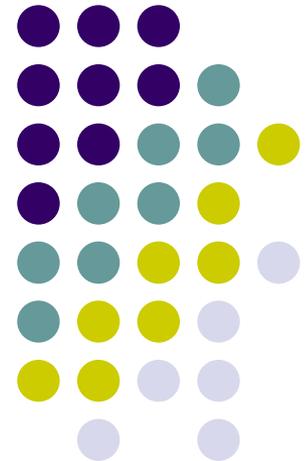
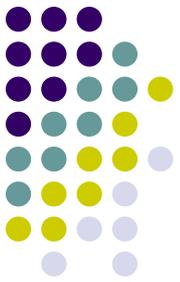


# Многопоточное программирование в OpenMP

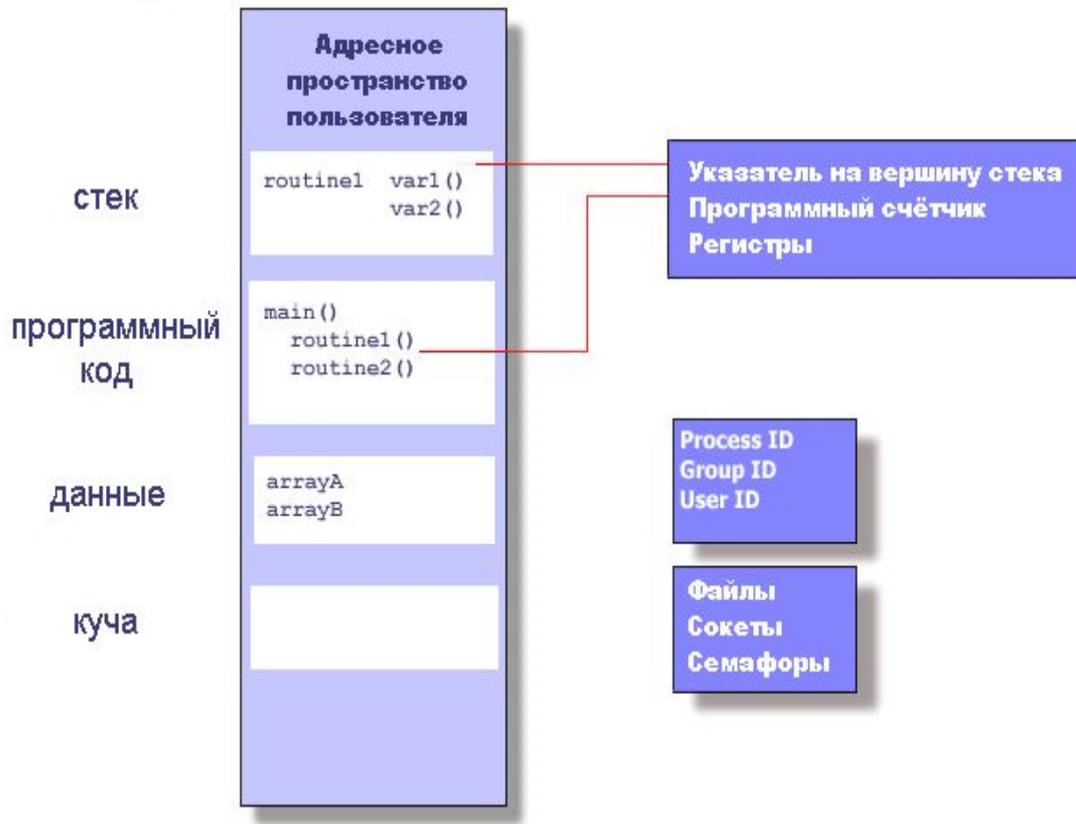
Киреев Сергей  
ИВМиМГ



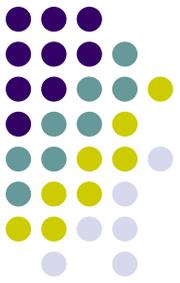


# Процессы и потоки

Процесс – это среда выполнения задачи (программы).

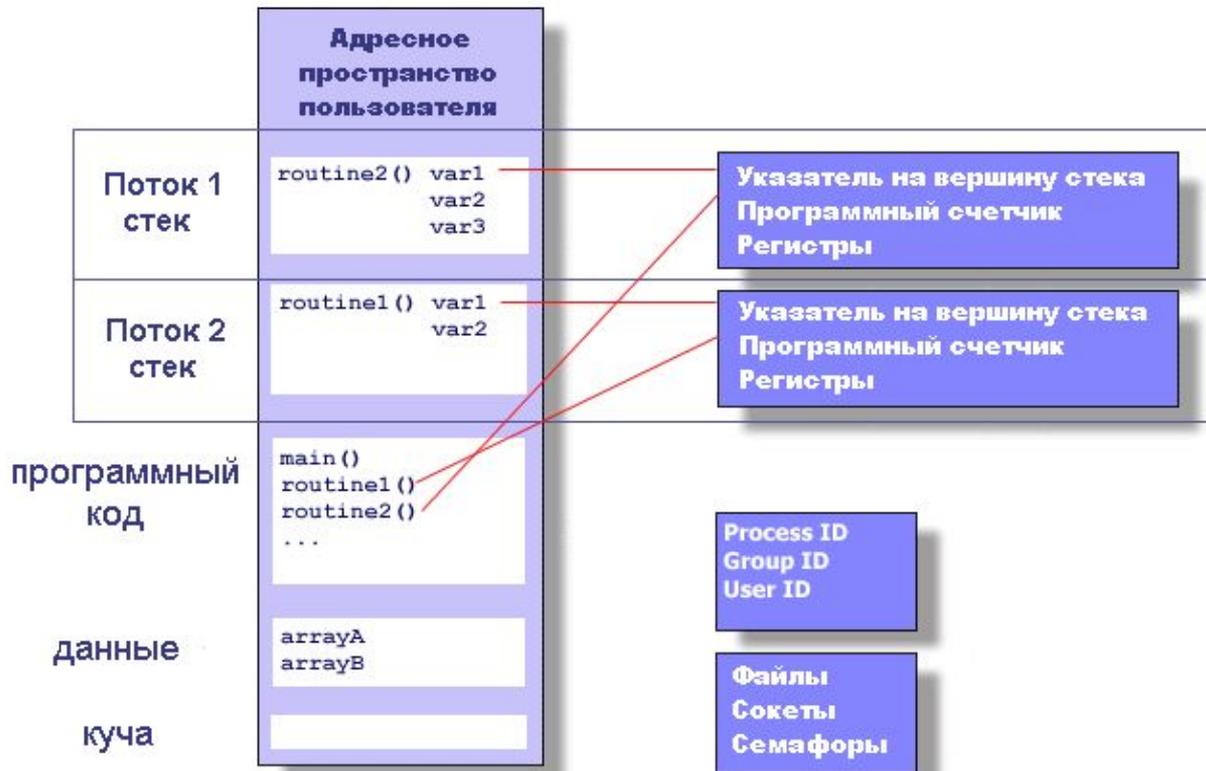


Процесс создаётся ОС и содержит информацию о программных ресурсах и текущем состоянии выполнения программы.



# Процессы и потоки

Поток – это «облегченный процесс».

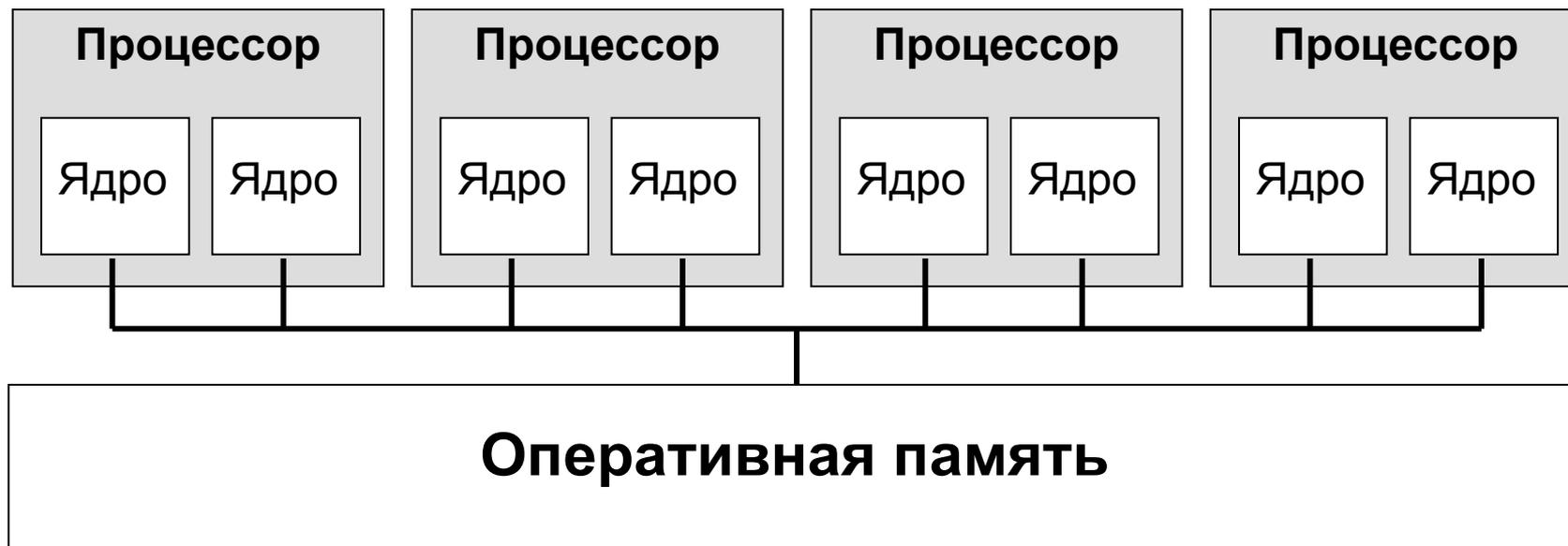


- Создается в рамках процесса,
- Имеет свой поток управления,
- Разделяет ресурсы процесса-родителя с другими потоками,
- Погибает, если погибает родительский процесс.

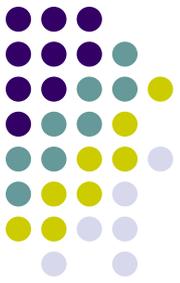
# Многопоточное программирование



- Используется для создания параллельных программ для систем с общей памятью

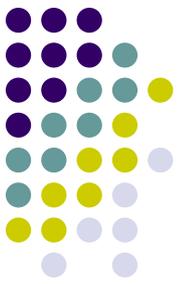


- И для других целей...



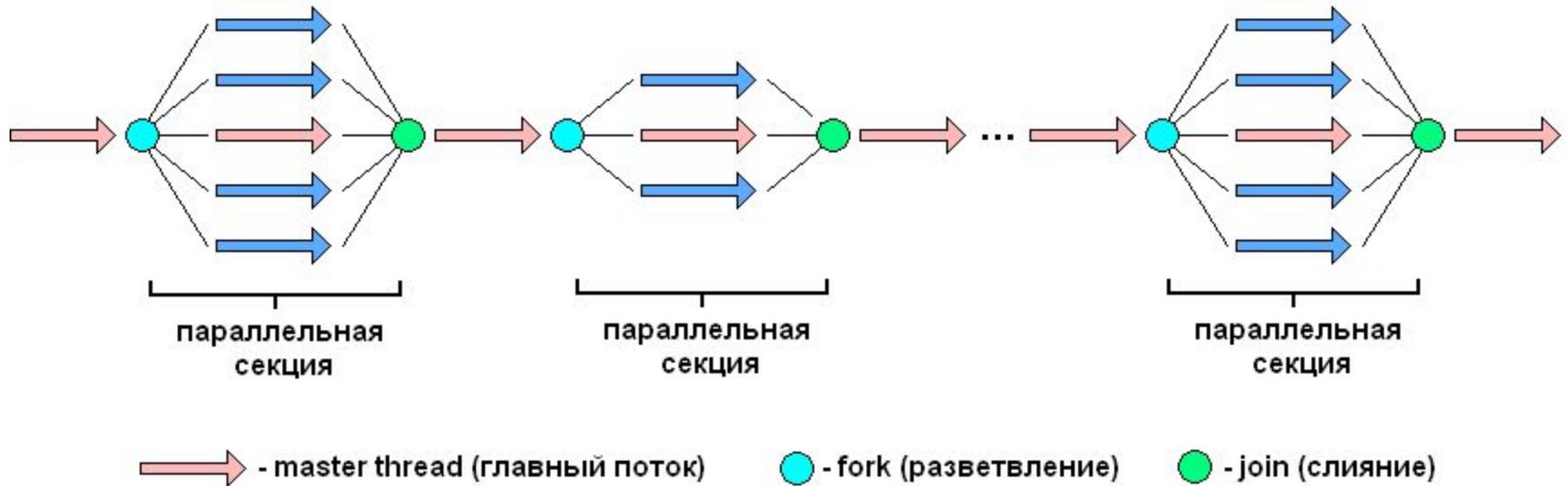
# OpenMP – это...

- 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`



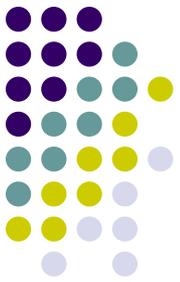
# Модель программирования

- Fork-join параллелизм



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

# Пример: Объявление параллельной секции

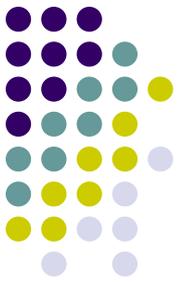


```
#include <omp.h>
int main()
{
    // последовательный код
    #pragma omp parallel
    {
        // параллельный код
    }
    // последовательный код

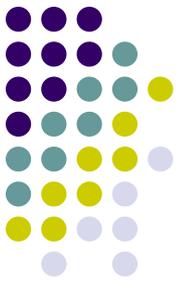
    return 0;
}
```

# 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;
}
```



# Задание числа потоков

- Переменная окружения OMP\_NUM\_THREADS

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

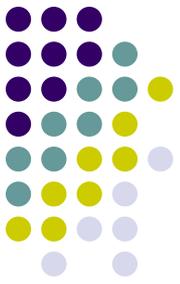
- Функция `omp_set_num_threads(int)`

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

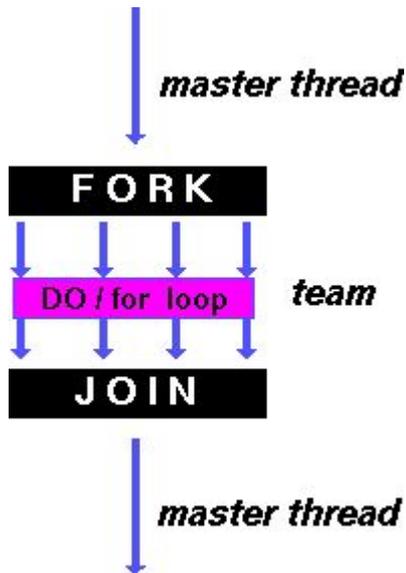
- Параметр `num_threads`

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

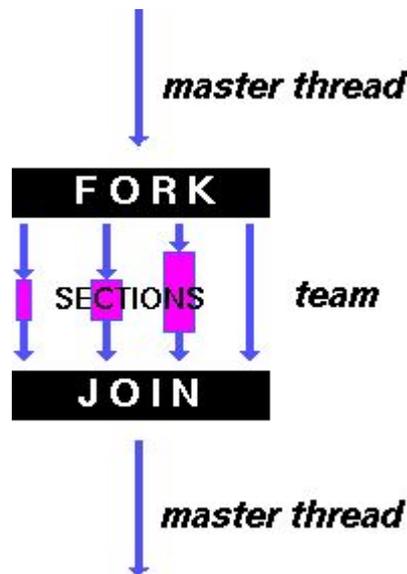
# Способы разделения работы между потоками



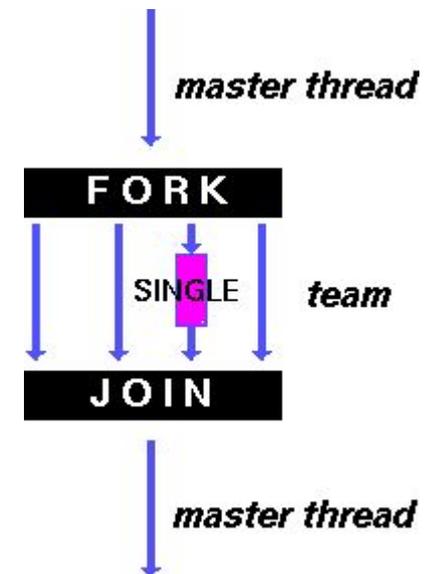
```
#pragma omp for
for (i=0;i<N;i++)
{
  // code
}
```

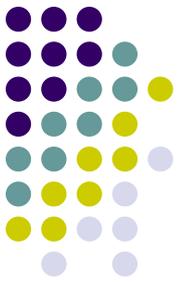


```
#pragma omp sections
{
  #pragma omp section
  // code 1
  #pragma omp section
  // code 2
}
```



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

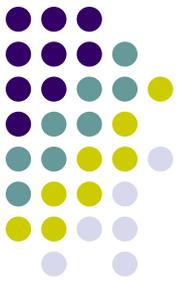




## Пример:

### 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`

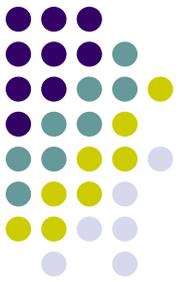
```
#include <stdio.h>
#include <omp.h>
int main()
{ int i;

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

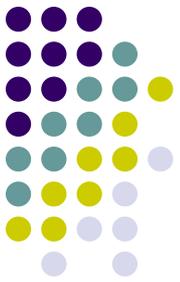
  return 0;
}
```

# Пример:

## Директива omp sections



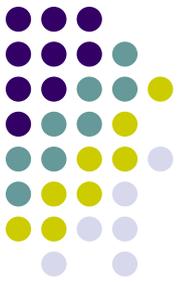
```
#include <stdio.h>
#include <omp.h>
int main()
{ int i;
  #pragma omp parallel sections private(i)
  {
    #pragma omp section
    printf("1st half\n");
    for (i=0;i<500;i++) printf("%d ");
    #pragma omp section
    printf("2nd half\n");
    for (i=501;i<1000;i++) printf("%d ");
  }
  return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

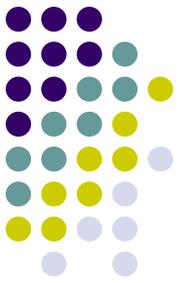


# Пример:

## 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 ");
    #pragma omp master
      printf("I'm Master!\n");
    #pragma omp for
      for (i=0;i<1000;i++) printf("%d ");
  }
  return 0;
}
```

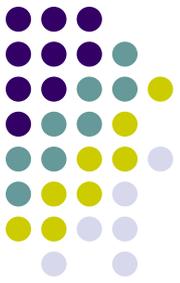
# Способы разделения работы между потоками



- 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
  - Параметры области видимости переменных...

# Пример:

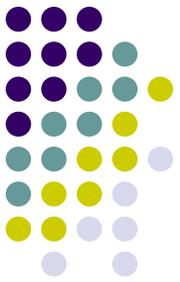
## Директива omp for



```
#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;
}
```

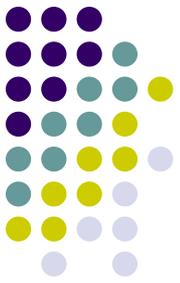
# Области видимости переменных



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

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

# Области видимости переменных



- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - lastprivate
  - shared
  - default
  - reduction
  - threadprivate
  - copying

# Области видимости переменных

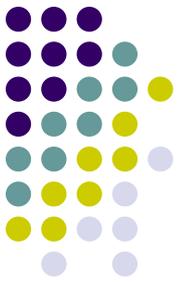


- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - **private**
  - firstprivate
  - lastprivate
  - shared
  - default
  - reduction
  - threadprivate
  - copying

**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

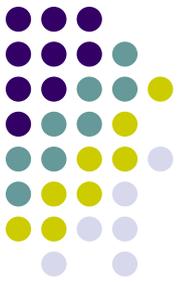


- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - **firstprivate**
  - lastprivate
  - shared
  - default
  - reduction
  - threadprivate
  - copying

Локальная переменная с инициализацией

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

# Области видимости переменных



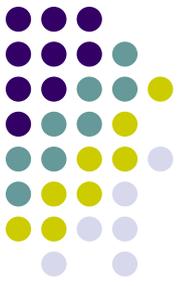
- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - **lastprivate**
  - shared
  - default
  - reduction
  - threadprivate
  - copying

**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



- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - lastprivate
  - **shared**
  - default
  - reduction
  - threadprivate
  - copying

## Разделяемая (общая) переменная

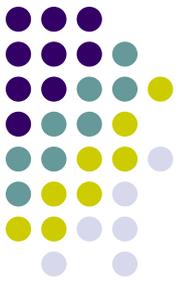
```
int i, j;
```

```
#pragma omp parallel for \  
                shared(j)
```

```
for (i=0; i<100; i++) j=i;
```

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

# Области видимости переменных

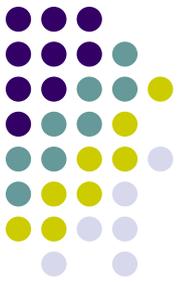


- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - lastprivate
  - shared
  - **default**
  - reduction
  - threadprivate
  - copying

**Задание области видимости не указанных явно переменных**

```
int i,k,n=2;
#pragma omp parallel shared(n) \
    default(private)
{
    i = omp_get_thread_num() / n;
    k = omp_get_thread_num() % n;
    printf("%d %d %d\n", i, k, n);
}
```

# Области видимости переменных



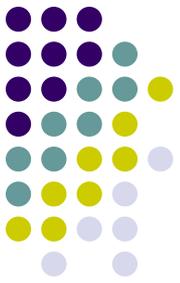
- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - lastprivate
  - shared
  - default
  - **reduction**
  - threadprivate
  - copying

**Переменная для выполнения редуционной 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

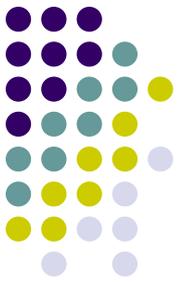


- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - lastprivate
  - shared
  - default
  - reduction
  - **threadprivate**
  - copying

**Объявление глобальных переменных локальными для потоков**

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

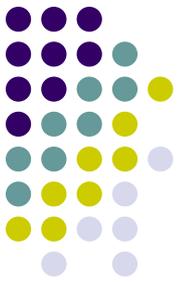
# Области видимости переменных



- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - lastprivate
  - shared
  - default
  - reduction
  - threadprivate
  - **copying**

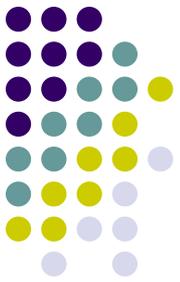
**Объявление глобальных переменных локальными для потоков с инициализацией**

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



# Синхронизация потоков

- Директивы синхронизации потоков:
  - master
  - critical
  - barrier
  - atomic
  - flush
  - ordered
- Блокировки
  - omp\_lock\_t

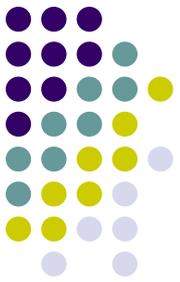


# Синхронизация потоков

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

**Выполнение кода только главным потоком**

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

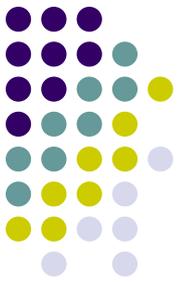


# Синхронизация потоков

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

## Критическая секция

```
int x;  
x = 0;  
  
#pragma omp parallel  
{  
    #pragma omp critical  
        x = x + 1;  
}
```

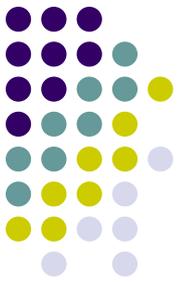


# Синхронизация потоков

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

## Барьер

```
int i;  
  
#pragma omp parallel for  
for (i=0;i<1000;i++)  
{  
    printf("%d ",i);  
    #pragma omp barrier  
}
```



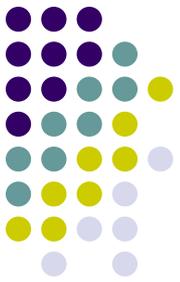
# Синхронизация потоков

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

## Атомарная операция

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

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



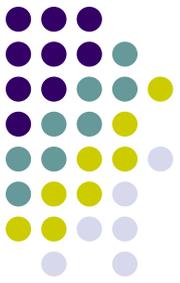
# Синхронизация потоков

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

**Согласование значения переменных  
между потоками**

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

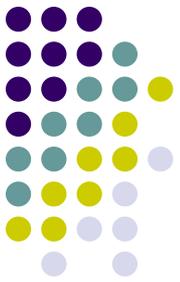
# Синхронизация потоков



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

## Выделение упорядоченного блока в цикле

```
int i,j,k;
double x;
#pragma omp parallel for ordered
for (i=0;i<N;i++)
{
    k = rand(); x = 1.0;
    for (j=0;j<k;j++) x=sin(x);
    printf("No order: %d\n",i);
    #pragma omp ordered
    printf("Order: %d\n",i);
}
```



# Синхронизация потоков

- Блокировки

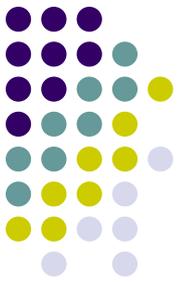
- `omp_lock_t`

- `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)`

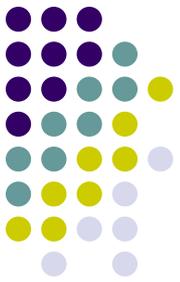
- `omp_nest_lock_t`

- `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)`

# Пример: Использование блокировок



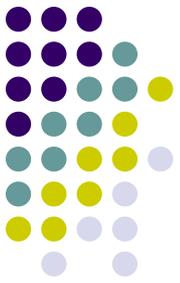
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int x[1000];
int main()
{ int i,max;
  omp_lock_t lock;
  omp_init_lock(&lock);
  for (i=0;i<1000;i++) x[i]=rand();
  max = x[0];
  #pragma omp parallel for shared(x,lock)
    for(i=0;i<1000;i++)
      { omp_set_lock(&lock);
        if (x[i]>max)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



# Функции OpenMP

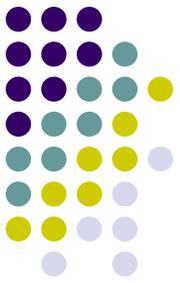
- `void omp_set_num_threads(int num_threads)`
- `int omp_get_num_threads(void)`
- `int omp_get_max_threads(void)`
- `int omp_get_thread_num(void)`
- `int omp_get_num_procs(void)`
- `int omp_in_parallel(void)`
- `void omp_set_dynamic(int dynamic_threads)`
- `int omp_get_dynamic(void)`
- `void omp_set_nested(int nested)`
- `int omp_get_nested (void)`
- `double omp_get_wtick(void)`
- Функции работы с блокировками

# Порядок создания параллельных программ



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

# Пример программы: сложение двух векторов



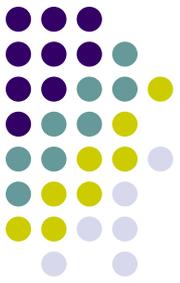
## Последовательная программа

```
#define N 1000
double x[N], y[N], z[N];
int main()
{ int i;

  for (i=0; i<N; i++) x[i]=y[i]=i;

  for (i=0; i<N; i++)
    z[i]=x[i]+y[i];
  return 0;
}
```

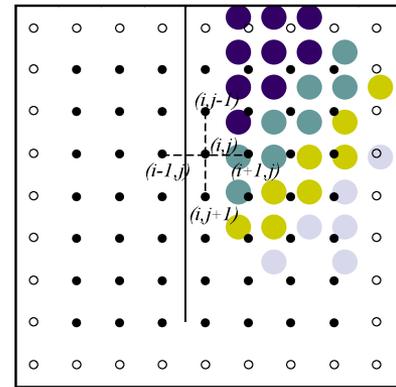
# Пример программы: сложение двух векторов



## Параллельная программа

```
#include<omp.h>
#define N 1000
double x[N],y[N],z[N];
int main()
{ int i;
  int num;
  for (i=0;i<N;i++) x[i]=y[i]=i;
  num = omp_get_num_threads();
  #pragma omp parallel for schedule(static,N/num)
  for (i=0;i<N;i++)
    z[i]=x[i]+y[i];
  return 0;
}
```

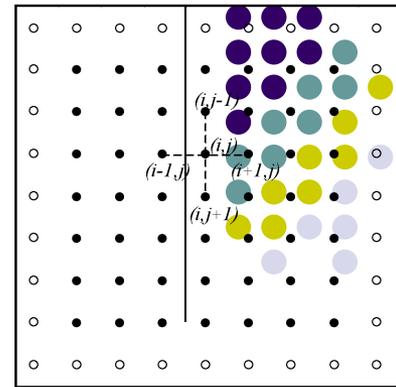
# Пример программы: решение краевой задачи



## Метод Зейделя

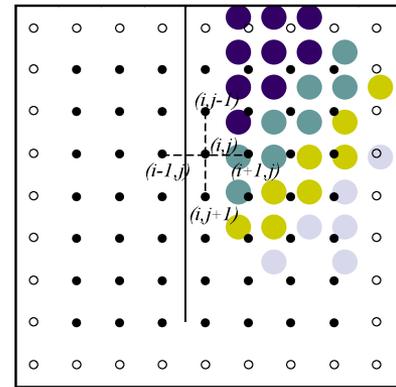
```
do {  
    dmax = 0; // максимальное изменение значений u  
    for ( i=1; i<N+1; i++ )  
        for ( j=1; j<N+1; j++ ) {  
            temp = u[i][j];  
            u[i][j] = 0.25*(u[i-1][j]+u[i+1][j]+  
                u[i][j-1]+u[i][j+1]-h*h*f[i][j]);  
            dm = fabs(temp-u[i][j]);  
            if ( dmax < dm ) dmax = dm;  
        }  
} while ( dmax > eps );
```

# Пример программы: решение краевой задачи



```
omp_lock_t dmax_lock;
omp_init_lock (&dmax_lock);
do {
    dmax = 0; // максимальное изменение значений u
    #pragma omp parallel for shared(u,n,dmax) private(i,temp,d)
    for ( i=1; i<N+1; i++ ) {
        #pragma omp parallel for shared(u,n,dmax) private(j,temp,d)
        for ( j=1; j<N+1; j++ ) {
            temp = u[i][j];
            u[i][j] = 0.25*(u[i-1][j]+u[i+1][j]+
                u[i][j-1]+u[i][j+1]-h*h*f[i][j]);
            d = fabs(temp-u[i][j]);
            omp_set_lock(&dmax_lock);
            if ( dmax < d ) dmax = d;
            omp_unset_lock(&dmax_lock);
        } // конец вложенной параллельной области
    } // конец внешней параллельной области
} while ( dmax > eps );
```

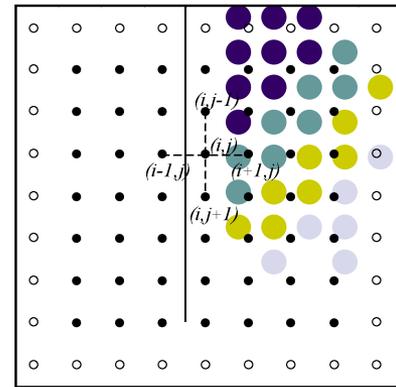
# Пример программы: решение краевой задачи



```
omp_lock_t dmax_lock;  
omp_init_lock (&dmax_lock);  
do {  
    dmax = 0; // максимальное изменение значений u  
    #pragma omp parallel for shared(u,n,dmax) private(i,temp,d)  
    for ( i=1; i<N+1; i++ ) {  
        #pragma omp parallel for shared(u,n,dmax) private(j,temp,d)  
        for ( j=1; j<N+1; j++ ) {  
            temp = u[i][j];  
            u[i][j] = 0.25*(u[i-1][j]+u[i+1][j]+  
                u[i][j-1]+u[i][j+1]-h*h*f[i][j]);  
            d = fabs(temp-u[i][j])  
            omp_set_lock(&dmax_lock);  
            if ( dmax < d ) dmax = d;  
            omp_unset_lock(&dmax_lock);  
        } // конец вложенной параллельной области  
    } // конец внешней параллельной области  
} while ( dmax > eps );
```

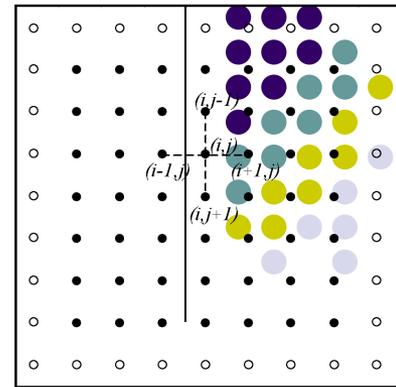
Синхронизация –  
узкое место

# Пример программы: решение краевой задачи



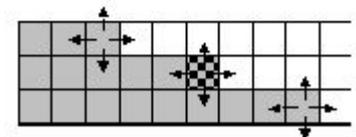
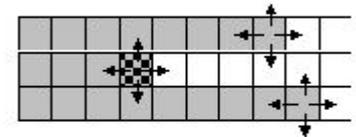
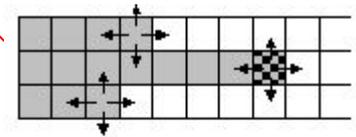
```
omp_lock_t dmax_lock;
omp_init_lock(&dmax_lock);
do {
    dmax = 0; // максимальное изменение значений u
    #pragma omp parallel for shared(u,n,dmax) private(i,temp,d,dm)
    for ( i=1; i<N+1; i++ ) {
        dm = 0;
        for ( j=1; j<N+1; j++ ) {
            temp = u[i][j];
            u[i][j] = 0.25*(u[i-1][j]+u[i+1][j]+
                u[i][j-1]+u[i][j+1]-h*h*f[i][j]);
            d = fabs(temp-u[i][j])
            if ( dm < d ) dm = d;
        }
        omp_set_lock(&dmax_lock);
        if ( dmax < dm ) dmax = dm;
        omp_unset_lock(&dmax_lock);
    }
    // конец параллельной области
} while ( dmax > eps );
```

# Пример программы: решение краевой задачи

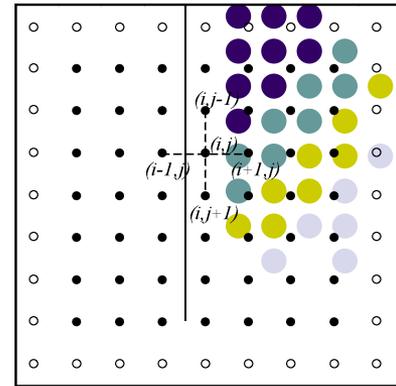


```
omp_lock_t dmax_lock;
omp_init_lock(&dmax_lock);
do {
    dmax = 0; // максимальное изменение значений u
    #pragma omp parallel for shared(u,n,dmax) private(i,temp,d,dm)
    for ( i=1; i<N+1; i++ ) {
        dm = 0;
        for ( j=1; j<N+1; j++ ) {
            temp = u[i][j];
            u[i][j] = 0.25*(u[i-1][j]+u[i+1][j]+
                u[i][j-1]+u[i][j+1]-h*h*f[i][j]);
            d = fabs(temp-u[i][j])
            if ( dm < d ) dm = d;
        }
        omp_set_lock(&dmax_lock);
        if ( dmax < dm ) dmax = dm;
        omp_unset_lock(&dmax_lock);
    }
    // конец параллельной области
} while ( dmax > eps );
```

## Неоднозначность вычислений

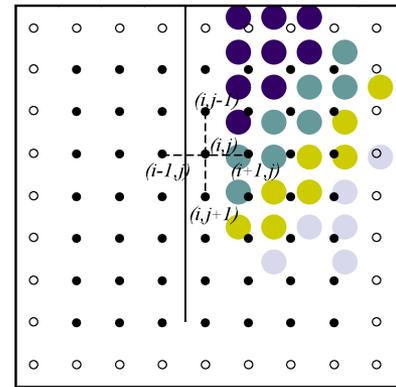


# Пример программы: решение краевой задачи



```
omp_lock_t dmax_lock;
omp_init_lock(&dmax_lock);
do {
    dmax = 0; // максимальное изменение значений u
    #pragma omp parallel for shared(u,n,dmax) private(i,temp,d,dm)
    for ( i=1; i<N+1; i++ ) {
        dm = 0;
        for ( j=1; j<N+1; j++ ) {
            temp = u[i][j];
            un[i][j] = 0.25*(u[i-1][j]+u[i+1][j]+
                u[i][j-1]+u[i][j+1]-h*h*f[i][j]);
            d = fabs(temp-un[i][j])
            if ( dm < d ) dm = d;
        }
        omp_set_lock(&dmax_lock);
        if ( dmax < dm ) dmax = dm;
        omp_unset_lock(&dmax_lock);
    }
    // конец параллельной области
    for ( i=1; i<N+1; i++ ) // обновление данных
        for ( j=1; j<N+1; j++ )
            u[i][j] = un[i][j];
} while ( dmax > eps );
```

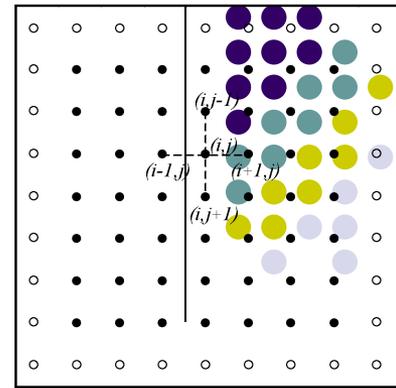
# Пример программы: решение краевой задачи



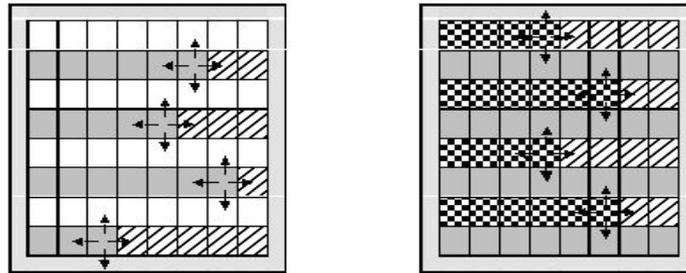
```
omp_lock_t dmax_lock;
omp_init_lock(&dmax_lock);
do {
    dmax = 0; // максимальное изменение значений u
    #pragma omp parallel for shared(u,n,dmax) private(i,temp,d,dm)
    for ( i=1; i<N+1; i++ ) {
        dm = 0;
        for ( j=1; j<N+1; j++ ) {
            temp = u[i][j];
            un[i][j] = 0.25*(u[i-1][j]+u[i+1][j]+
                u[i][j-1]+u[i][j+1]-h*h*f[i][j]);
            d = fabs(temp-un[i][j]);
            if ( dm < d ) dm = d;
        }
        omp_set_lock(&dmax_lock);
        if ( dmax < dm ) dmax = dm;
        omp_unset_lock(&dmax_lock);
    } // конец параллельной области
    for ( i=1; i<N+1; i++ ) // обновление данных
        for ( j=1; j<N+1; j++ )
            u[i][j] = un[i][j];
} while ( dmax > eps );
```

Получили  
метод  
Якоби

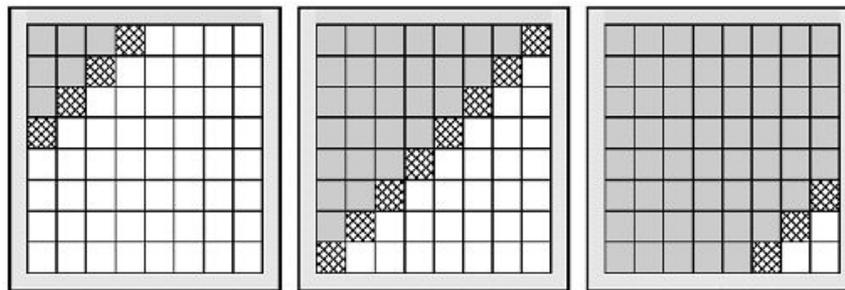
# Пример программы: решение краевой задачи



- Другие способы устранения зависимостей
  - Четно-нечетное упорядочивание

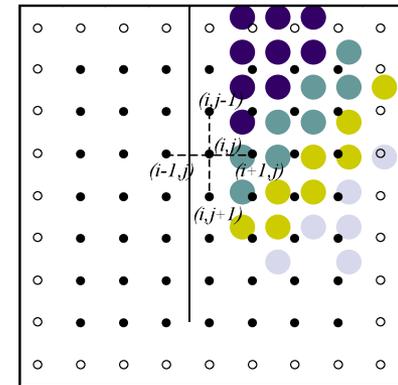


- Волновые схемы



# Пример программы: решение краевой задачи

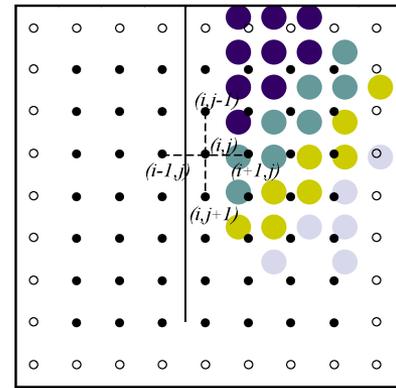
## Метод Зейделя: волновая схема



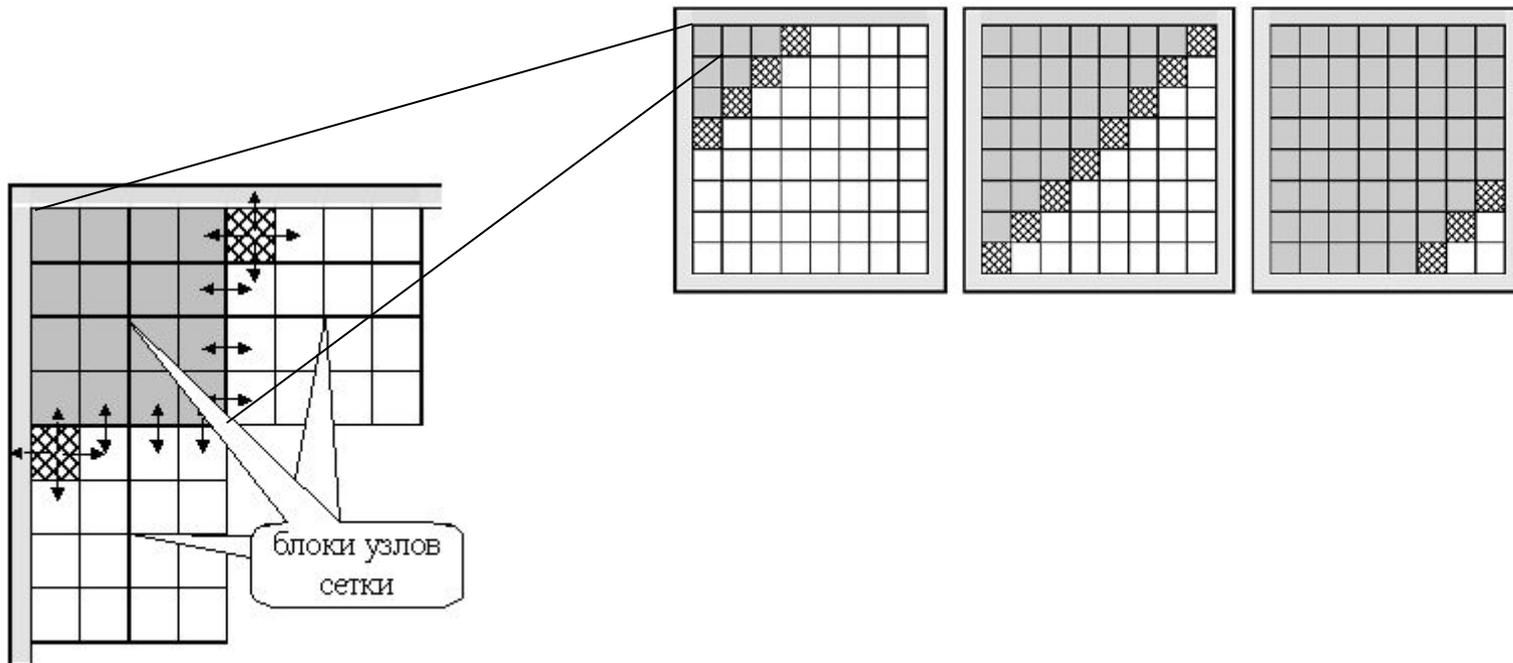
```
omp_lock_t dmax_lock;
omp_init_lock(&dmax_lock);
do {
    // максимальное изменение значений u
    dmax = 0;
    // нарастание волны (nx – размер волны)
    for ( nx=1; nx<N+1; nx++ ) {
        dm[nx] = 0;
        #pragma omp parallel for \
        shared(u, nx, dm)          \
        private(i, j, temp, d)
        for ( i=1; i<nx+1; i++ ) {
            j      = nx + 1 - i;
            temp = u[i][j];
            u[i][j] = 0.25*(u[i-1][j]+
                u[i+1][j]+u[i][j-1]+u[i][j+1]
                -h*h*f[i][j]);
            d = fabs(temp-u[i][j]);
            if ( dm[i] < d ) dm[i] = d;
        } // конец параллельной области
    }
}
```

```
// затухание волны
for ( nx=N-1; nx>0; nx-- ) {
    #pragma omp parallel for \
    shared(u, nx, dm) private(i, j, temp, d)
    for ( i=N-nx+1; i<N+1; i++ ) {
        j      = 2*N - nx - I + 1;
        temp = u[i][j];
        u[i][j] = 0.25*(u[i-1][j]+
            u[i+1][j]+u[i][j-1]+u[i][j+1]
            -h*h*f[i][j]);
        d = fabs(temp-u[i][j]);
        if ( dm[i] < d ) dm[i] = d;
    } // конец параллельной области
}
#pragma omp parallel for \
shared(n, dm, dmax) private(i)
for ( i=1; i<nx+1; i++ ) {
    omp_set_lock(&dmax_lock);
    if ( dmax < dm[i] ) dmax = dm[i];
    omp_unset_lock(&dmax_lock);
} // конец параллельной области
} while ( dmax > eps );
```

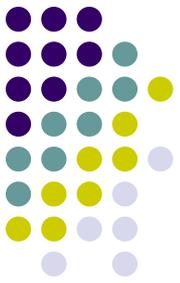
# Пример программы: решение краевой задачи



- Волновая схема с разбиением на блоки



# Рекомендуемая литература по OpenMP



- <http://openmp.org>
- [http://www.parallel.ru/tech/tech\\_dev/openmp.html](http://www.parallel.ru/tech/tech_dev/openmp.html)
- <https://computing.llnl.gov/tutorials/openMP/>