

Рабочие материалы по курсу
ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ

Раздел
**ВВЕДЕНИЕ В ТЕОРИЮ АНАЛИЗА СТРУКТУРЫ
ПРОГРАММ И АЛГОРИТМОВ**

Вл. В. Воеводин
заместитель директора НИВЦ МГУ,
член-корреспондент РАН,
voevodin@parallel.ru

Пользователь: почему?

$$A_{ijk} = A_{i-1jk} + B_{jk} + B_{jk}, \quad i=1,40; \quad j=1,40; \quad k=1,1000$$

Cray C90, пиковая производительность **960** Mflop/s

```
do k = 1, 1000
```

```
  do j = 1, 40
```

```
    do i = 1, 40
```

```
      A(i,j,k) = A(i-1,j,k)+B(j,k)+B(j,k)
```

Производительность: **20** Mflop/s на Cray C90

Пользователь: почему?

$$A_{ijk} = A_{i-1jk} + B_{jk} + B_{jk}, \quad i=1,40; \quad j=1,40; \quad k=1,1000$$

Cray C90, пиковая производительность **960** Mflop/s

```
do i = 1, 40, 2
```

```
  do j = 1, 40
```

```
    do k = 1, 1000
```

```
      A(i,j,k) = A(i-1,j,k)+2*B(j,k)
```

```
      A(i+1,j,k) = A(i,j,k)+2*B(j,k)
```

Производительность: **700** Mflop/s на Cray C90

Умножение матриц: все ли просто?

Фрагмент исходного текста:

```
for( i = 0; i < n; ++i)
```

```
    for( j = 0; j < n; ++j)
```

```
        for( k = 0; k < n; ++k)
```

```
            A[i][j] = A[i][j] + B[i][k]*C[k][j]
```

Возможен ли порядок:

(i, k, j) - ? **ДА**

(k, i, j) - ? **ДА**

(k, j, i) - ? **ДА**

(j, i, k) - ? **ДА**

(j, k, i) - ? **ДА**

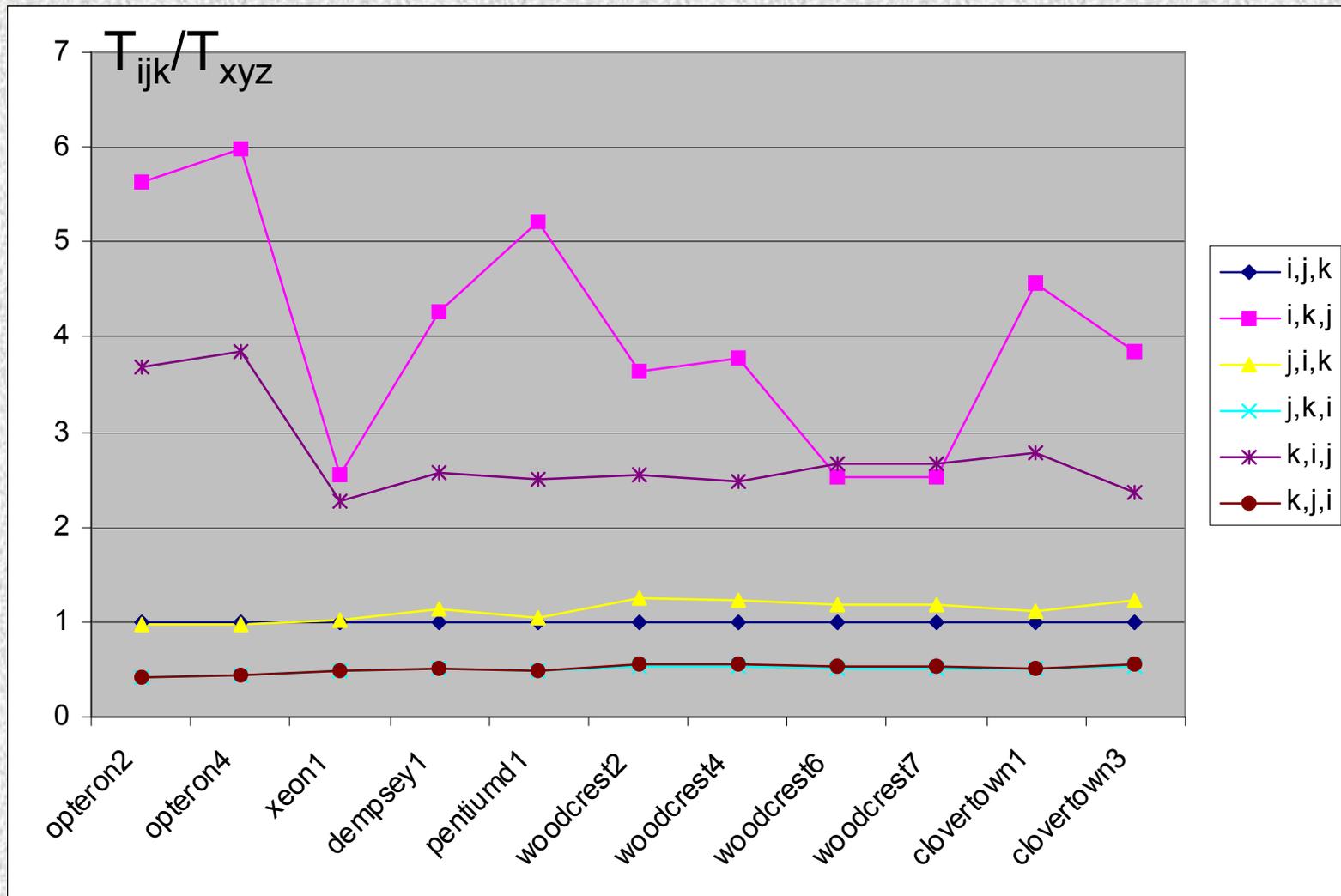
Порядок циклов: (i, j, k)

Почему возможен
другой порядок?

А зачем нужен
другой порядок?

Умножение матриц: все ли просто?

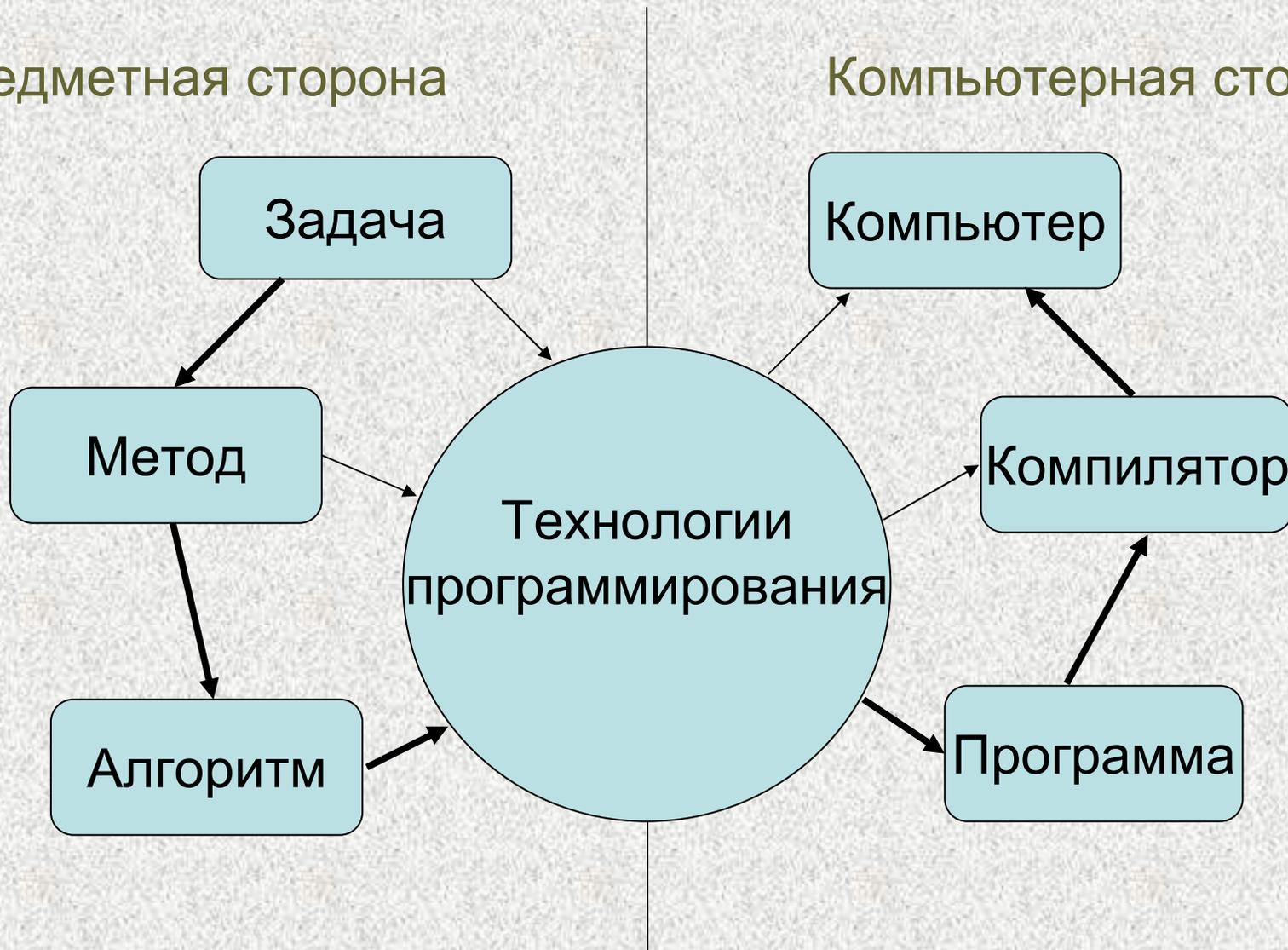
(сравнение с порядком (i, j, k))



Решение задачи на компьютере

Предметная сторона

Компьютерная сторона



Графовые модели программ

*Будем представлять программы с помощью графов:
набор вершин и множество соединяющих их
направленных дуг.*

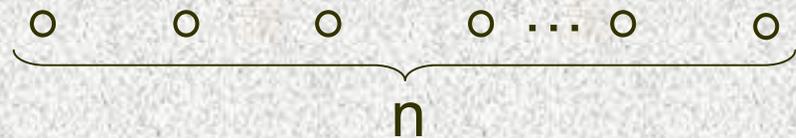
*Вершины: процедуры, циклы, линейные участки,
операторы, итерации циклов, срабатывания
операторов...*

Графовые модели программ

Будем представлять программы с помощью графов:
набор вершин и множество соединяющих их
направленных дуг.

Вершины: *итерации циклов*.

```
for( i = 0; i < n; ++i) {  
    A[i] = A[i - 1] + 2;  
    B[i] = B[i] + A[i];  
}
```



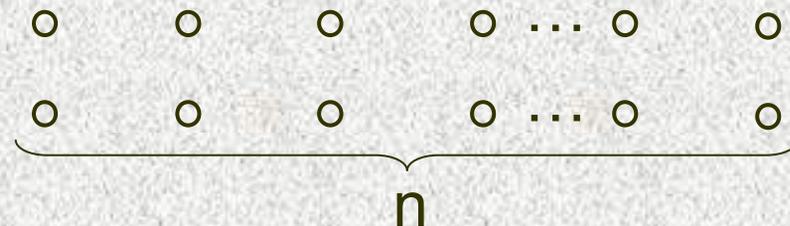
Каждая вершина соответствует
двум операторам (телу цикла),
выполненным на одной и той же
итерации цикла.

Графовые модели программ

Будем представлять программы с помощью графов:
набор вершин и множество соединяющих их
направленных дуг.

Вершины: **срабатывания операторов.**

```
for( i = 0; i < n; ++i) {  
    A[i] = A[i - 1] + 2;  
    B[i] = B[i] + A[i];  
}
```



Каждая вершина соответствует
одному из двух операторов тела
данного цикла, выполненному на
некоторой итерации.

Графовые модели программ

Будем представлять программы с помощью графов: набор вершин и множество соединяющих их направленных дуг.

Вершины: процедуры, циклы, линейные участки, операторы, итерации циклов, срабатывания операторов...

Дуги: отражают связь (отношение) между вершинами.

Выделяют два типа отношений:

- операционное отношение,
- информационное отношение.

Графовые модели программ

Будем представлять программы с помощью графов: набор вершин и множество соединяющих их направленных дуг.

Дуги: **операционное отношение**:



Две вершины **A** и **B** соединяются направленной дугой тогда и только тогда, когда вершина **B** может быть выполнена сразу после вершины **A**.

Операционное отношение = отношение по передаче управления.

Графовые модели программ

Будем представлять программы с помощью графов:
набор вершин и множество соединяющих их
направленных дуг.

Дуги: *операционное отношение*:

$$x(i) = a + b(i) \quad (1)$$

$$y(i) = 2 * x(i) - 3 \quad (2)$$

$$t1 = y(i) * y(i) + 1 \quad (3)$$

$$t2 = b(i) - y(i) * a \quad (4)$$



Графовые модели программ

Будем представлять программы с помощью графов: набор вершин и множество соединяющих их направленных дуг.

Дуги: *информационное отношение*:



Две вершины **A** и **B** соединяются направленной дугой тогда и только тогда, когда вершина **B** использует в качестве аргумента некоторое значение, полученное в вершине **A**.

Информационное отношение = отношение по передаче данных.

Графовые модели программ

Будем представлять программы с помощью графов: набор вершин и множество соединяющих их направленных дуг.

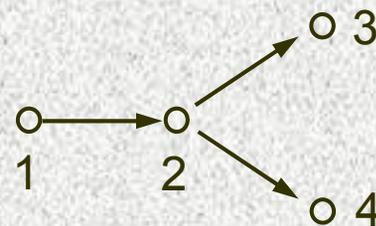
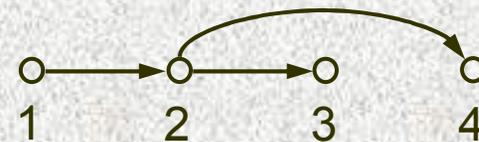
Дуги: *информационное отношение*:

$$x(i) = a + b(i) \quad (1)$$

$$y(i) = 2 * x(i) - 3 \quad (2)$$

$$t1 = y(i) * y(i) + 1 \quad (3)$$

$$t2 = b(i) - y(i) * a \quad (4)$$



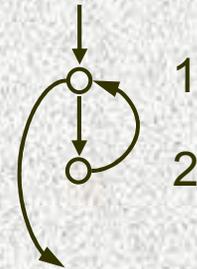
Четыре основные модели программ

Граф управления программы.

Вершины: операторы

Дуги: операционное отношение

```
for( i = 0; i < n; ++i) {  
    A[i] = A[i - 1] + 2;      (1)  
    B[i] = B[i] + A[i];      (2)  
}
```



Четыре основные модели программ

Информационный граф программы.

Вершины: операторы

Дуги: информационное отношение

```
for( i = 0; i < n; ++i) {  
    A[i] = A[i - 1] + 2;  
    B[i] = B[i] + A[i];  
}
```

(1)

(2)



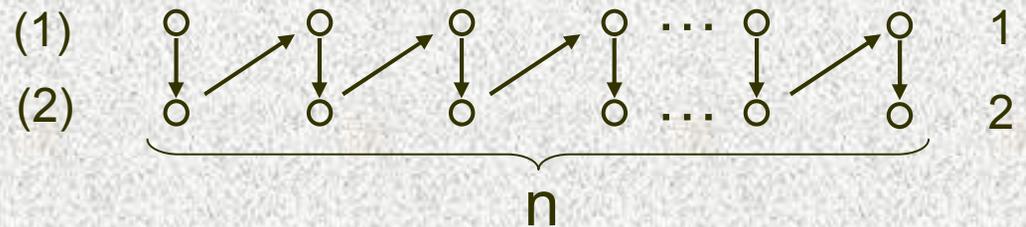
Четыре основные модели программ

Операционная история программы.

Вершины: срабатывания операторов

Дуги: операционное отношение

```
for( i = 0; i < n; ++i) {  
    A[i] = A[i - 1] + 2;  
    B[i] = B[i] + A[i];  
}
```



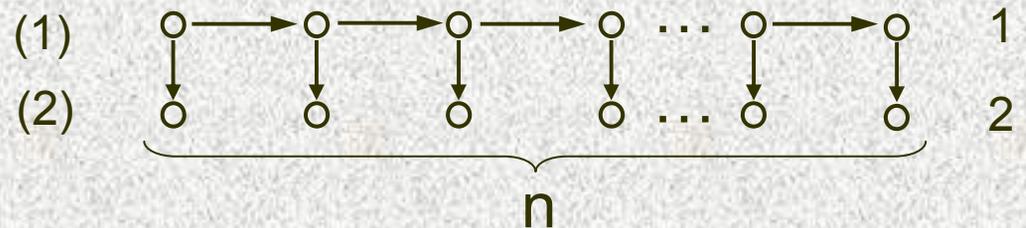
Четыре основные модели программ

Информационная история программы.

Вершины: срабатывания операторов

Дуги: информационное отношение

```
for( i = 0; i < n; ++i) {  
    A[i] = A[i - 1] + 2;  
    B[i] = B[i] + A[i];  
}
```

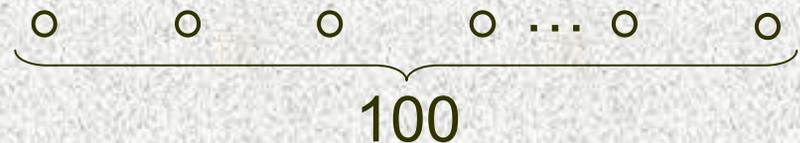


Несколько вопросов...

Может ли информационная история некоторого фрагмента программы иметь 100 вершин и ни одной дуги?

ДА.

```
for( i = 0; i < 100; ++i)  
  A[i] = B[i] + C[i]*x;
```



Несколько вопросов...

Может ли информационная история некоторого фрагмента программы иметь 67 вершин и 3 дуги?

ДА.

```
for( i = 0; i < 63; ++i)
  A[i] = B[i] + C[i]*x;
x1 = 10;
x2 = x1+1;
x3 = x2+2;
x4 = x3+3;
```



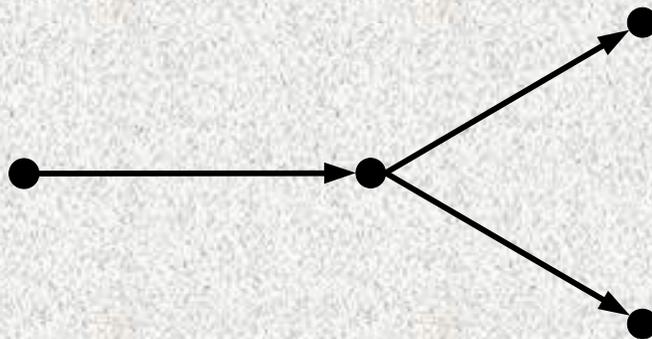
Несколько вопросов...

Может ли информационная история некоторого фрагмента программы иметь 20 вершин и 200 дуг?

НЕТ.

Несколько вопросов...

Модель некоторого фрагмента программы в качестве подграфа содержит следующий граф:



Какой моделью могла бы быть исходная модель?

ГУ

ИГ

~~ОИ~~

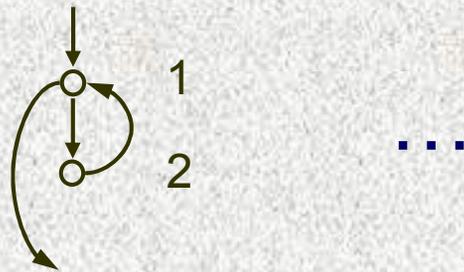
ИИ

Множество графовых моделей программ (опорные точки)

Операционные
модели

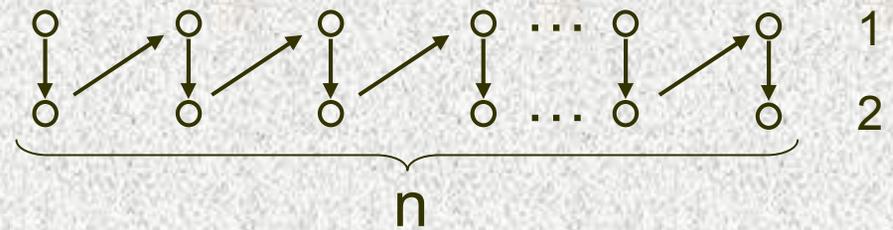
Компактные модели

Граф управления



Истории

Операционная история

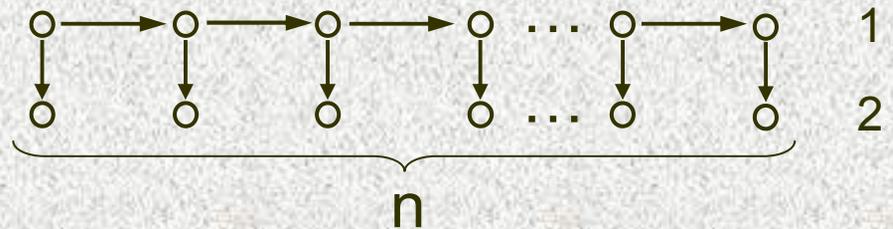


Информационные
модели

Информационный граф



Информационная история



Какое отношение выбрать для описания свойств программ?

Операционное отношение?

$$x(i) = a + b(i) \quad (1)$$

$$y(i) = 2 * x(i) - 3 \quad (2)$$

$$t1 = y(i) * y(i) + 1 \quad (3)$$

$$t2 = b(i) - y(i) * a \quad (4)$$



Какое отношение выбрать для описания свойств программ?

Информационная структура – это основа анализа свойств программ и алгоритмов.

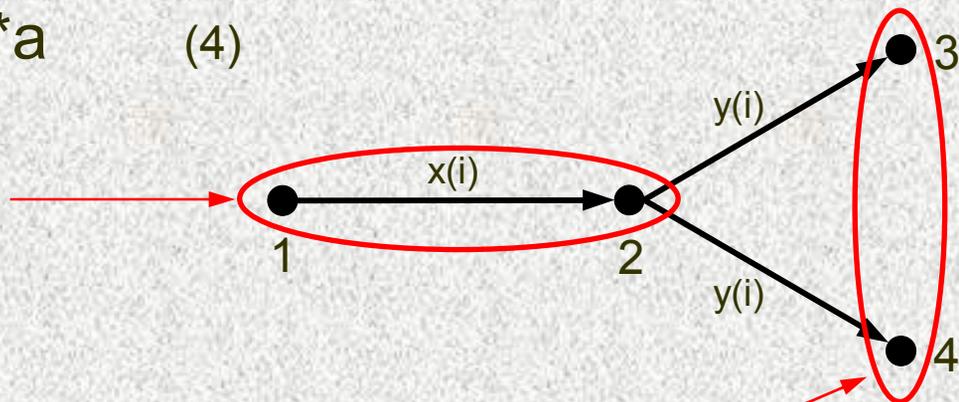
$$x(i) = a + b(i) \quad (1)$$

$$y(i) = 2 * x(i) - 3 \quad (2)$$

$$t1 = y(i) * y(i) + 1 \quad (3)$$

$$t2 = b(i) - y(i) * a \quad (4)$$

Исполнять только последовательно !

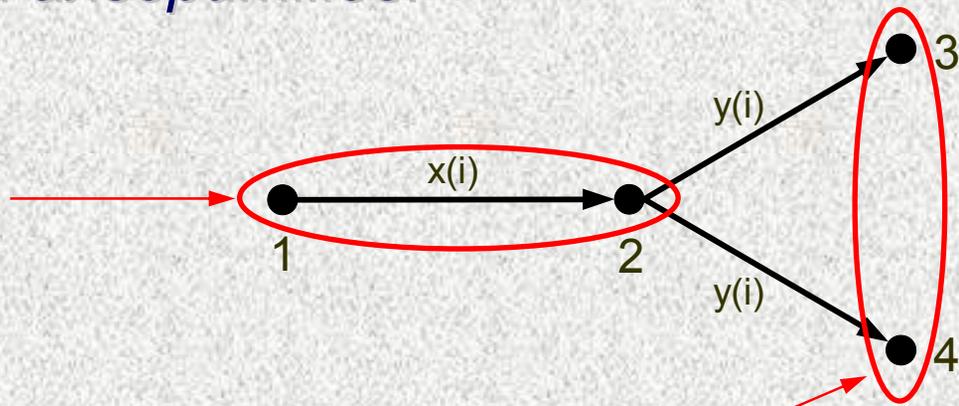


Можно исполнять параллельно !

Какое отношение выбрать для описания свойств программ?

Информационная структура – это основа анализа свойств программ и алгоритмов.

Исполнять только последовательно !



Можно исполнять параллельно !

Информационная зависимость определяет критерий эквивалентности преобразований программ.

Информационная независимость определяет ресурс параллелизма программы.

От компактных до историй: что выбрать для описания свойств программ?

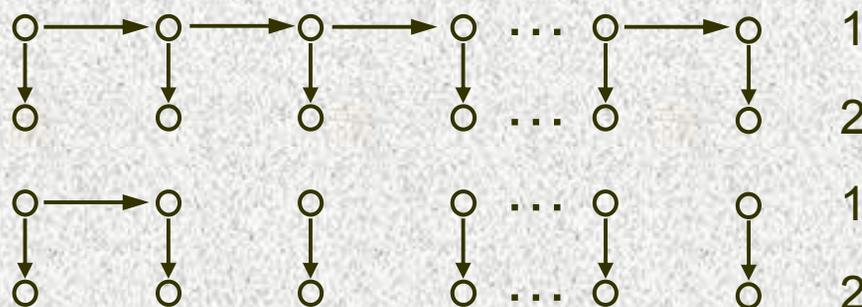
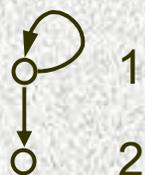
Аргументы для выбора степени компактности модели:

- компактность описания,*
- информативность,*
- сложность построения.*

От компактных до историй: что выбрать для описания свойств программ?

Аргументы для выбора степени компактности модели:

- компактность описания,
- информативность,



- сложность построения.

От компактных до историй: что выбрать для описания свойств программ?

Аргументы для выбора степени компактности модели:

- компактность описания, (компактные +)
- информативность, (истории +)
- сложность построения. (компактные +)

Граф алгоритма – это параметризованная информационная история:

- компактность описания за счет параметризации,
- имеет информативность истории,
- разработана методика построения графа алгоритма по исходному тексту программ.

Схема анализа и преобразования структуры программ

Исходная программа



Построение
графа алгоритма



Исследование
графа алгоритма



Преобразование
графа алгоритма



Преобразованная программа

Теорема о построении графа алгоритма

Теорема. Если фрагмент принадлежит к линейному классу программ, то на основе статического анализа можно построить компактное описание его графа алгоритма в следующем виде:

для каждого входа каждого оператора фрагмента будет указано конечное множество троек вида

$$(N, \Delta(N), F(\Delta, N))_k ,$$

где:

N – линейный выпуклый многогранник в пространстве внешних переменных фрагмента,

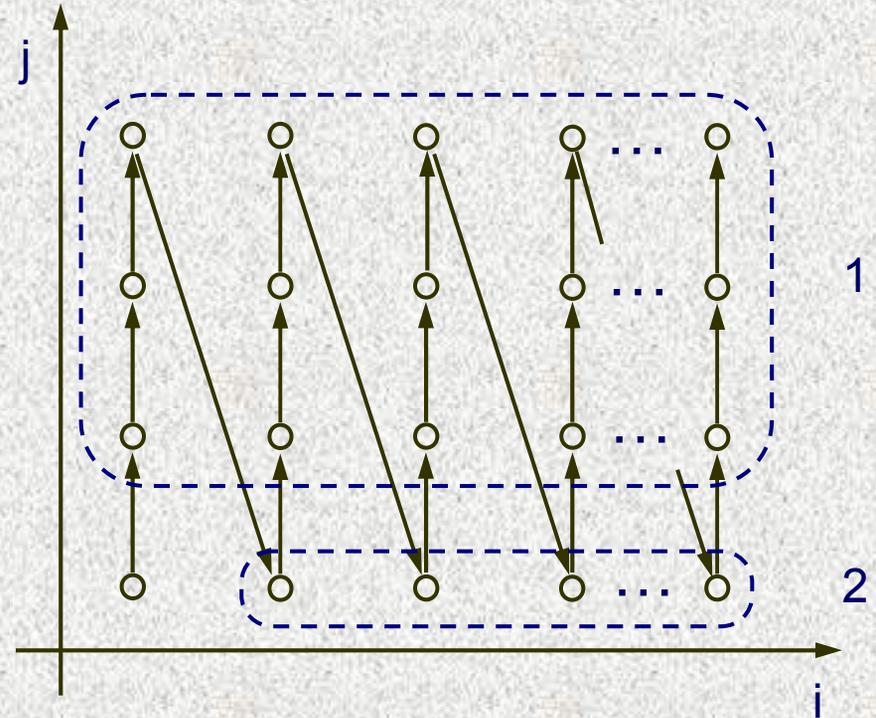
$\Delta(N)$ – линейный выпуклый многогранник в пространстве итераций фрагмента,

$F(\Delta, N)$ – линейная векторная функция, описывающая входящие дуги оператора.

Программы и их графы алгоритма

```

Do i = 1, n
  Do j = 1, m
    s = s + A(i, j)
  
```



Для входа s:

$$\begin{aligned}
 N_1 &= \begin{cases} n \geq 1 \\ m \geq 2 \end{cases} & I_1 &= \begin{cases} 1 \leq i \leq n \\ 2 \leq j \leq m \end{cases} & F_1 &= \begin{cases} i' = i \\ j' = j - 1 \end{cases} \\
 N_2 &= \begin{cases} n \geq 2 \\ m \geq 1 \end{cases} & I_2 &= \begin{cases} 2 \leq i \leq n \\ j = 1 \end{cases} & F_2 &= \begin{cases} i' = i - 1 \\ j' = m \end{cases}
 \end{aligned}$$

Программы и их графы алгоритма

$s = 0$ (1)

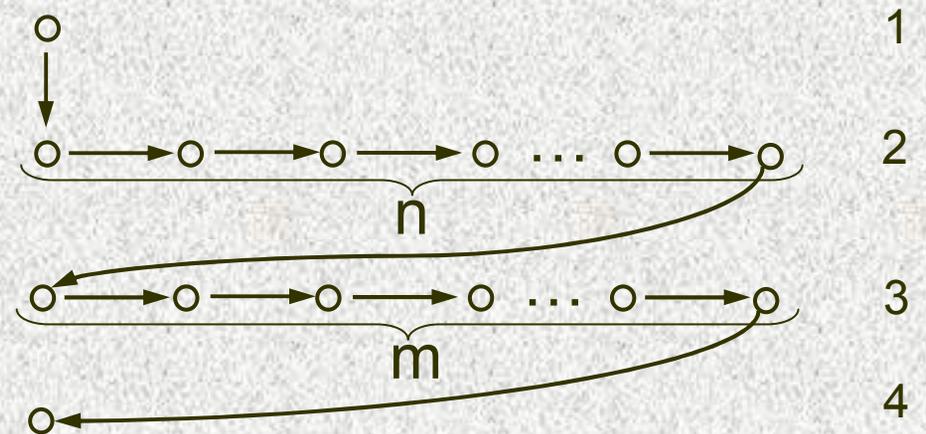
Do $i = 1, n$

$s = s + 1$ (2)

Do $i = 1, m$

$s = s + 1$ (3)

$s = s + 1$ (4)

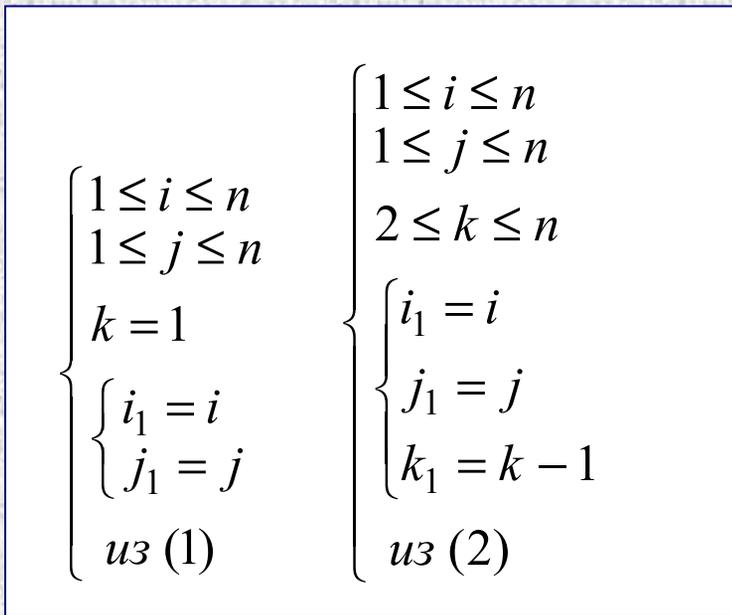
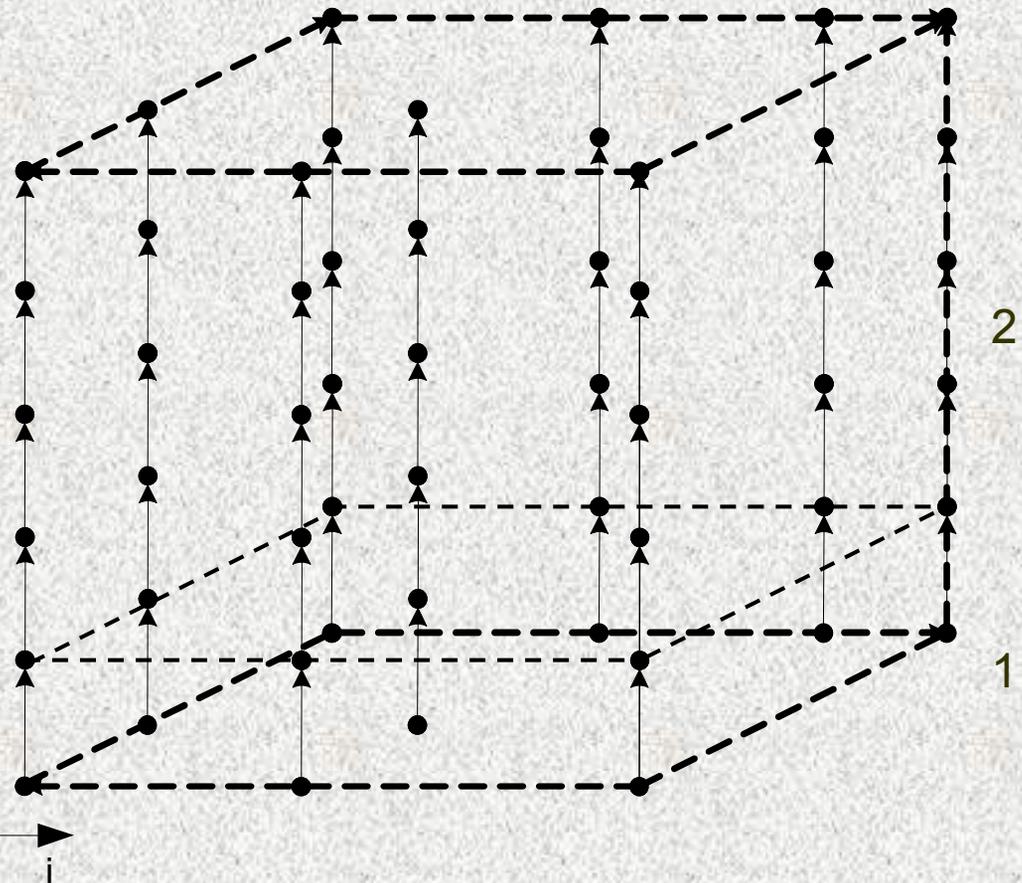


$$\left\{ \begin{array}{l} m \geq 1 \\ j_1 = m \\ \text{из } 3 \end{array} \right. \quad \left\{ \begin{array}{l} m < 1 \\ n \geq 1 \\ j_1 = n \\ \text{из } 2 \end{array} \right. \quad \left\{ \begin{array}{l} m < 1 \\ n < 1 \\ \text{из } 1 \end{array} \right.$$

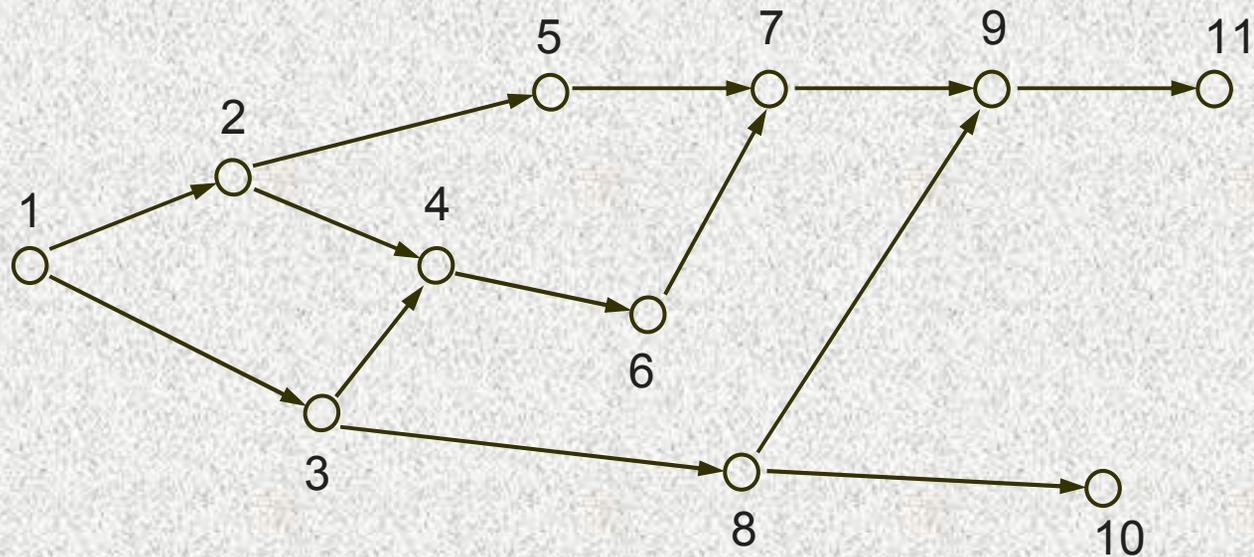
Программы и их графы алгоритма (умножение матриц)

```

Do i = 1, n
  Do j = 1, n
1   A(i,j) = 0
    Do k = 1, n
2   A(i,j) = A(i,j) + B(i,k)*C(k,j)
  
```

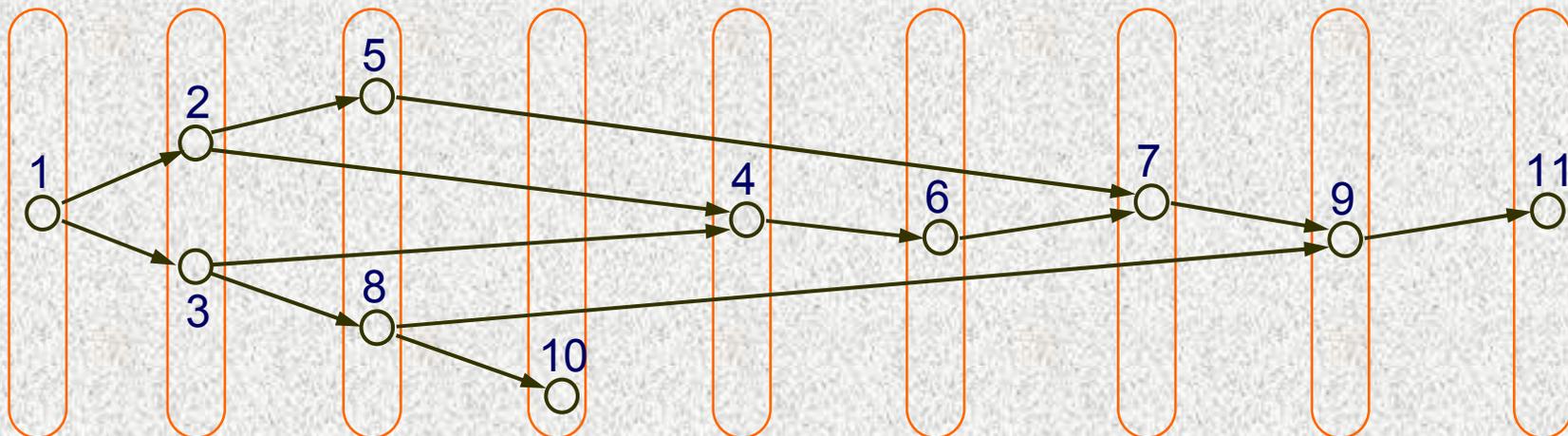
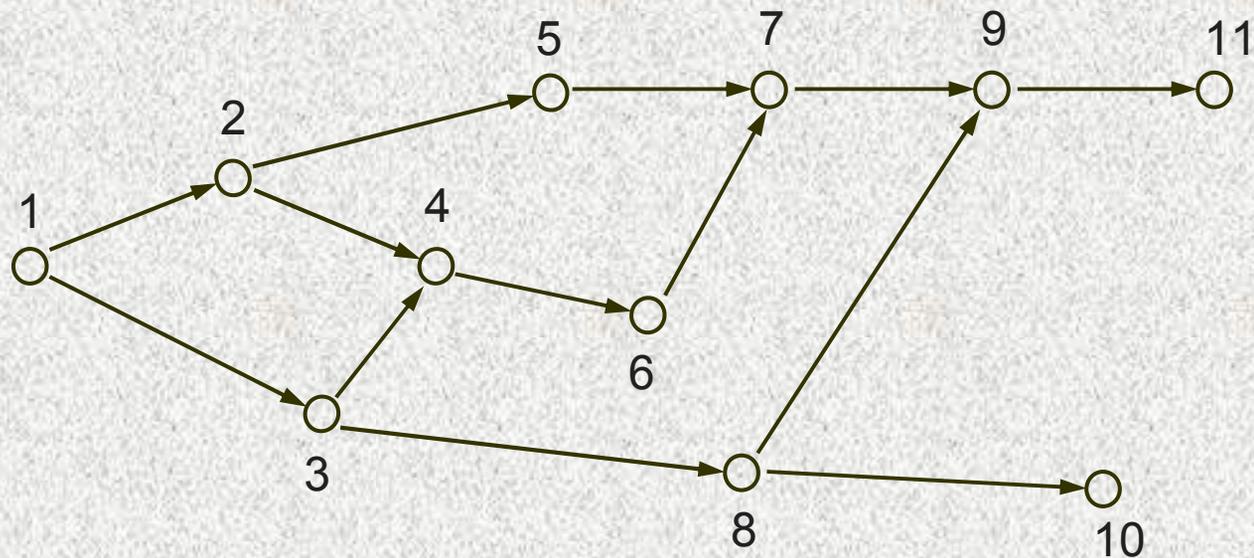


Ярусно-параллельная форма графа алгоритма

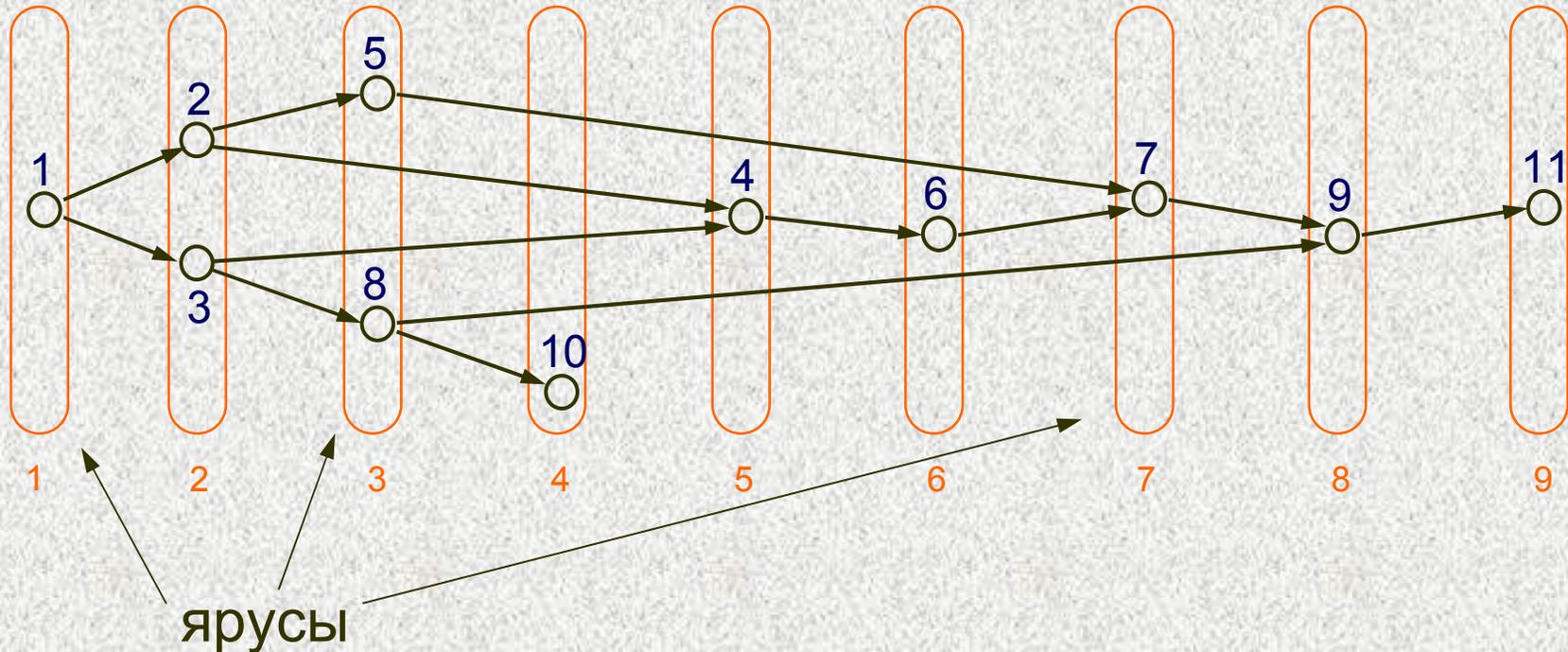


Как определить и сделать понятным ресурс параллелизма в графе алгоритма (в программе, в алгоритме) ?

Ярусно-параллельная форма графа алгоритма

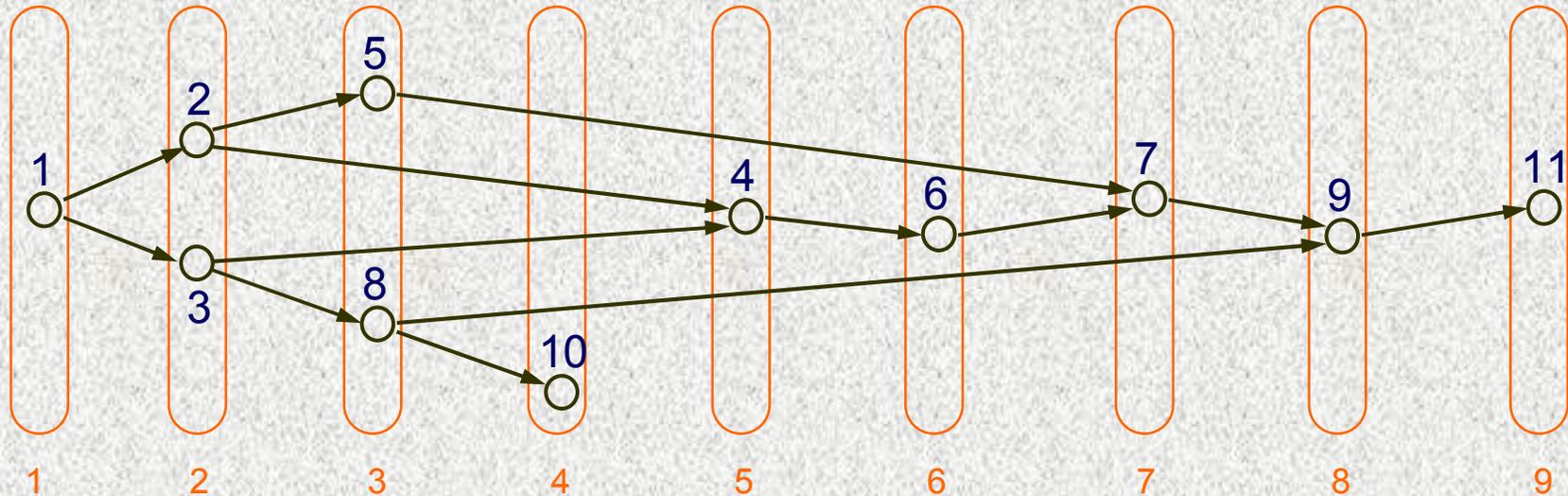


Ярусно-параллельная форма графа алгоритма



- начальная вершина каждой дуги расположена на ярусе с номером меньше, чем номер яруса конечной вершины.
- между вершинами, расположенными на одном ярусе, не может быть дуг.

Ярусно-параллельная форма графа алгоритма



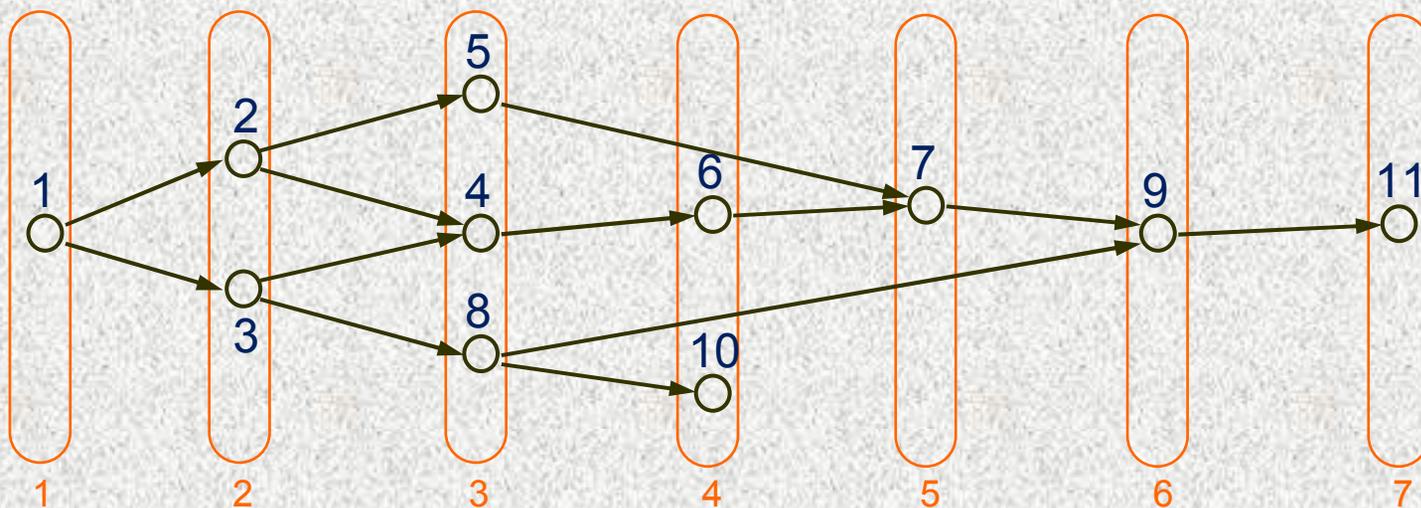
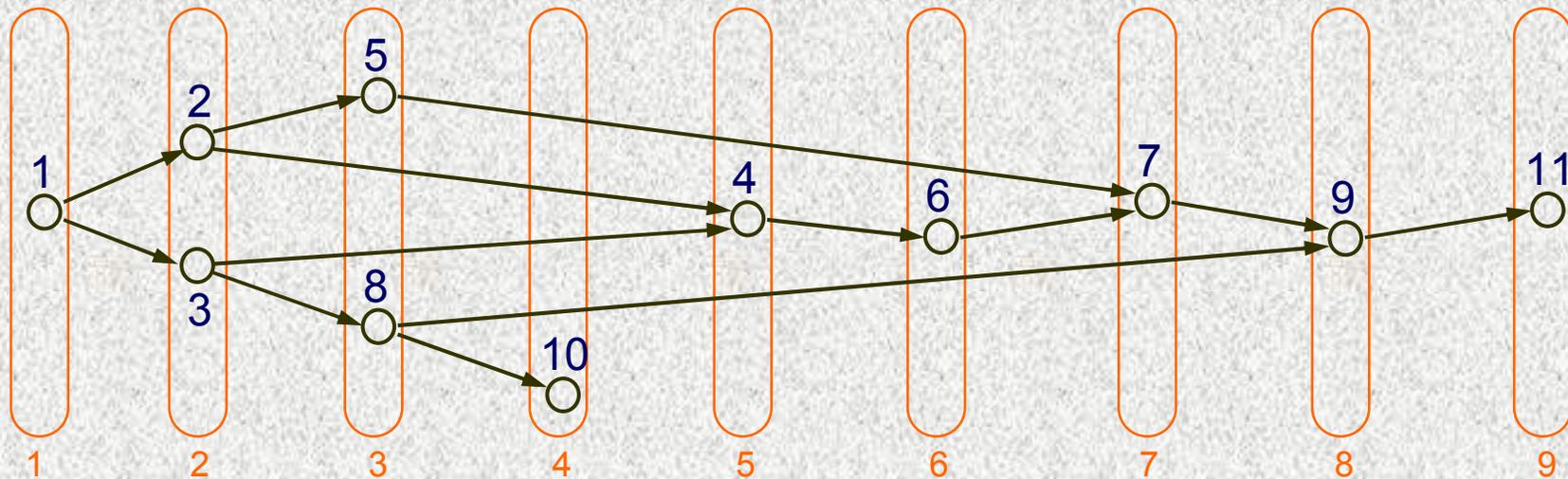
Высота ЯПФ – это число ярусов,

Ширина яруса – число вершин, расположенных на ярусе,

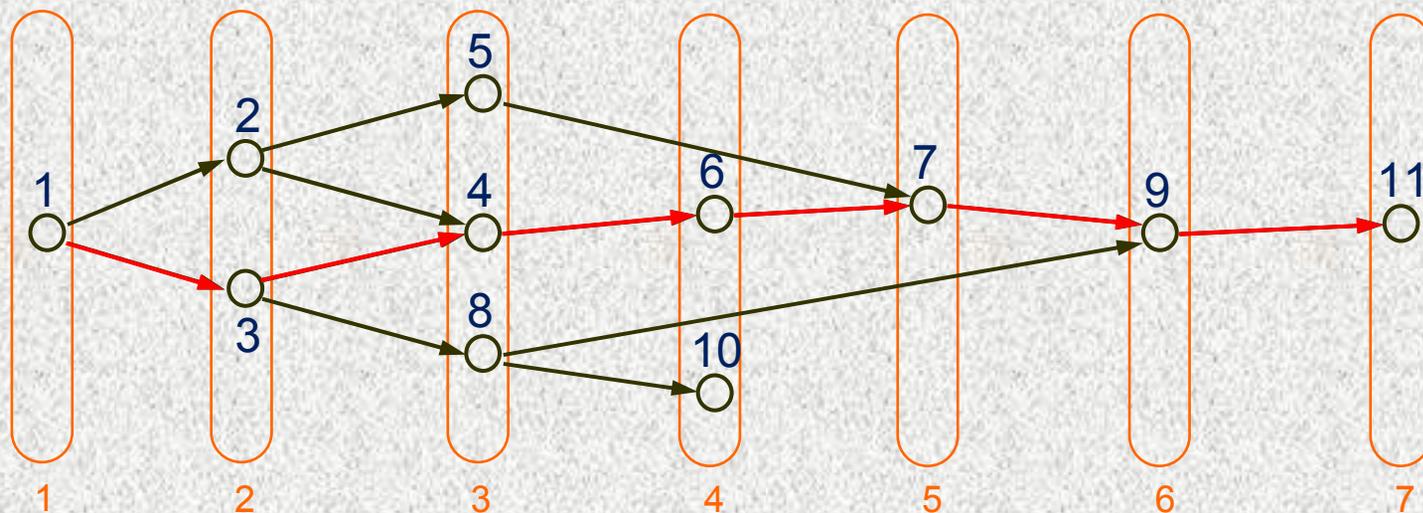
Ширина ЯПФ – это максимальная ширина ярусов в ЯПФ.

Высота ЯПФ = сложность параллельной реализации алгоритма/программы.

Ярусно-параллельная форма графа алгоритма



Каноническая ярусно-параллельная форма графа алгоритма



Высота канонической ЯПФ = длине критического пути + 1.

Каноническая ярусно-параллельная форма графа алгоритма

```
for( i = 0; i < n; ++i)
  for( j = 0; j < m; ++j)
    A[i][j] = A[i][j-1] + C[i][j]*x;
```

Чему, согласно закону Амдала, равно максимальное ускорение, которое можно получить при исполнении данного фрагмента на параллельной вычислительной системе?

Закон Амдала:

$$S \leq \frac{1}{\alpha + \frac{(1 - \alpha)}{p}}$$

где:

α – доля последовательных операций,
 p – число процессоров в системе.

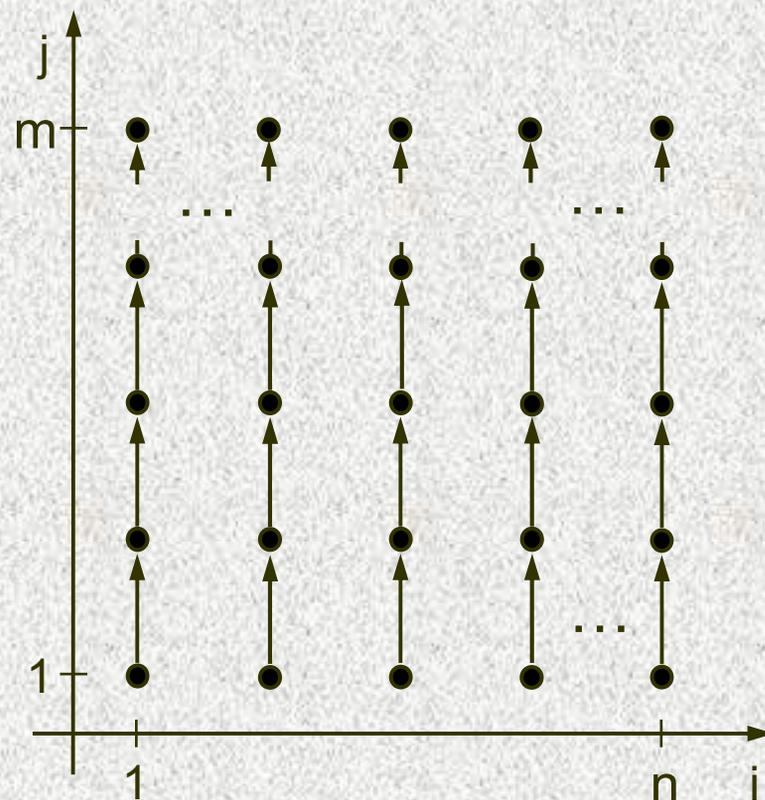
Каноническая ярусно-параллельная форма графа алгоритма

```
for( i = 0; i < n; ++i)
  for( j = 0; j < m; ++j)
    A[i][j] = A[i][j-1] + C[i][j]*x;
```

$$S \approx \frac{1}{\alpha}$$

$$\alpha = \frac{\text{Число последовательных операций}}{\text{Общее число операций}} = \frac{m}{n*m} = \frac{1}{n}$$

$$S \approx n$$



Виды параллелизма в алгоритмах и программах



Конечный параллелизм определяется информационной независимостью некоторых фрагментов в тексте программы.

Массовый параллелизм определяется информационной независимостью итераций циклов программы.

Виды параллелизма в алгоритмах и программах

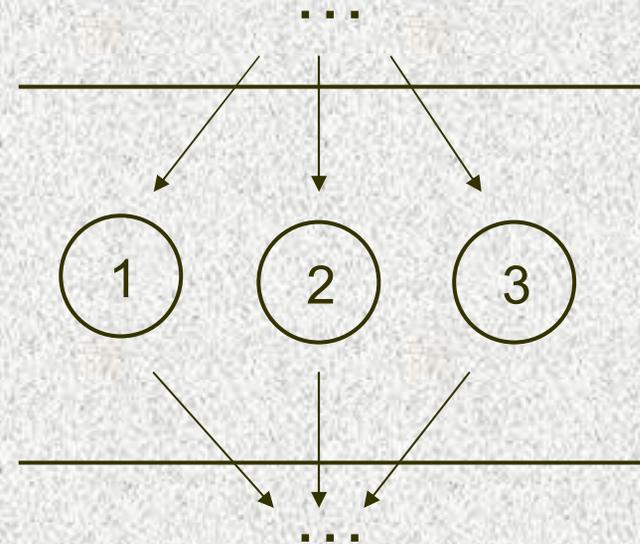
Конечный параллелизм.

```
cout << "N=" << N << endl;  
cycleTestWithUnroll_KJI("k");  
cycleTestWithUnroll_KJI("j");  
cycleTestWithUnroll_KJI("i");  
cycleTestWithUnroll_KJI_3("k");  
cycleTestWithUnroll_KJI_3("j");  
cycleTestWithUnroll_KJI_3("i");  
cycleTest("j,i,k");  
cycleTest("i,k,j");  
cycleTest("k,j,i");  
cycleTest("i,j,k");  
cycleTest("k,i,j");  
cycleTest("j,k,i");  
float ***a12=new float**[N];  
for (i=0;i<N;i++) {  
    a12[i]=new float*[N];  
    for (j=0;j<N;j++) {  
        a12[i][j]=new float[N];  
        for (k=0;k<N;k++) {  
            a12[i][j][k]=(float)1/(i+j+k+1);  
        }  
    }  
}  
for (i=1;i<N;i++)  
    for (j=1;j<N;j++)  
        for (k=1;k<N;k++) {  
            testee[i][k] = testee[i][k] + S[k]*A[k][i][i] + P[i][i]*A[k][i][i-1] +  
                P[i][k]*A[k][j-1][i] + P[j][k]*A[k-1][i][i];  
        }  
...  
}
```

1

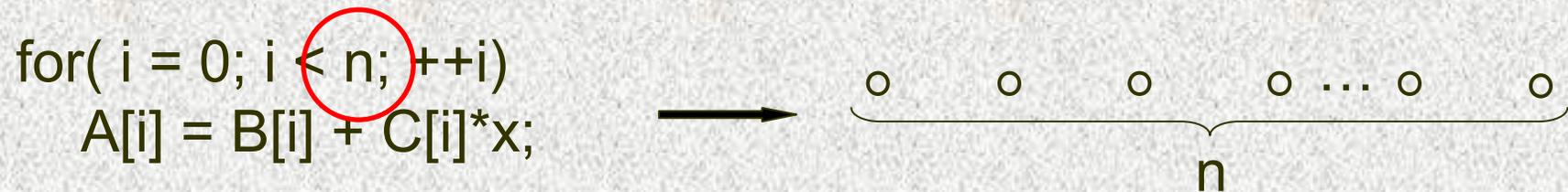
2

3

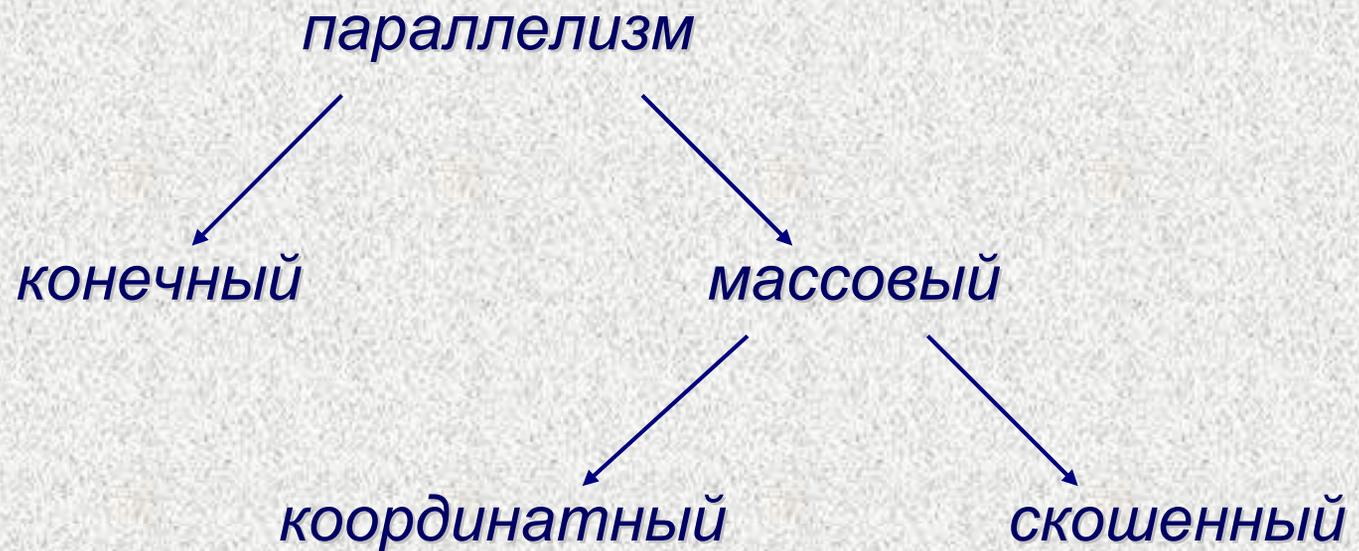


Виды параллелизма в алгоритмах и программах

Массовый параллелизм.



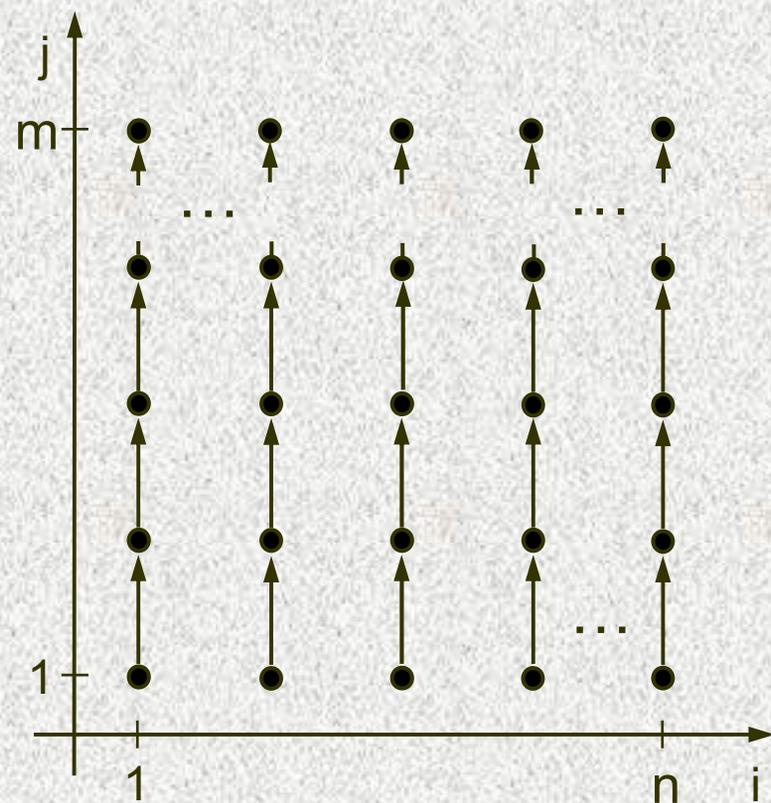
Виды параллелизма в алгоритмах и программах



Виды параллелизма в алгоритмах и программах

Координатный параллелизм.

```
#pragma omp parallel for  
for( i = 0; i < n; ++i)  
  for( j = 0; j < m; ++j)  
    A[i][j] = A[i][j-1] + C[i][j]*x;
```



Виды параллелизма в алгоритмах и программах

Координатный параллелизм.

Утверждение: для того чтобы цикл был параллельным необходимо и достаточно, чтобы для любой тройки графа алгоритма данного цикла включение $\Delta_i \subset G_i$ было верным, где

Δ_i — это многогранник из тройки,

$G_i = \{f_1 = i_1,$

i_1 — это параметр анализируемого цикла,

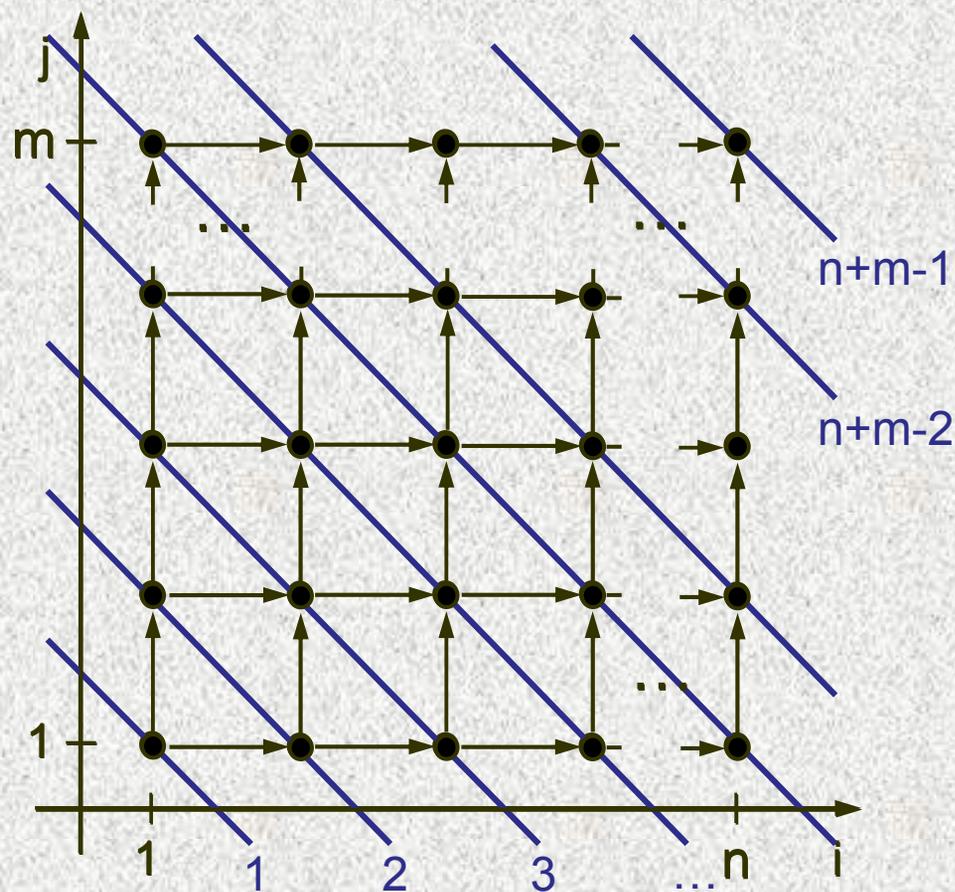
f_1 — это первая компонента векторной функции F_i из тройки.

Виды параллелизма в алгоритмах и программах

Скошенный параллелизм.

```
for( i = 0; i < n; ++i)
  for( j = 0; j < m; ++j)
    A[i][j] = A[i][j-1] + A[i-1][j]*x;
```

→



Эквивалентные преобразования программ

Исходная программа



Построение
графа алгоритма



Исследование
графа алгоритма



Преобразование
графа алгоритма

Преобразованная программа

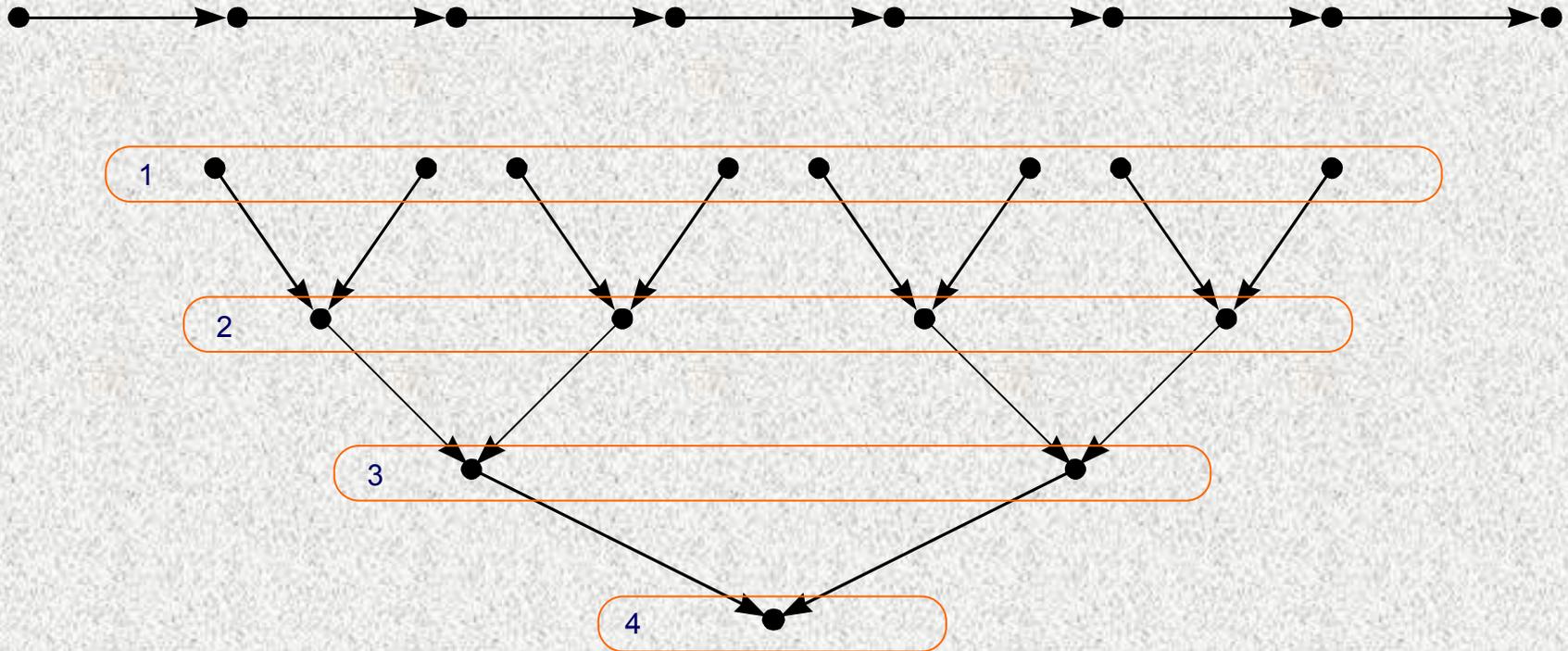


Эквивалентные преобразования программ (суммирование элементов массива)

```
s = 0.0;  
for ( i = 0; i < n; ++i )  
    s = s + A[ i ];
```

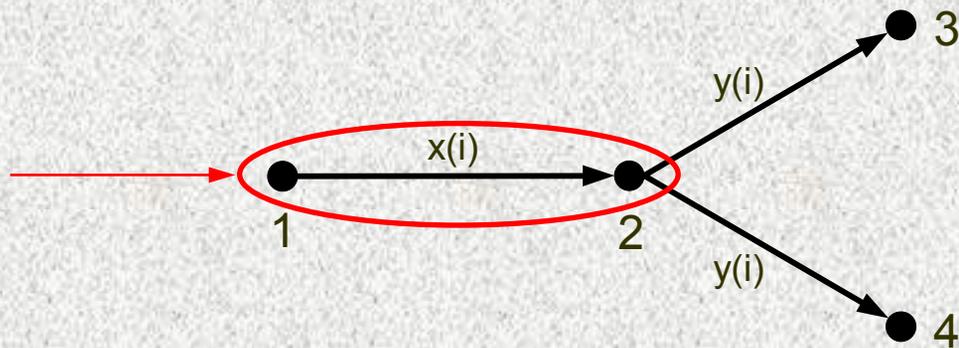


Эквивалентные преобразования программ (суммирование элементов массива)



Эквивалентные преобразования программ

*Исполнять только
последовательно !*



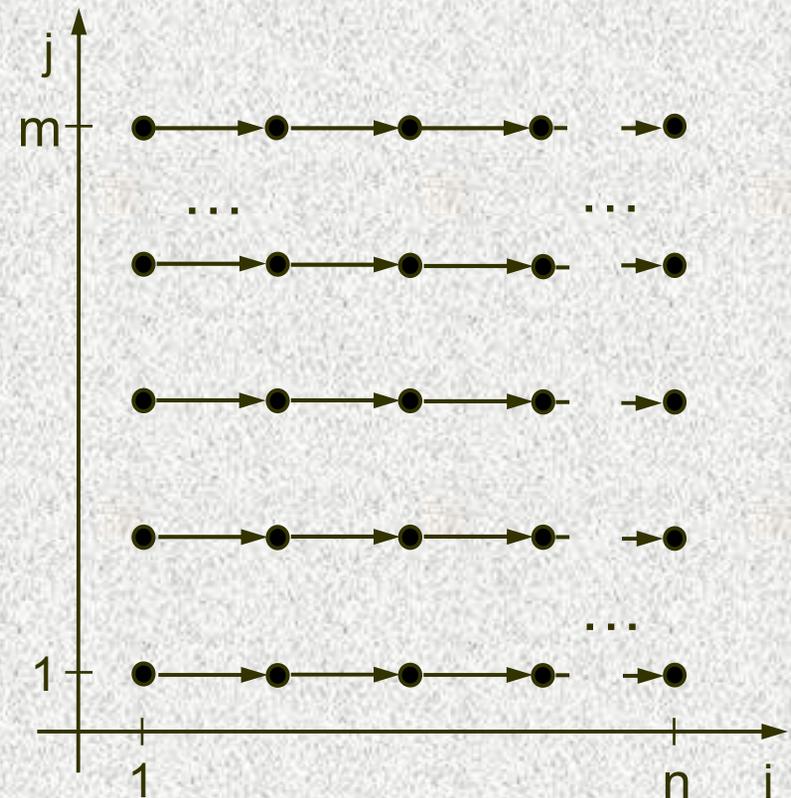
Информационная зависимость определяет критерий эквивалентности преобразований программ.

Информационная независимость определяет ресурс параллелизма программы.

Элементарные преобразования циклов

Перестановка циклов.

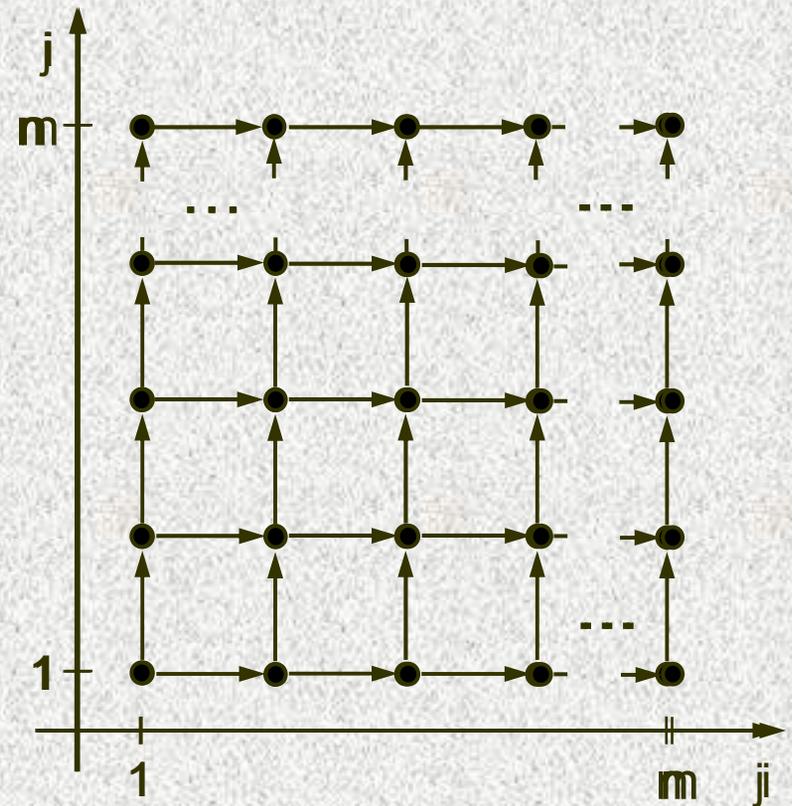
```
for( i = 0; i < n; ++i)
  #pragma omp parallel for
    A[i][j] = A[i-1][j] + C[i][j]*x;
```



Элементарные преобразования циклов

Перестановка циклов.

```
#pragma omp parallel for  
for( i = 0; i < n; ++i)  
  for( j = 0; j < m; ++j)  
    A[i][j] = A[i-1][j] + C[i][j]*x;
```

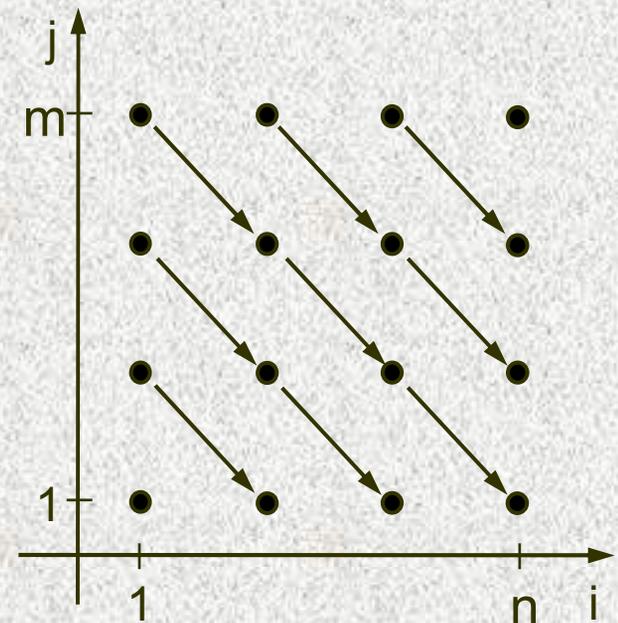


Элементарные преобразования циклов

Всегда ли перестановка циклов является эквивалентным преобразованием?

```
for( i = 0; i < n; ++i)
  for( j = 0; j < m; ++j)
    A[i][j] = A[i+c1][j+c2] + C[i][j]*x;
```

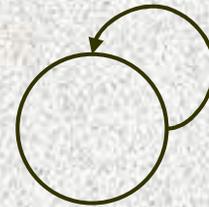
```
for( i = 0; i < n; ++i)
  for( j = 0; j < m; ++j)
    A[i][j] = A[i-1][j+1] + C[i][j]*x;
```



Элементарные преобразования циклов

Распределение циклов.

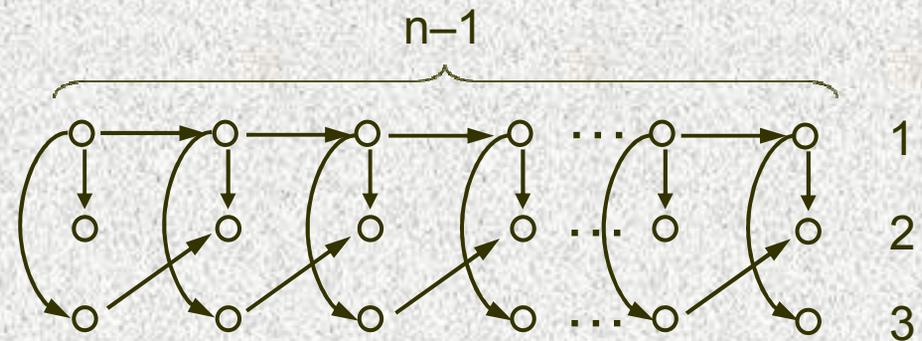
```
for( i = 1; i < n; ++i) {  
1   A[i] = A[i-1]*p + q;  
2   C[i] = (A[i] + B[i-1])*s;  
3   B[i] = (A[i] - B[i])*t;  
}
```



Элементарные преобразования циклов

Распределение циклов.

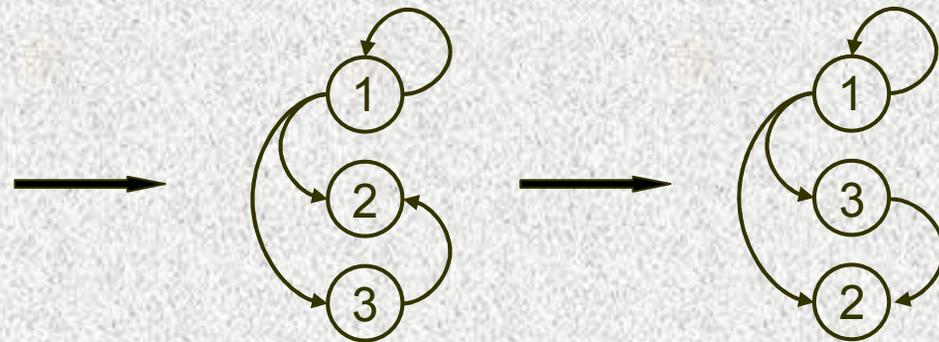
```
for( i = 1; i < n; ++i) {  
1   A[i] = A[i-1]*p + q;  
2   C[i] = (A[i] + B[i-1])*s;  
3   B[i] = (A[i] - B[i])*t;  
}
```



Элементарные преобразования циклов

Распределение циклов.

```
for( i = 1; i < n; ++i) {  
1   A[i] = A[i-1]*p + q;  
2   C[i] = (A[i] + B[i-1])*s;  
3   B[i] = (A[i] - B[i])*t;  
}
```



Утверждение: для того чтобы можно было выполнить распределение цикла необходимо и достаточно, чтобы распределяемые части находились в разных компонентах сильной связности информационного графа тела данного цикла.

Элементарные преобразования циклов

Распределение циклов.

```
for( i = 1; i < n; ++i) {  
    A[i] = A[i-1]*p + q;  
    C[i] = (A[i] + B[i-1])*s;  
    B[i] = (A[i] - B[i])*t;  
}
```

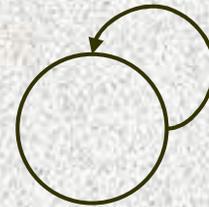


```
for( i = 1; i < n; ++i)  
    A[i] = A[i-1]*p + q;  
#pragma omp parallel for  
for( i = 1; i < n; ++i)  
    B[i] = (A[i] - B[i])*t;  
#pragma omp parallel for  
for( i = 1; i < n; ++i)  
    C[i] = (A[i] + B[i-1])*s;
```

Элементарные преобразования циклов

Расщепление циклов.

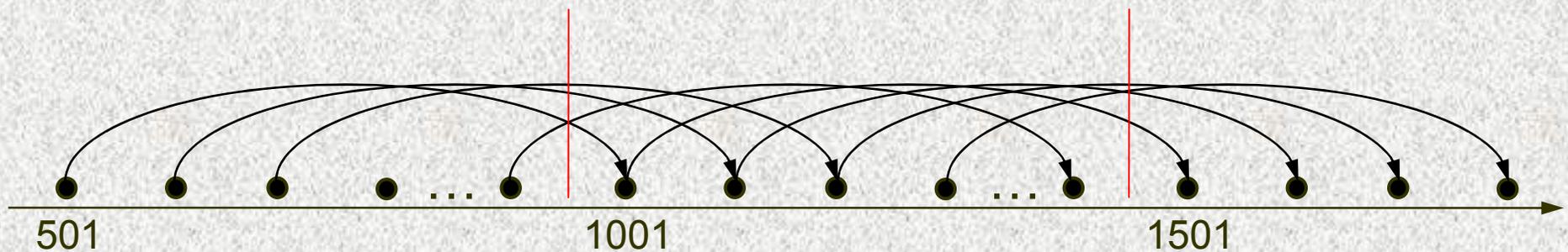
```
for( i = 501; i <= 2000; ++i)  
  A[i] = A[i] + A[i-500];
```



Элементарные преобразования циклов

Расщепление циклов.

```
for( i = 501; i <= 2000; ++i)  
  A[i] = A[i] + A[i-500];
```



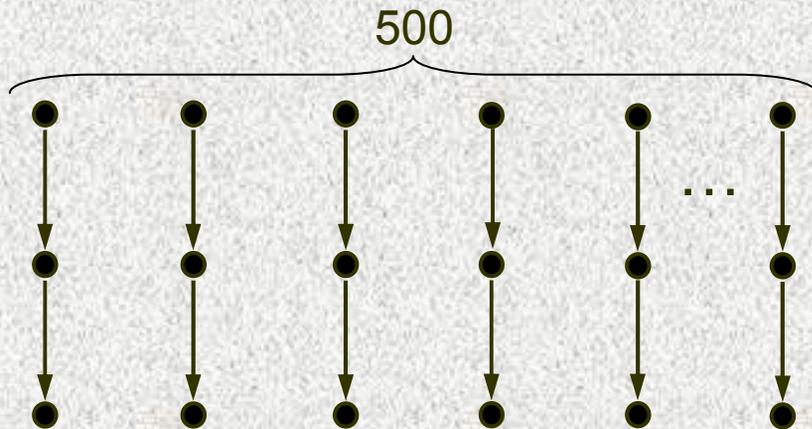
Элементарные преобразования циклов

Расщепление циклов.

```
for( i = 501; i <= 2000; ++i)  
  A[i] = A[i] + A[i-500];
```



```
#pragma omp parallel for  
for( i = 501; i <= 1000; ++i)  
  A[i] = A[i] + A[i-500];  
#pragma omp parallel for  
for( i = 1001; i <= 1500; ++i)  
  A[i] = A[i] + A[i-500];  
#pragma omp parallel for  
for( i = 1501; i <= 2000; ++i)  
  A[i] = A[i] + A[i-500];
```



Эквивалентные преобразования программ

Эквивалентно ли преобразование?

```
for( i = 0; i < n; ++i)
```

```
    D[i] = D[i] * F[i];
```

```
if( m == 3 )
```

```
    for( i = 0; i < n; ++i)
```

```
        R[i] = P[i] + D[i];
```

```
else
```

```
    for( i = 0; i < n; ++i)
```

```
        R[i] = Q[i] - D[i];
```



```
if( m == 3 )
```

```
    for( i = 0; i < n; ++i)
```

```
        R[i] = P[i] + D[i] * F[i];
```

```
else
```

```
    for( i = 0; i < n; ++i)
```

```
        R[i] = Q[i] - D[i] * F[i];
```

Простой пример...

(последовательный вариант)

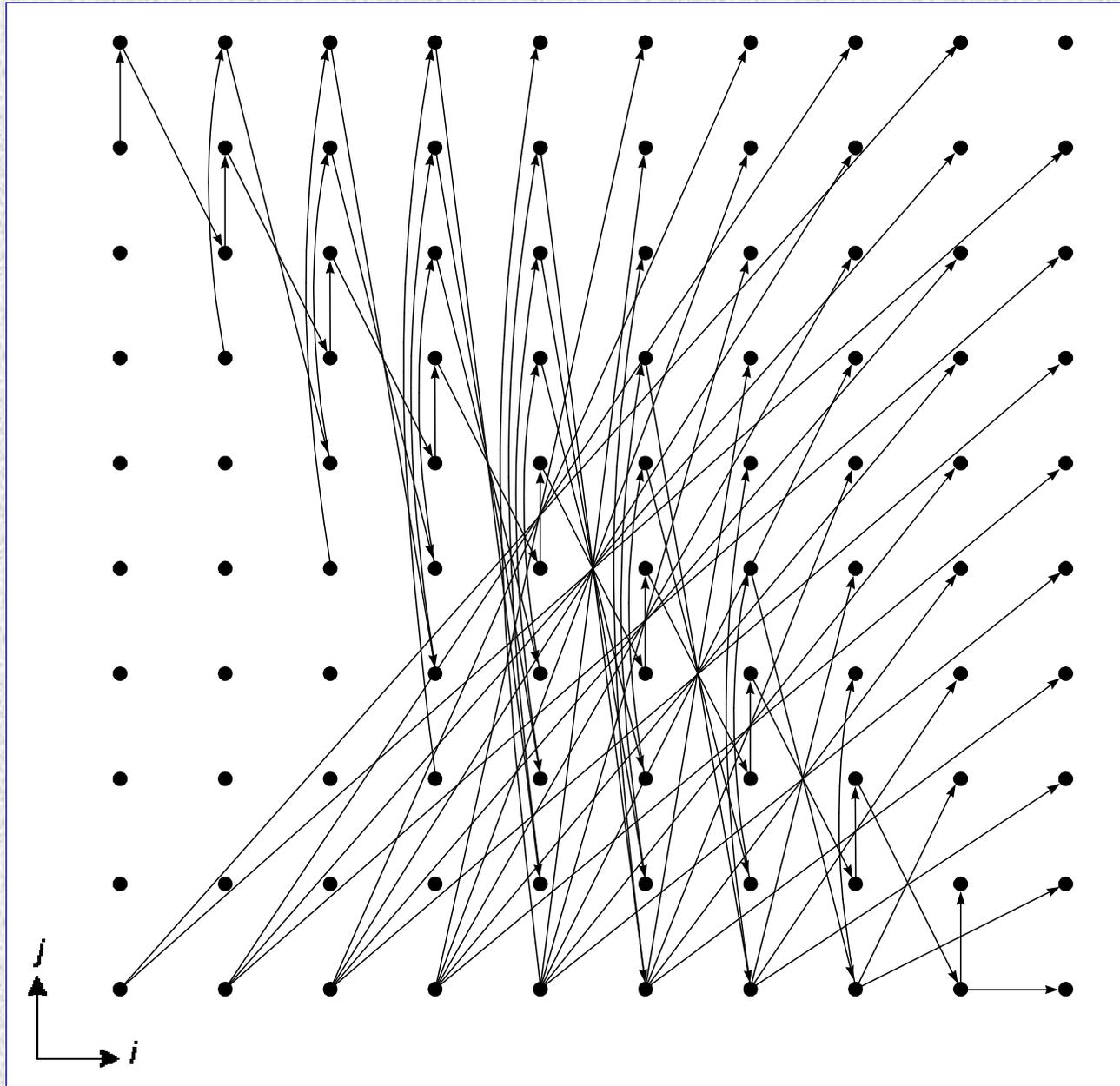
```
DO i = 1, n
```

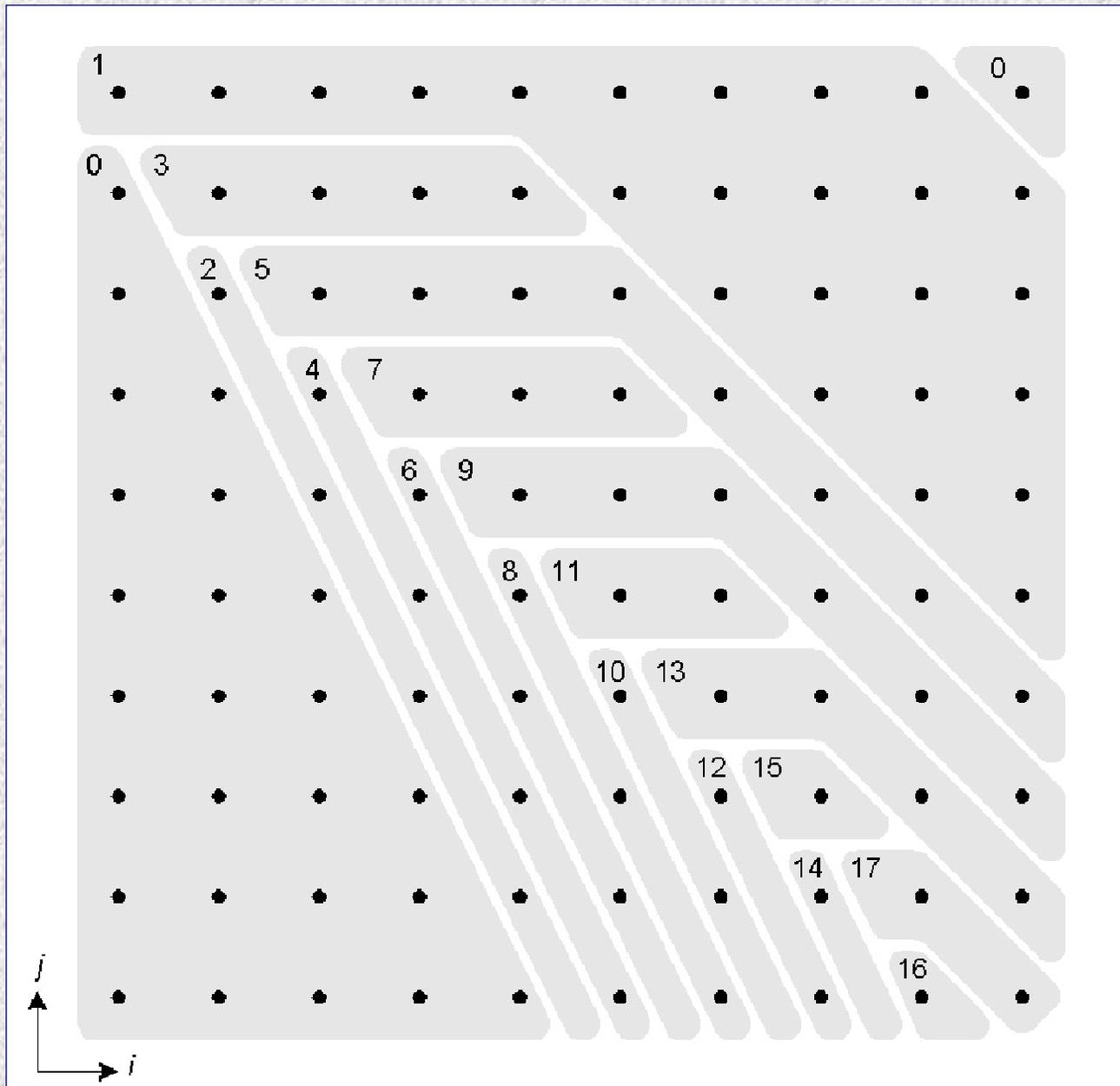
```
  DO j = 1, n
```

```
    U(i + j) = U( 2*n - i - j + 1)*q + p
```

```
  EndDO
```

```
EndDO
```





Совсем не простой пример...

(параллельный вариант)

DO i = 1, n

DO j = 1, n - i **Параллельный цикл !**

$$U(i + j) = U(2 * n - i - j + 1) * q + p$$

End DO

DO j = n - i + 1, n **Параллельный цикл !**

$$U(i + j) = U(2 * n - i - j + 1) * q + p$$

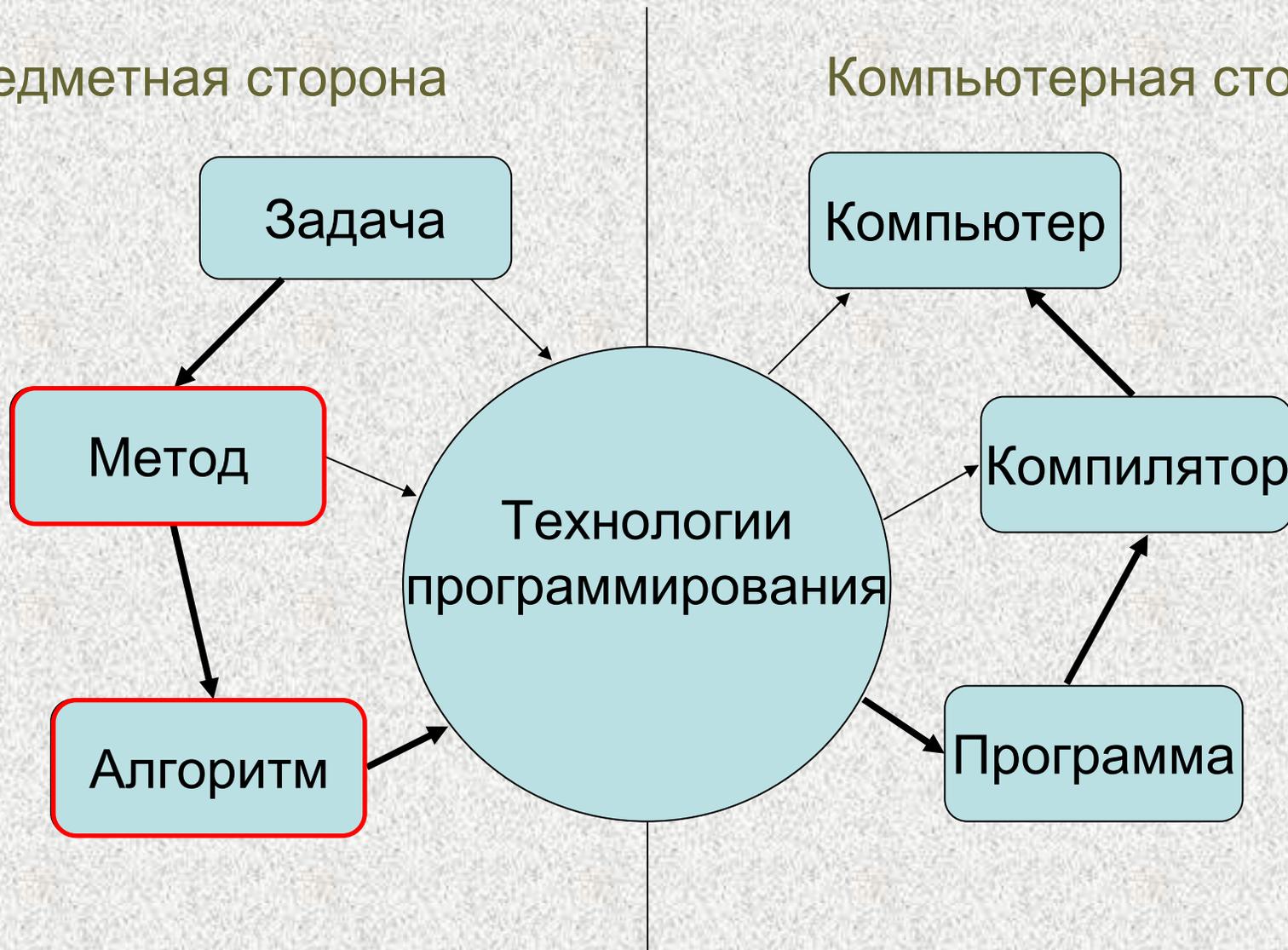
End DO

End DO

Решение задачи на компьютере

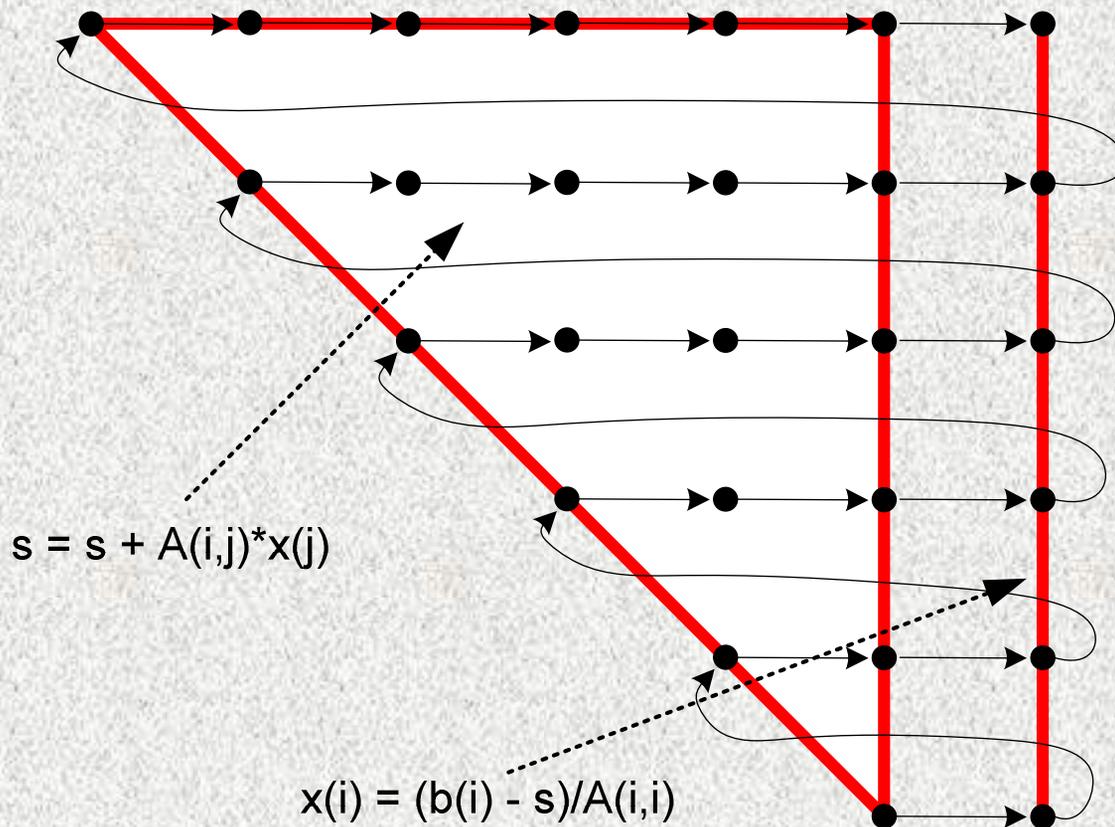
Предметная сторона

Компьютерная сторона



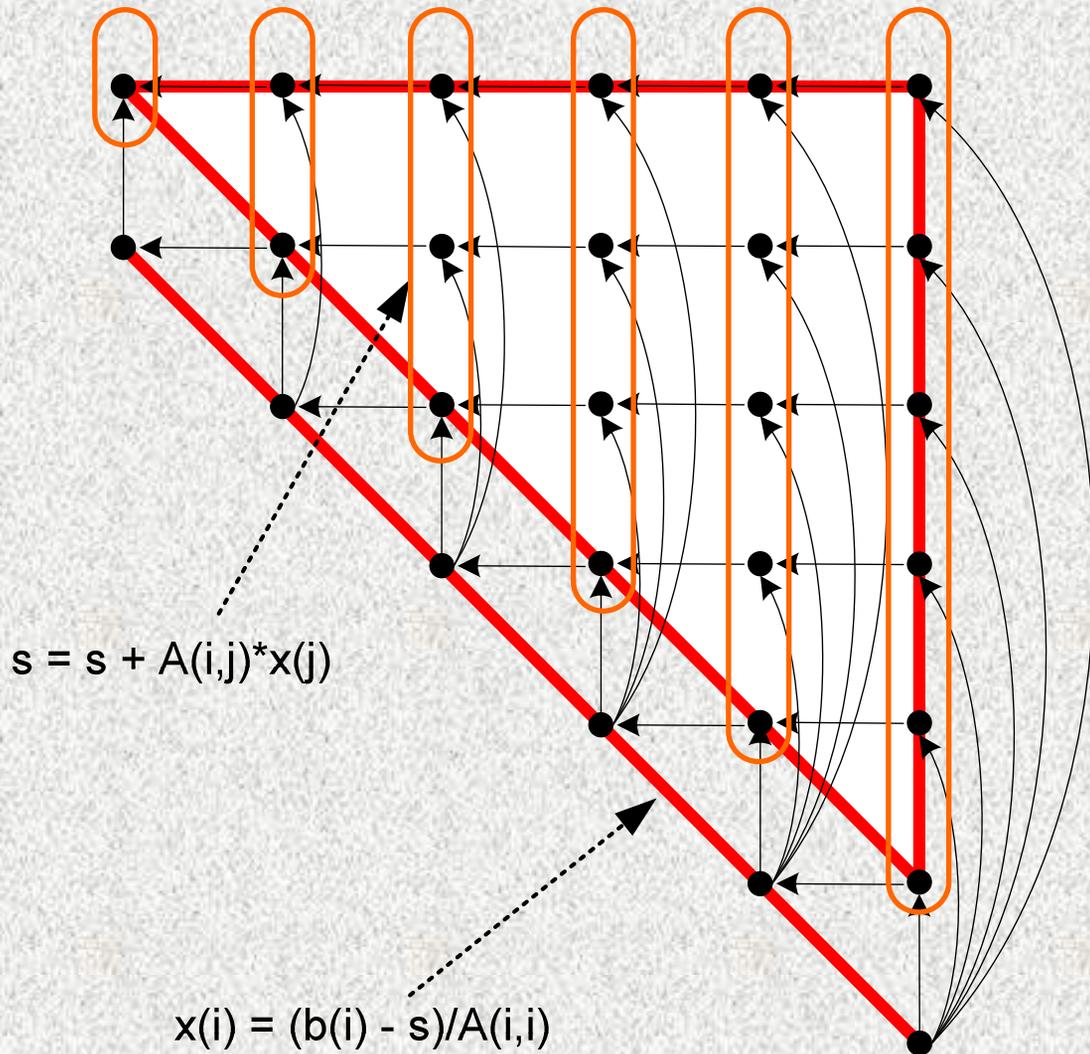
Решение СЛАУ: от метода к алгоритму

(информационная структура, последовательный вариант)



```
do i = n, 1, -1
  s = 0
  do j = i+1, n
    s = s + A(i,j)*x(j)
  end do
  x(i) = (b(i) - s)/A(i,i)
end do
```

Решение СЛАУ: от метода к алгоритму (информационная структура, параллельный вариант)



```
do i = n, 1, -1
  s = 0
  do j = n, i+1, -1
    s = s + A(i,j)*x(j)
  end do
  x(i) = (b(i) - s)/A(i,i)
end do
```

Где узнать больше?

