
for(i = 0; i < N; i++) { … }
```



Синхронизация

`#pragma omp single {…}`
Код будет выполнен только первой нитью, которая дойдет до блока кода

`#pragma omp master {…}`
Код будет выполнен только нитью номер 0

`#pragma omp critical {…}` – Критическая секция
Блок не выполняется двумя и более нитями одновременно

`#pragma omp barrier`
Точка синхронизации всех нитей

Синхронизация

$$\pi = \sum_{i=0}^{N} \frac{4}{1 + {x_i}^2} \qquad x_i = \frac{\left(i + \frac{1}{2}\right)}{N}$$

```
int Pi= 0;
double step = 1.0 / N;
double x, Pi = 0;

#pragma omp parallel for private(x,i)
for(i = 0; i < N; i++) {
    x = step * (0.5 + i);

    #pragma omp critical
    Pi += 4.0 / (1 + x*x);
}
```

Синхронизация

$$\pi = \sum_{i=0}^{N} \frac{4}{1 + x_i{}^2} \qquad x_i = \frac{\left(i + \frac{1}{2}\right)}{N}$$

```
int Pi= 0;
double step = 1.0 / N;
double x, Pi = 0;

#pragma omp parallel for private(x,i)
for(i = 0; i < N; i++) {
    x = step * (0.5 + i);

    #pragma omp atomic
    Pi += 4.0 / (1 + x*x);
}
```

Синхронизация

```
#pragma omp parallel for ordered
for(i = 0; i < count; i++) {
   …
   #pragma omp ordered
   {
      printf(“point #%d\n”, i);
   }
}
```

Блок внутри цикла будет вызываться по очередности счетчика `i`

Синхронизация

```
#pragma omp parallel for shared(x)
for(i = 0; i < count; i++) {
  x = …
  #pragma omp flush(x)
}
```

Оператор flush позволяет обновить значение переменной для всех нитей.

Синхронизация

```
#pragma omp parallel
{
  #pragma omp for nowait
  for(i = 0; i < count; i++) {…}

  #pragma omp for
  for(j = 0; j < count; j++) {…}

  #pragma omp barrier
  call_function()
}
```

Директива if

```
int count = omp_get_max_threads()

#pragma omp parallel if(N > count)
{
  Если условие выполнено,
    то код выполняется параллельно.

  Если условие неверное,
    то код выполняется как обычный последовательный.
}
```

Домашнее задание вариант 1

Разработать параллельную версию алгоритма K-Means.

http://ru.wikipedia.org/wiki/K-means

Домашнее задание вариант 2

Реализовать параллельное решение задачи теплопроводности в одномерном случае.

$$u_t = a^2 u_{xx,} \quad u(0) = 1, u(1) = 0$$

Алгоритм должен использовать метод приграничных полос.

Вместо теплопроводности возможно решение любой задачи вычислительной механики.