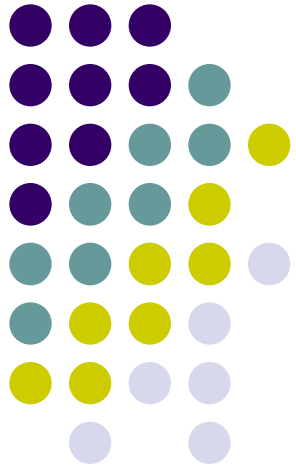
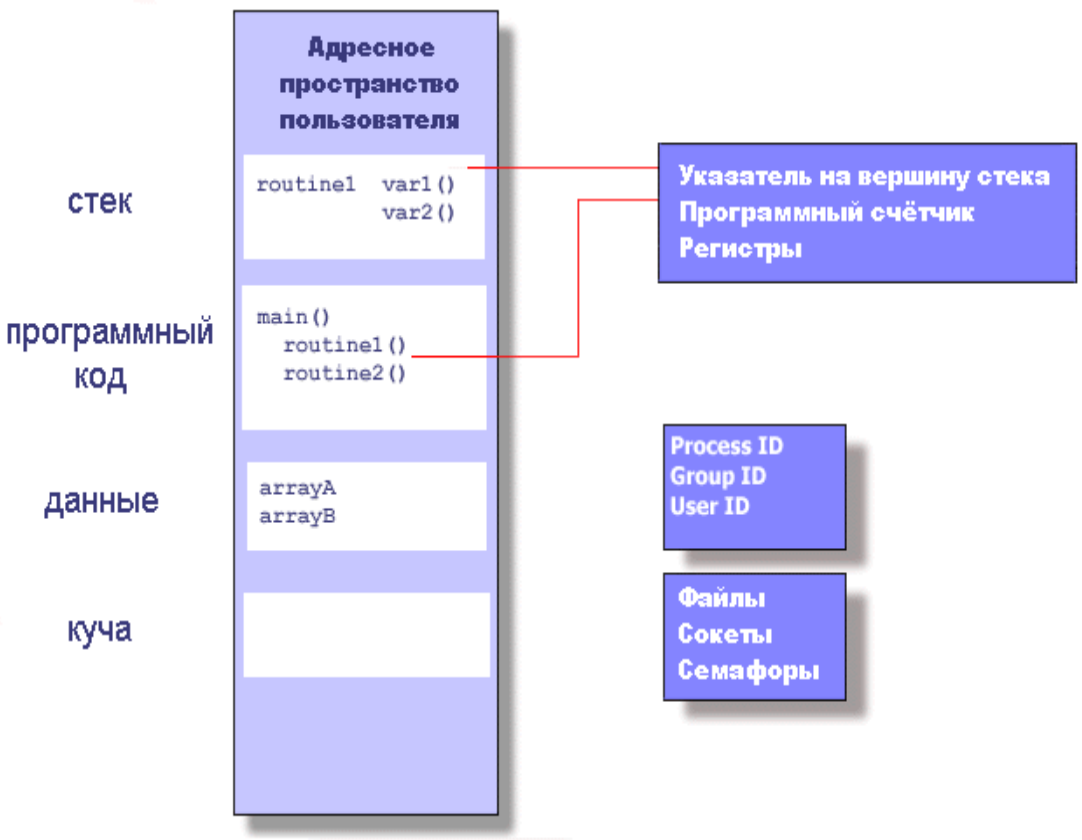
OpenMP: Краткий обзор

Киреев Сергей
ИВМиМГ СО РАН



Процессы и потоки

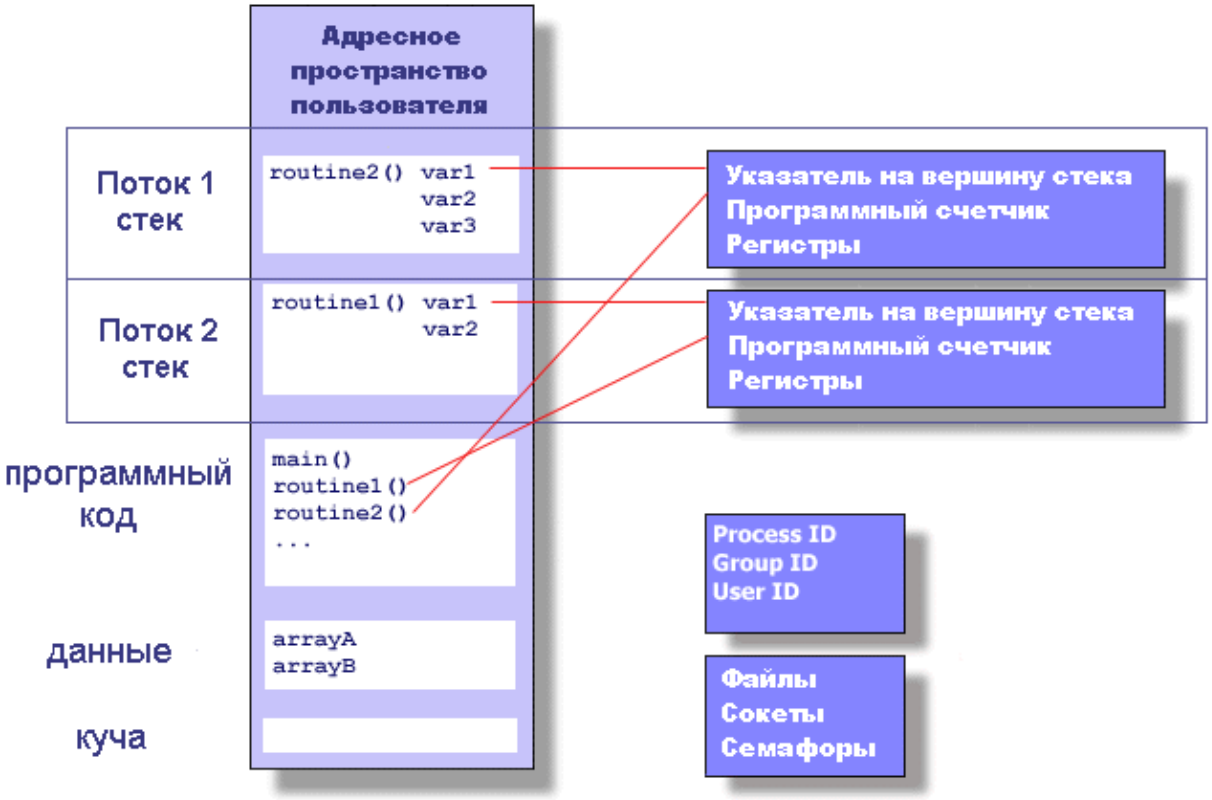
Процесс – это среда выполнения задачи (программы).



Процесс создаётся ОС и содержит информацию о программных ресурсах и текущем состоянии выполнения программы.

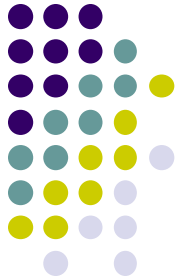
Процессы и потоки

Поток – это «облегченный процесс».

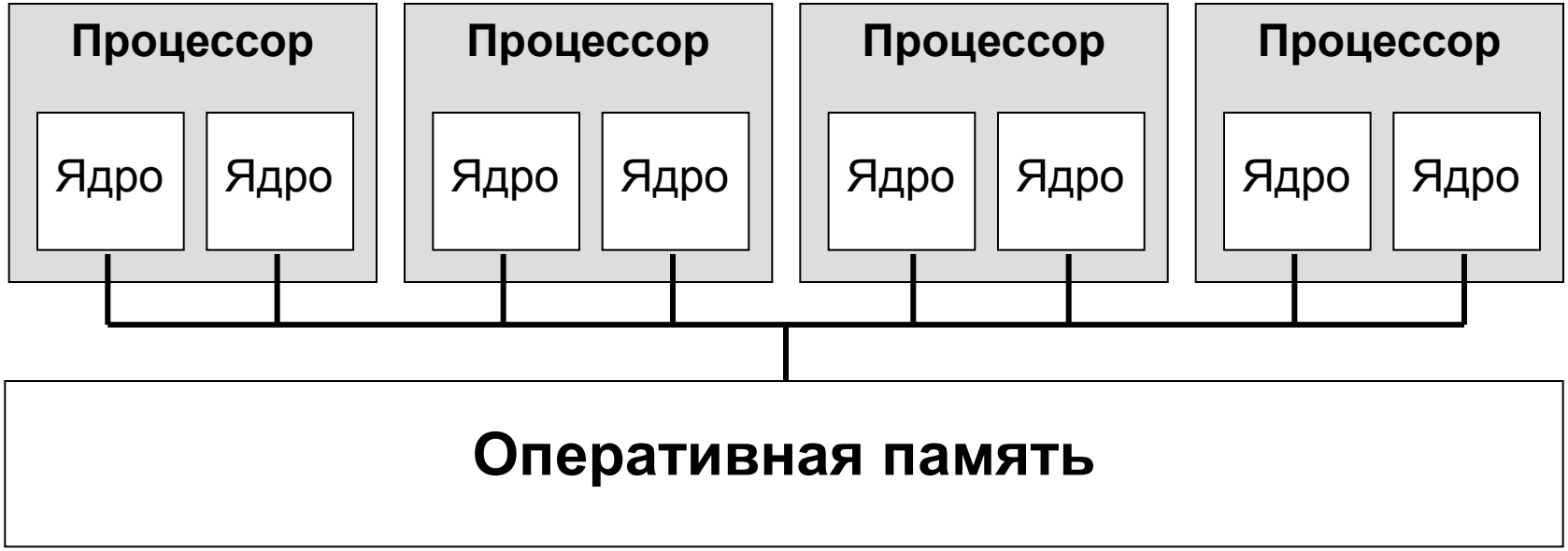


- Создается в рамках процесса,
- Имеет свой поток управления,
- Разделяет ресурсы процесса-родителя с другими потоками,
- Погибает, если погибает родительский процесс.

Многопоточное программирование

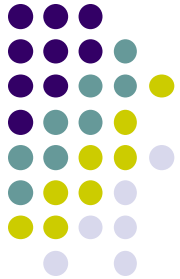


- Используется для создания параллельных программ для систем с общей памятью



- И для других целей…

OpenMP – это…



- Стандарт интерфейса для многопоточного программирования над общей памятью

- Набор средств для языков C/C++ и Fortran:
  - Директивы компилятора

    #pragma omp …
  - Библиотечные подпрограммы

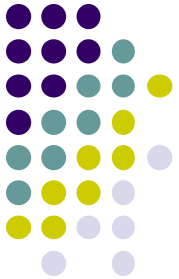
    get_num_threads()
  - Переменные окружения

    OMP_NUM_THREADS

Порядок создания параллельных программ с использованием OpenMP

1. Написать и отладить последовательную программу
2. Дополнить программу директивами OpenMP
3. Скомпилировать программу компилятором с поддержкой OpenMP
4. Задать переменные окружения
5. Запустить программу

Вводный пример: последовательная программа



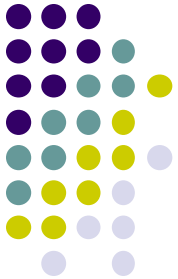
```
#include <math.h>
#define N 10000
float x[N];

int main()
{  int i;
   float k = 2*3.14159265/N;


   for (i=0;i<N;i++) x[i]=sinf(k*i);

   return 0;
}
```

Вводный пример: параллельная программа



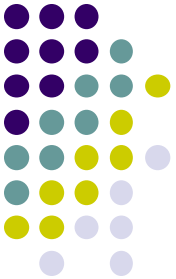
```
#include <math.h>
#define N 10000
float x[N];

int main()
{  int i;
   float k = 2*3.14159265/N;

   #pragma omp parallel for
   for (i=0;i<N;i++) x[i]=sinf(k*i);

   return 0;
}
```

Компиляция с OpenMP



- GNU C/C++ Compiler

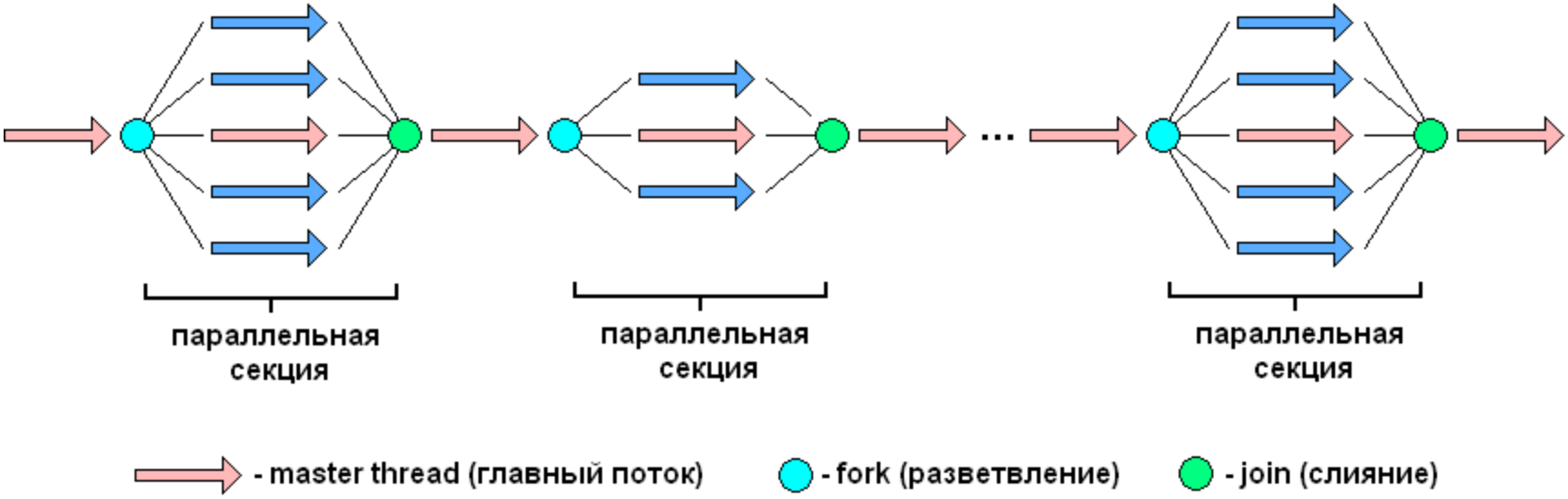
```
gcc -fopenmp -o prog prog.c
g++ -fopenmp -o prog prog.cpp
```

- Intel C/C++ Compiler

```
icc -openmp -o prog prog.c
icpc -openmp -o prog prog.cpp
```

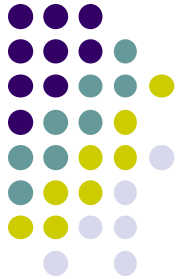
Модель программирования

- Fork-join параллелизм



- Явное указание параллельных секций
- Поддержка вложенного параллелизма
- Поддержка динамических потоков

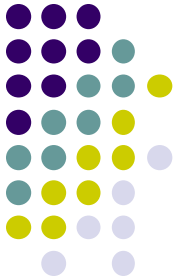
Объявление параллельной секции



```
#include <omp.h>
int main()
{
  // последовательный код
  #pragma omp parallel
  {
    // параллельный код
  }
  // последовательный код

  return 0;
}
```

Условное объявление параллельной секции

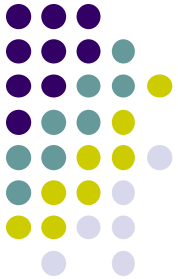


```
#include <omp.h>
int main()
{
  // последовательный код
  #pragma omp parallel if (expr)
  {
    // параллельный код
  }
  // последовательный код

  return 0;
}
```

Пример:

Hello, World!



```
#include <stdio.h>
#include <omp.h>
int main()
{ printf("Hello, World!\n");
  #pragma omp parallel
  { int i,n;
    i = omp_get_thread_num();
    n = omp_get_num_threads();
    printf("I'm thread %d of %d\n",i,n);
  }
  return 0;
}
```

Пример:

Hello, World!

```
#include <stdio.h>
#include <omp.h>
int main()
{ printf("Hello, World!\n");
  #pragma omp parallel
  { int i,n;
    i = omp_get_thread_num();
    n = omp_get_num_threads();
    printf("I'm thread %d of %d\n",i,n);
  }
  return 0;
}
```

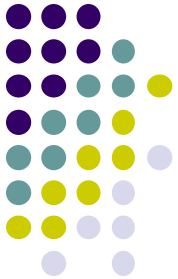
- **Компиляция:**

```
> gcc -fopenmp -o hello hello.c
```

- **Запуск:**

```
> OMP_NUM_THREADS=4 ./hello
Hello, World!
I'm thread 1 of 4
I'm thread 0 of 4
I'm thread 3 of 4
I'm thread 2 of 4
```

Задание числа потоков



- Переменная окружения OMP_NUM_THREADS

```
>OMP_NUM_THREADS=4 ; ./a.out
```

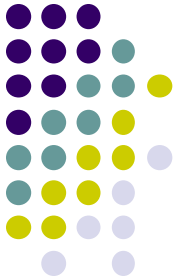
- Функция omp_set_num_threads(int)

```
omp_set_num_threads(4);
#pragma omp parallel
{ . . .
}
```

- Параметр num_threads

```
#pragma omp parallel num_threads(4)
{ . . .
}
```

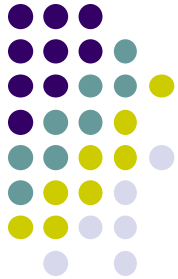
Получение числа потоков



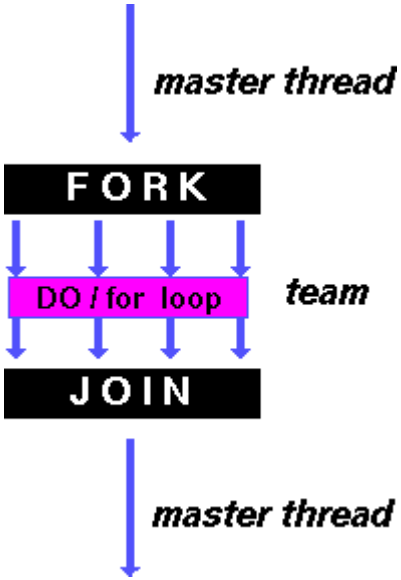
- Функция omp_get_num_threads()

```
#pragma omp parallel
{ …
  int n = omp_get_num_threads();
  …
}
```

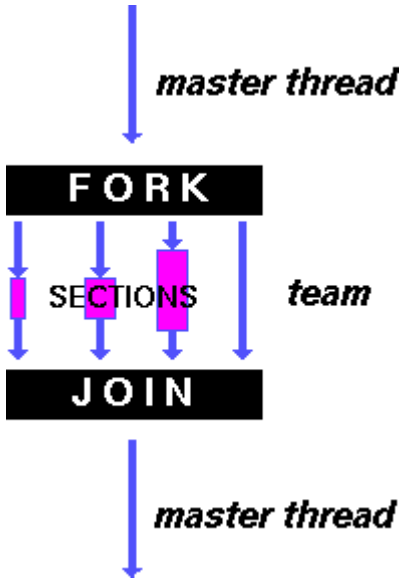
Способы разделения работы между потоками



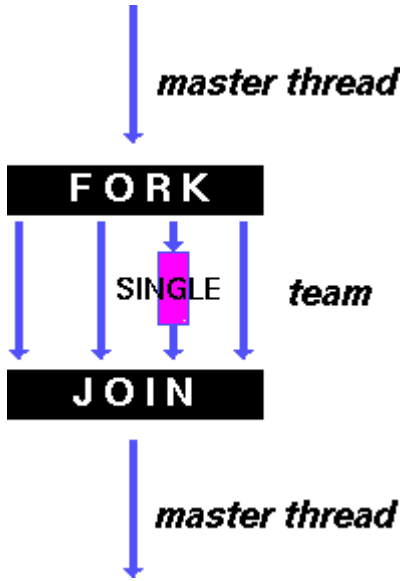
```
#pragma omp for
  for (i=0;i<N;i++)
  {
    // code
  }
```




```
#pragma omp sections
{
 #pragma omp section
 // code 1
 #pragma omp section
 // code 2
}
```

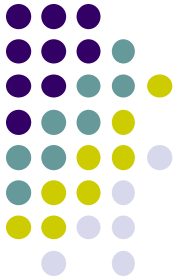



```
#pragma omp single
{
   // code
}
```



Пример:

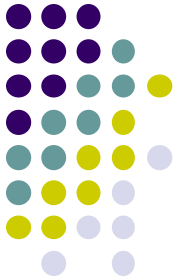
Директива omp for



```
#include <stdio.h>
#include <omp.h>
int main()
{ int i;
  #pragma omp parallel
  {
    #pragma omp for
    for (i=0;i<1000;i++)
      printf("%d ",i);
  }
  return 0;
}
```

Пример:

Директива omp for



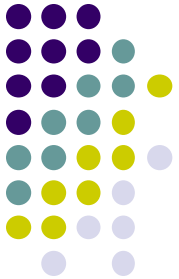
```
#include <stdio.h>
#include <omp.h>
int main()
{ int i;

  #pragma omp parallel for
    for (i=0;i<1000;i++)
      printf("%d ",i);

  return 0;
}
```

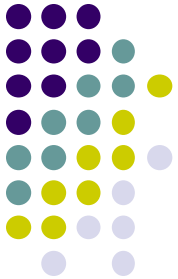
Пример:

Директива omp sections



```
#include <stdio.h>
#include <omp.h>
int main()
{ int i;
  #pragma omp parallel sections private(i)
  {
    #pragma omp section
    { printf("1st half\n");
      for (i=0;i<500;i++) printf("%d ",i);
    }
    #pragma omp section
    { printf("2nd half\n");
      for (i=501;i<1000;i++) printf("%d ",i);
    }
  }
  return 0;
}
```

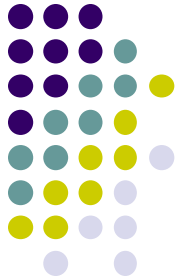
Пример:
Директива omp single



```
#include <stdio.h>
#include <omp.h>
int main()
{ int i;
  #pragma omp parallel private(i)
  {
    #pragma omp for
      for (i=0;i<1000;i++) printf("%d ",i);
    #pragma omp single
      printf("I'm thread %d!\n",get_thread_num());
    #pragma omp for
      for (i=0;i<1000;i++) printf("%d ",i);
  }
  return 0;
}
```

Пример:

Директива omp single

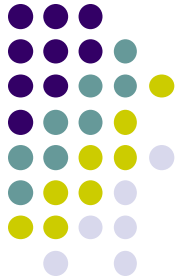


```
#include <stdio.h>
#include <omp.h>
int main()
{ int i;
  #pragma omp parallel private(i)
  {
    #pragma omp for
      for (i=0;i<1000;i++) printf("%d ",i);
    #pragma omp single
      printf("I'm thread %d!\n",get_thread_num());
    #pragma omp for
      for (i=0;i<1000;i++) printf("%d ",i);
  }
  return 0;
}
```

барьер

Пример:

Директива omp single

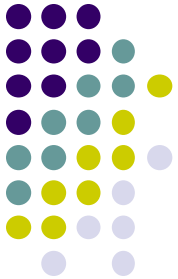


```
#include <stdio.h>
#include <omp.h>
int main()
{ int i;
  #pragma omp parallel private(i)
  {
    #pragma omp for
      for (i=0;i<1000;i++) printf("%d ",i);
    #pragma omp single nowait
      printf("I'm thread %d!\n",get_thread_num());
    #pragma omp for
      for (i=0;i<1000;i++) printf("%d ",i);
  }
  return 0;
}
```

нет барьера

Пример:

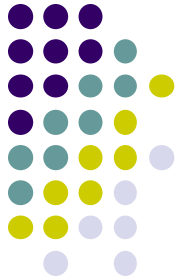
Директива omp master



```
#include <stdio.h>
#include <omp.h>
int main()
{ int i;
  #pragma omp parallel private(i)
  {
    #pragma omp for
      for (i=0;i<1000;i++) printf("%d ",i);
    #pragma omp master
      printf("I'm Master!\n")
    #pragma omp for
      for (i=0;i<1000;i++) printf("%d ",i);
  }
  return 0;
}
```

нет барьера

Способы разделения работы между потоками

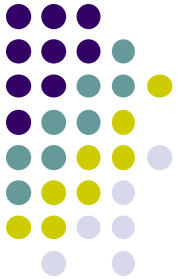


- Параллельное исполнение цикла for

  #pragma omp for *параметры:*
  - **schedule** - распределения итераций цикла между потоками
    - **schedule(static, n)** – статическое распределение
    - **schedule(dynamic, n)** – динамическое распределение
    - **schedule(guided, n)** – управляемое распределение
    - **schedule(runtime)** – определяется OMP_SCHEDULE
  - **nowait** – отключение синхронизации в конце цикла
  - **ordered** – выполнение итераций в последовательном порядке
  - **collapse(n)** – объединить n вложенных циклов в одно итерационное пространство
  - Параметры области видимости переменных…

Пример:

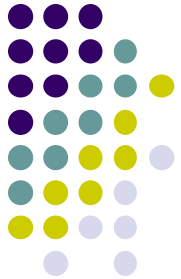
Директива omp for



```
#include <stdio.h>
#include <omp.h>
int main()
{ int i;

  #pragma omp parallel private(i)
  {
    #pragma omp for schedule(static,10) nowait
      for (i=0;i<1000;i++) printf("%d ",i);
    #pragma omp for schedule(dynamic,1)
      for (i='a';i<='z';i++) printf("%c ",i);
  }
  return 0;
}
```

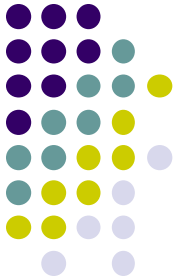
Области видимости переменных



- Переменные, объявленные внутри параллельного блока, являются локальными для потока:

```
#pragma omp parallel
{
  int num = omp_get_thread_num();
  printf("Thread %d\n",num);
}
```

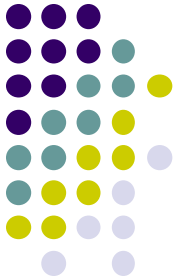
Области видимости переменных



- Переменные, объявленные вне параллельного блока, по умолчанию являются общими для всех потоков:

```
int n = 10;
#pragma omp parallel
{
  for (int i=0;i<n;i++)
    printf("%d\n",i);
}
```

Области видимости переменных

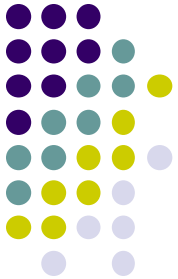


- Переменные, объявленные вне параллельного блока, по умолчанию являются общими для всех потоков:

```
int num;
#pragma omp parallel
{
  num = omp_get_thread_num();
  printf("Thread %d\n",num);
}
```

?

Области видимости переменных

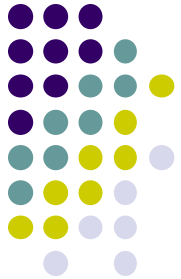


- Переменные, объявленные вне параллельного блока, по умолчанию являются общими для всех потоков:

```
int num;
#pragma omp parallel
{
  num = omp_get_thread_num();
  printf("Thread %d\n",num);
}
```

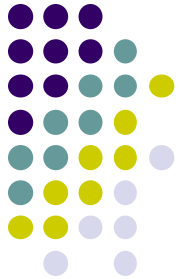
Результат не определён!

Области видимости переменных



- Область видимости переменных, объявленные вне параллельного блока, определяются параметрами директив:
  - private
  - firstprivate
  - lastprivate
  - shared
  - default
  - reduction
  - threadprivate
  - copyin

Области видимости переменных

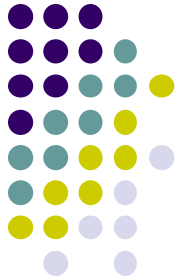


- Область видимости переменных, объявленные вне параллельного блока, определяются параметрами директив:
  - **private**
  - firstprivate
  - lastprivate
  - shared
  - default
  - reduction
  - threadprivate
  - copyin

Своя локальная переменная в каждом потоке

```
int num;
#pragma omp parallel private(num)
{
  num = omp_get_thread_num();
  printf("%d\n",num);
}
```

Области видимости переменных

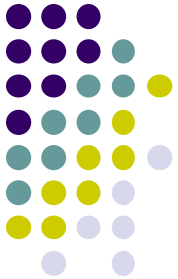


- Область видимости переменных, объявленные вне параллельного блока, определяются параметрами директив:
  - private
  - **firstprivate**
  - lastprivate
  - shared
  - default
  - reduction
  - threadprivate
  - copyin

Локальная переменная с инициализацией

```
int num = 5;
#pragma omp parallel \
                firstprivate(num)
{
  printf("%d\n",num);
}
```

Области видимости переменных



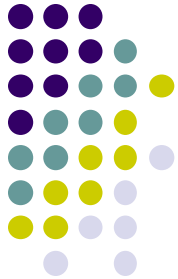
- Область видимости переменных, объявленные вне параллельного блока, определяются параметрами директив:
  - private
  - firstprivate
  - **lastprivate**
  - shared
  - default
  - reduction
  - threadprivate
  - copyin

Локальная переменная с сохранением последнего значения (в последовательном исполнении)

```
int i,j;
#pragma omp parallel for \
                lastprivate(j)
for (i=0;i<100;i++) j = i;

printf("Последний j = %d\n",j);
```

Области видимости переменных



- Область видимости переменных, объявленные вне параллельного блока, определяются параметрами директив:
  - private
  - firstprivate
  - lastprivate
  - **shared**
  - default
  - reduction
  - threadprivate
  - copyin

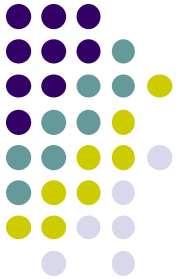
Разделяемая (общая) переменная

```
int i,j;

#pragma omp parallel for \
                shared(j)
for (i=0;i<100;i++) j = i;

printf("j = %d\n",j);
```

Области видимости переменных

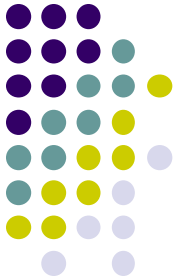


- Область видимости переменных, объявленные вне параллельного блока, определяются параметрами директив:
  - private
  - firstprivate
  - lastprivate
  - shared
  - **default**
  - reduction
  - threadprivate
  - copyin

Задание области видимости не указанных явно переменных

```
int i,k,n = 2;
#pragma omp parallel shared(n)
  default(none) private(i,k)
{
  i = omp_get_thread_num() / n;
  k = omp_get_thread_num() % n;
  printf("%d %d %d\n",i,k,n);
}
```

Области видимости переменных



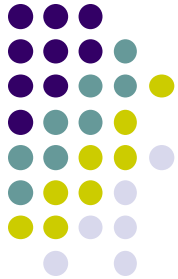
- Область видимости переменных, объявленные вне параллельного блока, определяются параметрами директив:
  - private
  - firstprivate
  - lastprivate
  - shared
  - default
  - **reduction**
  - threadprivate
  - copyin

Переменная для выполнения редукционной операции

```
int i,s = 0;
#pragma omp parallel for \
                reduction(+:s)
  for (i=0;i<100;i++)
    s += i;

printf("Sum: %d\n",s);
```

Области видимости переменных

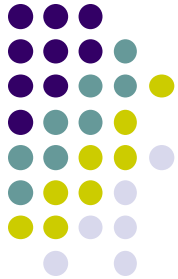


- Область видимости переменных, объявленные вне параллельного блока, определяются параметрами директив:
  - private
  - firstprivate
  - lastprivate
  - shared
  - default
  - reduction
  - **threadprivate**
  - copyin

Объявление глобальных переменных локальными для потоков

```
int x;
#pragma omp threadprivate(x)
int main()
{ . . .
  #pragma omp parallel
  . . .
}
```

Области видимости переменных

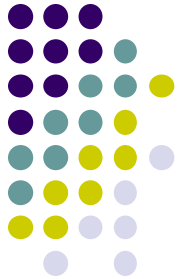


- Область видимости переменных, объявленные вне параллельного блока, определяются параметрами директив:
  - private
  - firstprivate
  - lastprivate
  - shared
  - default
  - reduction
  - threadprivate
  - **copyin**

Объявление глобальных переменных локальными для потоков с инициализацией

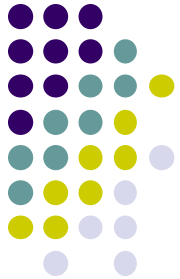
```
int x;
#pragma omp threadprivate(x)
int main()
{ . . .
  #pragma omp parallel copyin(x)
  . . .
}
```

Синхронизация потоков



- Директивы синхронизации потоков:
  - master
  - barrier
  - critical
  - atomic
  - flush
  - ordered
- Блокировки
  - omp_lock_t

Синхронизация потоков

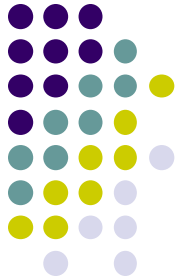


- Директивы синхронизации потоков:
  - **master**
  - barrier
  - critical
  - atomic
  - flush
  - ordered

Выполнение кода только главным потоком

```
#pragma omp parallel
{
   //code
   #pragma omp master
   {
      // critical code
   }
   // code
}
```

Синхронизация потоков



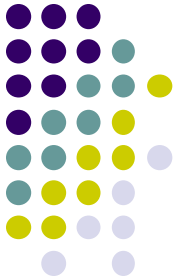
- Директивы синхронизации потоков:
  - master
  - **barrier**
  - critical
  - atomic
  - flush
  - ordered

Барьер

```
#pragma omp parallel
{
  printf("Hello!\n");

  #pragma omp barrier
  printf("I am thread %d\n",
          omp_get_thread_num());
}
```

Синхронизация потоков



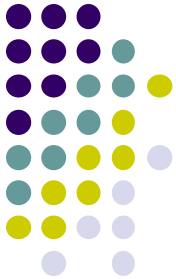
- Директивы синхронизации потоков:
  - master
  - barrier
  - **critical**
  - atomic
  - flush
  - ordered

Критическая секция

```
int i,idx[N],x[M];

#pragma omp parallel for
for (i=0;i<N;i++)
{
  #pragma omp critical
  {
    x[idx[i]] += count(i);
  }
}
```

Синхронизация потоков



- Директивы синхронизации потоков:
  - master
  - barrier
  - **critical**
  - atomic
  - flush
  - ordered

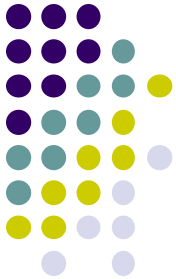
Критическая секция

```
int i,idx[N],x[M];

#pragma omp parallel for
for (i=0;i<N;i++)
{ int j = idx[i];
  int c = count(i);
  #pragma omp critical
  {
    x[j] += c;
  }
}
```

Так быстрее!

Синхронизация потоков



- Директивы синхронизации потоков:
  - master
  - barrier
  - critical
  - **atomic**
  - flush
  - ordered

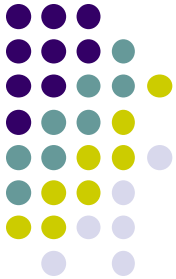
Атомарная операция с памятью

```
int i,idx[N],x[M];

#pragma omp parallel for
for (i=0;i<N;i++)
{
  #pragma omp atomic
    x[idx[i]] += count(i);
}
```

Так тоже!

Синхронизация потоков

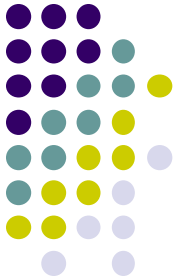


- Директивы синхронизации потоков:
  - master
  - barrier
  - critical
  - atomic
  - **flush**
  - ordered

Согласование значений переменных между потоками

```
int x = 0;
#pragma omp parallel sections
{
  #pragma omp section
  { x = 1;
    #pragma omp flush(x)
  }
  #pragma omp section
    while (!x);
}
```

Синхронизация потоков

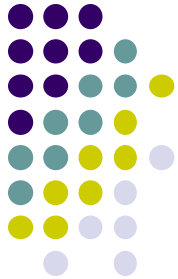


- Директивы синхронизации потоков:
  - master
  - barrier
  - critical
  - atomic
  - flush
  - **ordered**

Выделение упорядоченного блока в цикле

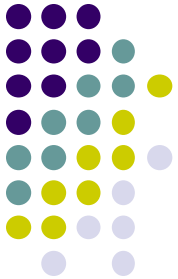
```
int i,j,k;
#pragma omp parallel for ordered
  for (i=0;i<N;i++)
  { printf("No order: %d\n",i);
    #pragma omp ordered
    printf("Order: %d\n",i);
  }
```

Синхронизация потоков



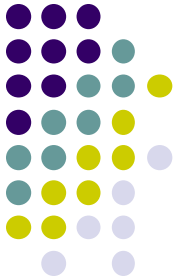
- Использование блокировок
  - Блокировка – особый объект, общий для потоков
  - Потоки могут захватывать (lock) и освобождать (unlock) блокировку
  - Только один поток в одно время может захватить блокировку
  - При попытке захвата блокировки потоки ждут её освобождения
- С помощью блокировок можно контролировать доступ к общим ресурсам

Пример: Использование блокировок



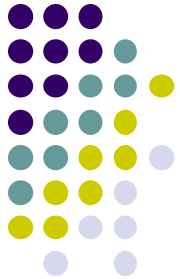
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int x[1000];
int main()
{ int i,max;
  omp_lock_t lock;
  omp_init_lock(&lock);
  for (i=0;i<1000;i++) x[i]=rand();
  max = x[0];
  #pragma omp parallel for
    for(i=0;i<1000;i++)
    { omp_set_lock(&lock);
      if (x[i]>max) max = x[i];
      omp_set_unlock(&lock);
    }
  omp_destroy_lock(&lock);
  return 0;
}
```

Синхронизация потоков



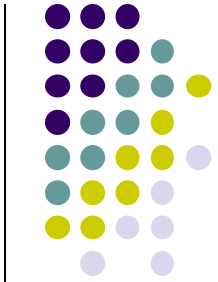
- Функции работы с блокировками
  - omp_lock_t – однократная блокировка
    - `void omp_init_lock(omp_lock_t *lock)`
    - `void omp_destroy_lock(omp_lock_t *lock)`
    - `void omp_set_lock(omp_lock_t *lock)`
    - `void omp_unset_lock(omp_lock_t *lock)`
    - `int omp_test_lock(omp_lock_t *lock)`
  - omp_nest_lock_t – многократная (вложенная) блокировка
    - `void omp_init_nest_lock(omp_nest_lock_t *lock)`
    - `void omp_destroy_nest_lock(omp_nest_lock_t *lock)`
    - `void omp_set_nest_lock(omp_nest_lock_t *lock)`
    - `void omp_unset_nest_lock(omp_nest_lock_t *lock)`
    - `int omp_test_nest_lock(omp_nest_lock_t *lock)`

Функции OpenMP



- `void omp_set_num_threads(int num_threads)`
- `int omp_get_num_threads()`
- `int omp_get_max_threads()`
- `int omp_get_thread_num()`
- `int omp_get_num_procs()`
- `int omp_in_parallel()`
- `void omp_set_dynamic(int dynamic_threads)`
- `int omp_get_dynamic()`
- `void omp_set_nested(int nested)`
- `int omp_get_nested ()`
- `double omp_get_wtime()` – текущее время потока, с
- `double omp_get_wtick()` – время между тактами таймера, с
- …

Новое в стандарте OpenMP 3.0



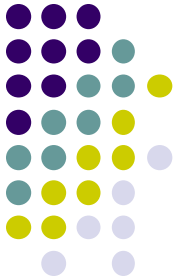
- Параллелизм задач

```
void traverse_list(List l)
{ Element e;
  for (e = l->first; e; e = e->next)
  #pragma omp task
    process(e);

  #pragma omp taskwait
}

. . .
#pragma omp parallel
#pragma omp single
traverse_list(some_list);
. . .
```

Новое в стандарте OpenMP 4.0



- Векторизация

```
#pragma omp simd
```

- Поддержка использования ускорителей

```
#pragma omp target device(acc)
```

- Привязка потоков к ядрам

```
#pragma omp parallel proc_bind(spread)
```

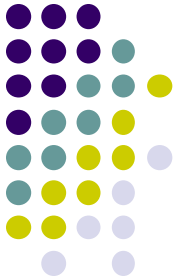
- Группы задач
  - `#pragma omp taskgroup`

- Определяемые пользователем редукции

```
#pragma declare reduction \
    (plus : int,char : omp_out+=omp_in) \
    initializer(omp_priv = 0)
```

- …

Литература по OpenMP



- http://openmp.org
- https://computing.llnl.gov/tutorials/openMP/
- …