

Международная летняя суперкомпьютерная Академия

Математические основы параллельных вычислений

Воеводин Вл.В.
профессор, МГУ имени М.В.Ломоносова

voevodin@parallel.ru

26 июня 2013 г., МГУ, Москва

*Почему важно понимать как
написаны программы?*

*Почему важно знать как
устроены алгоритмы?*

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

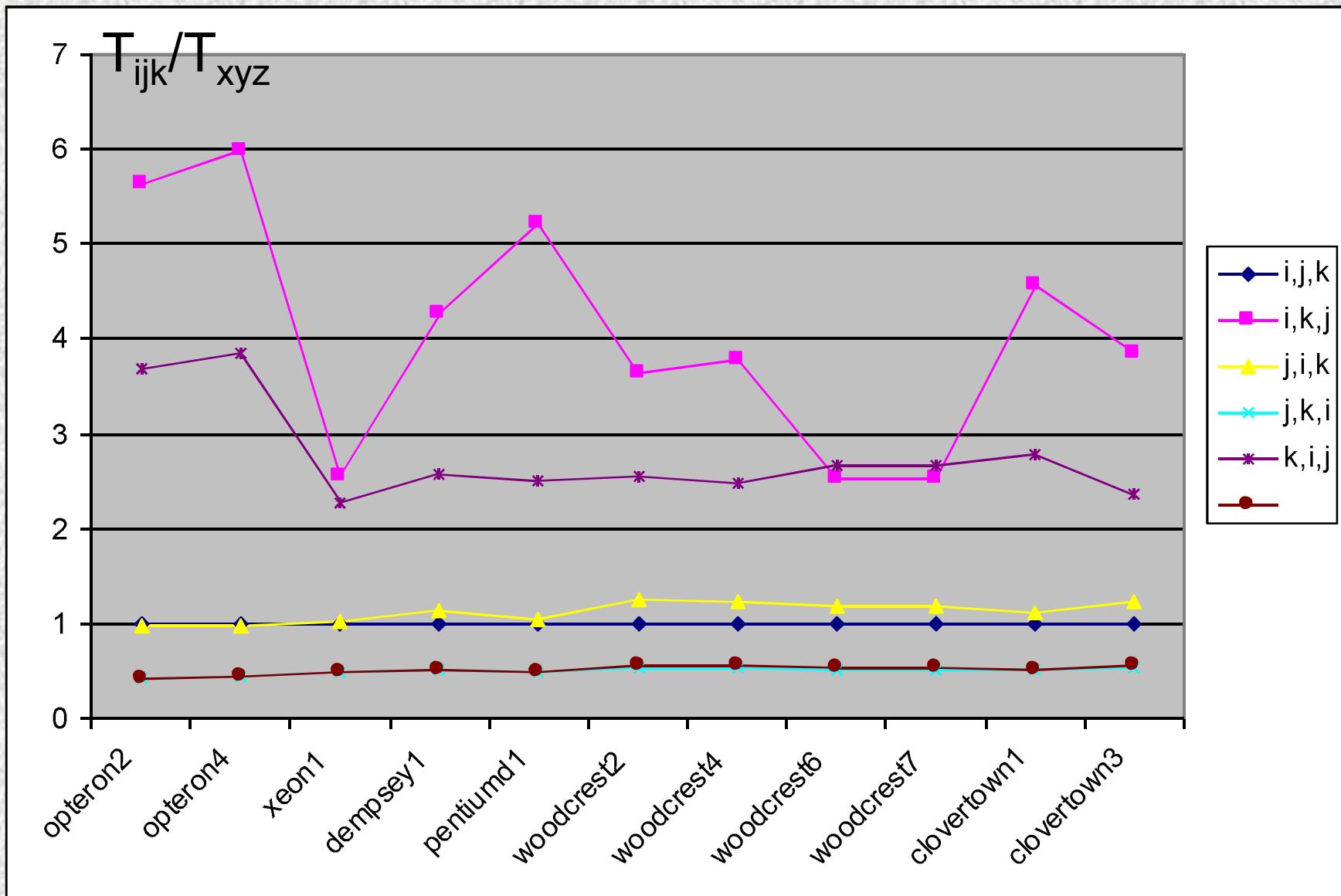
(j, k, i) - ? ДА

Порядок циклов: (i, j, k)

*Почему возможен
другой порядок?*

*А зачем нужен
другой порядок?*

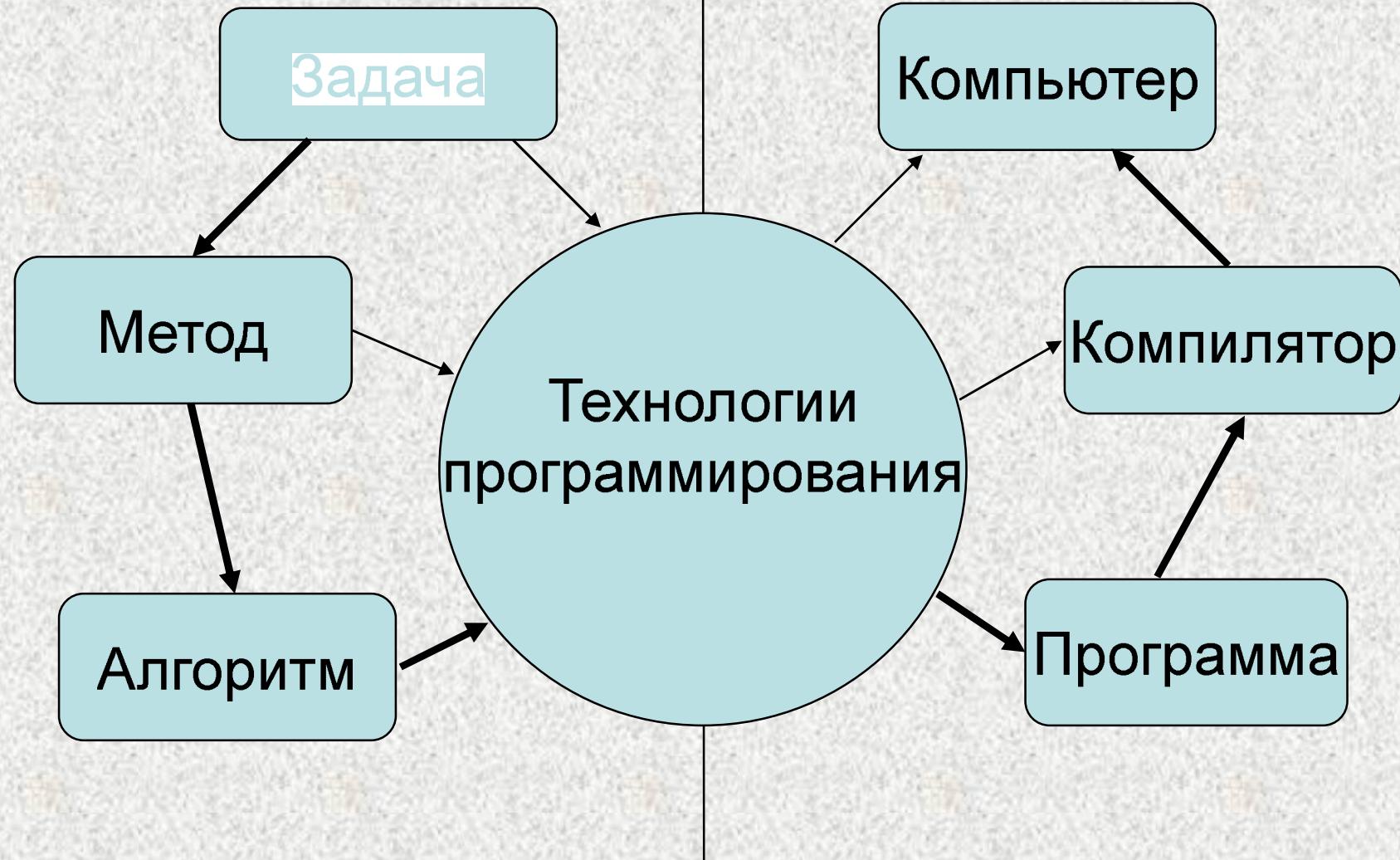
Умножение матриц: все ли просто? (сравнение с порядком (i, j, k))



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.*

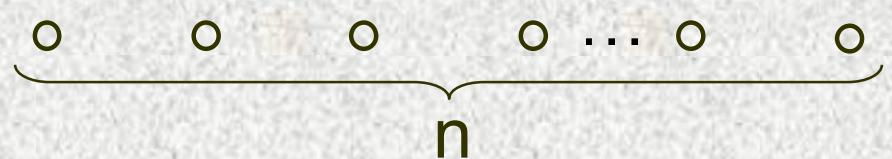
*Вершины: процедуры, циклы, линейные участки,
операторы, итерации циклов, срабатывания
операторов...*

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];  
}
```



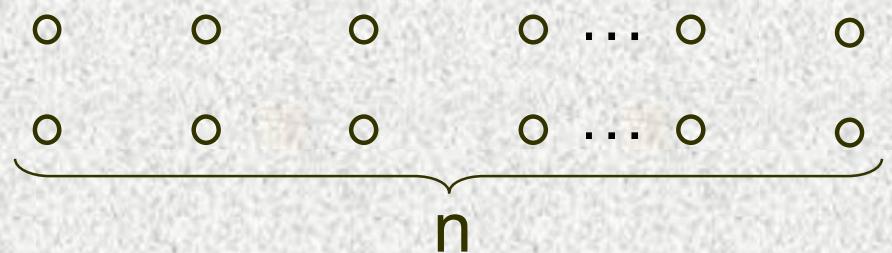
*Каждая вершина соответствует
двум операторам (телу цикла),
выполненным на одной и той же
итерации цикла.*

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 А и В соединяются направленной дугой
тогда и только тогда, когда вершина В может быть
выполнена сразу после вершины А.*

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 А и В соединяются направленной дугой тогда и только тогда, когда вершина В использует в качестве аргумента некоторое значение, полученное в вершине А.

Информационное отношение = отношение по передаче данных.

Графовые модели программ

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

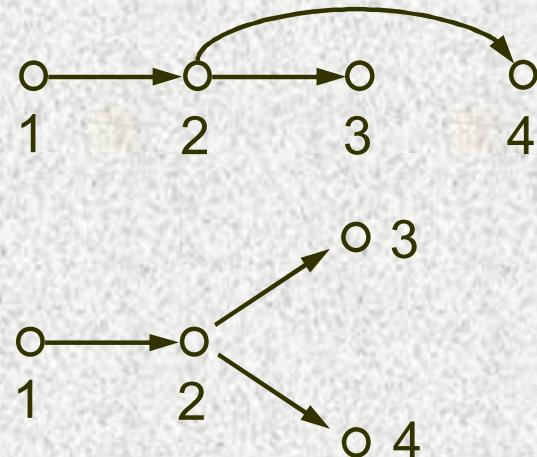
Дуги: информационное отношение:

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

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

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

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



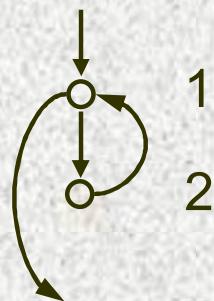
Четыре основные модели программ

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)  
}
```



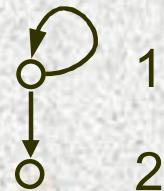
Четыре основные модели программ

Информационный граф программы.

Вершины: операторы

Дуги: информационное отношение

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

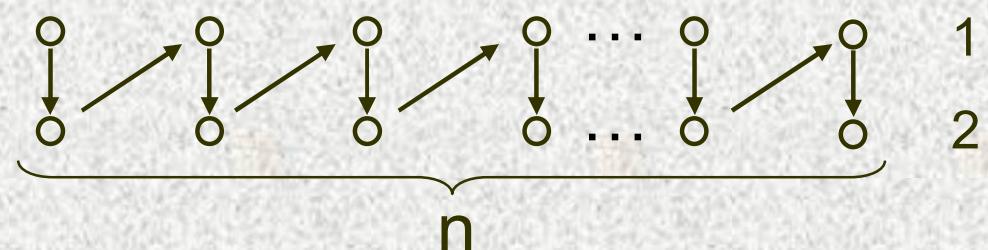


Четыре основные модели программ

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];  
}
```

(1)
(2)



Четыре основные модели программ

Информационная история программы.

Вершины: срабатывания операторов

Дуги: информационное отношение

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

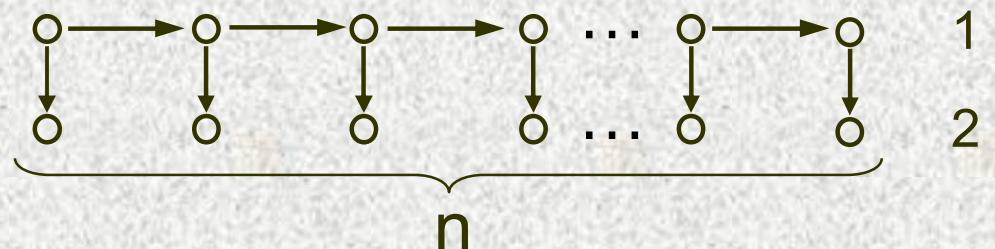
```
    A[i] = A[i - 1] + 2;
```

```
    B[i] = B[i] + A[i];
```

```
}
```

(1)

(2)

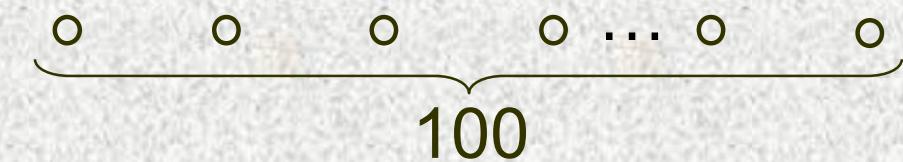


Несколько вопросов...

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;
```



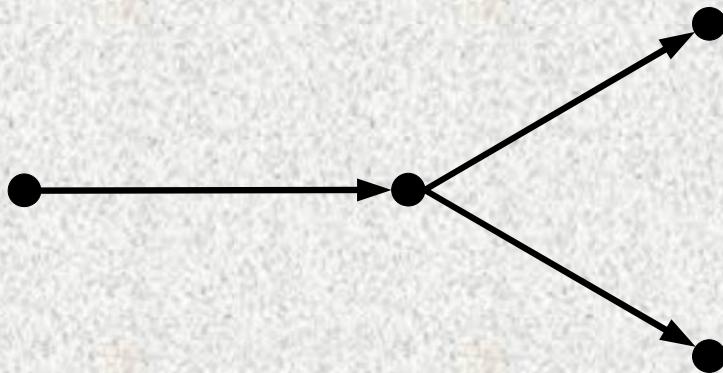
Несколько вопросов...

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?

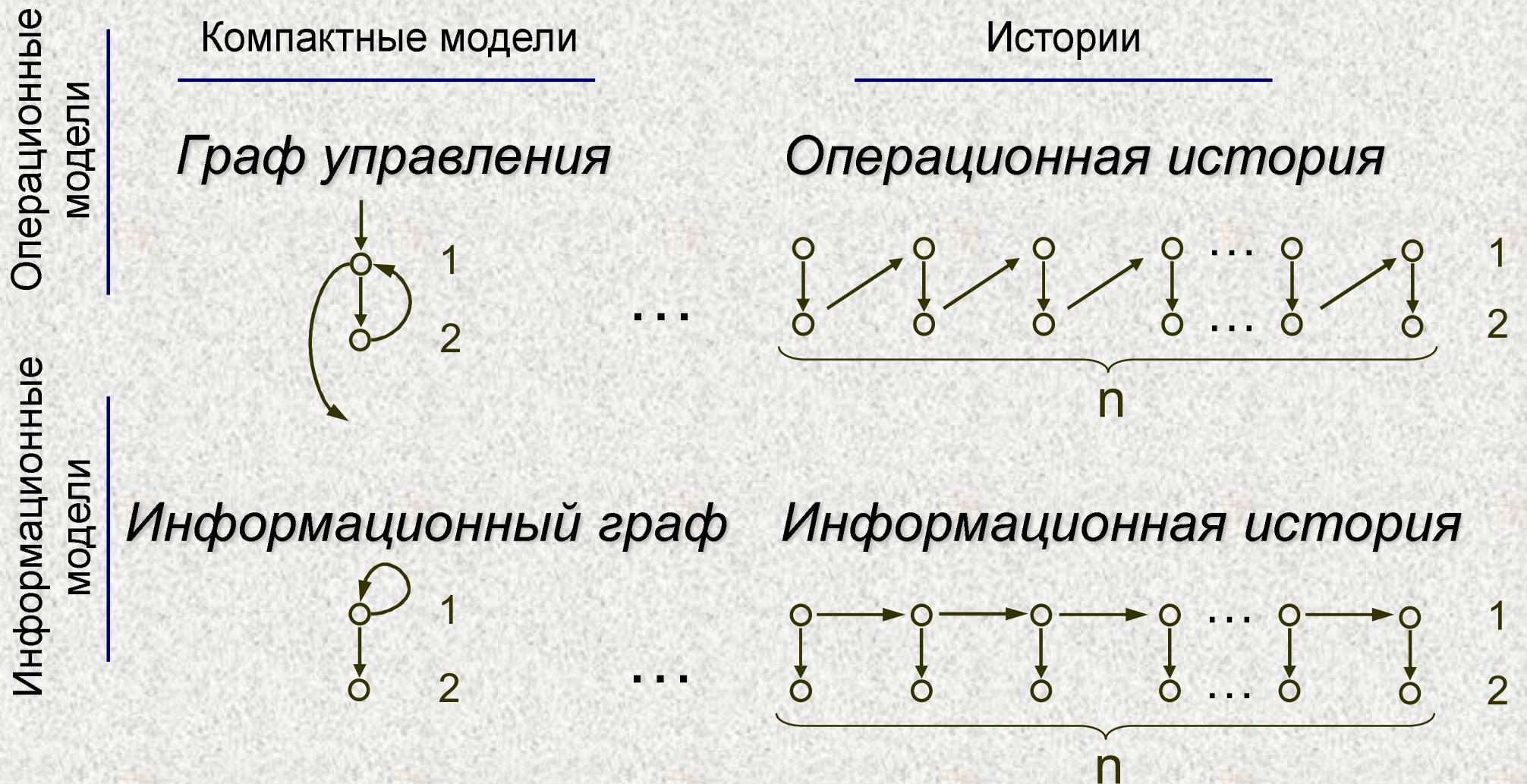
ГУ

ИГ

~~ОИ~~

ИИ

Множество графовых моделей программ (опорные точки)



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.

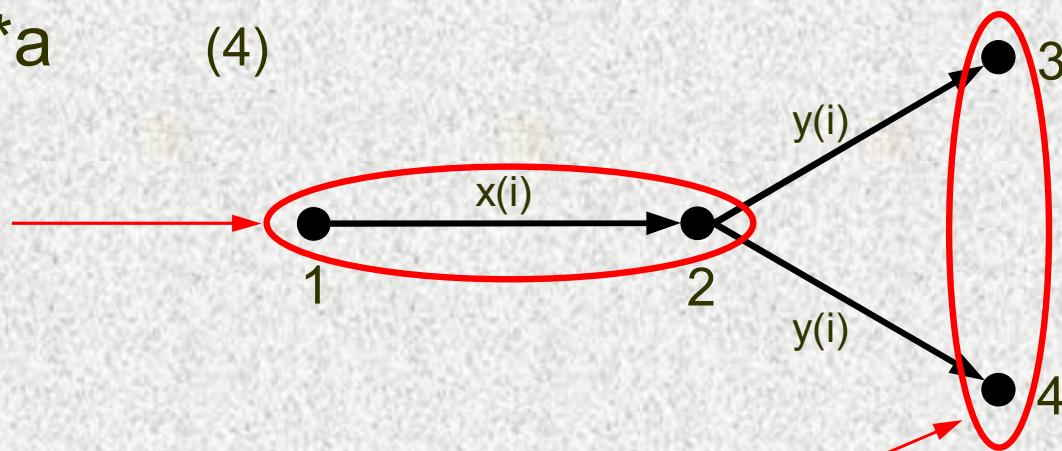
$$x(i) = a + b(i) \quad (1)$$

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

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

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

Исполнять только последовательно !

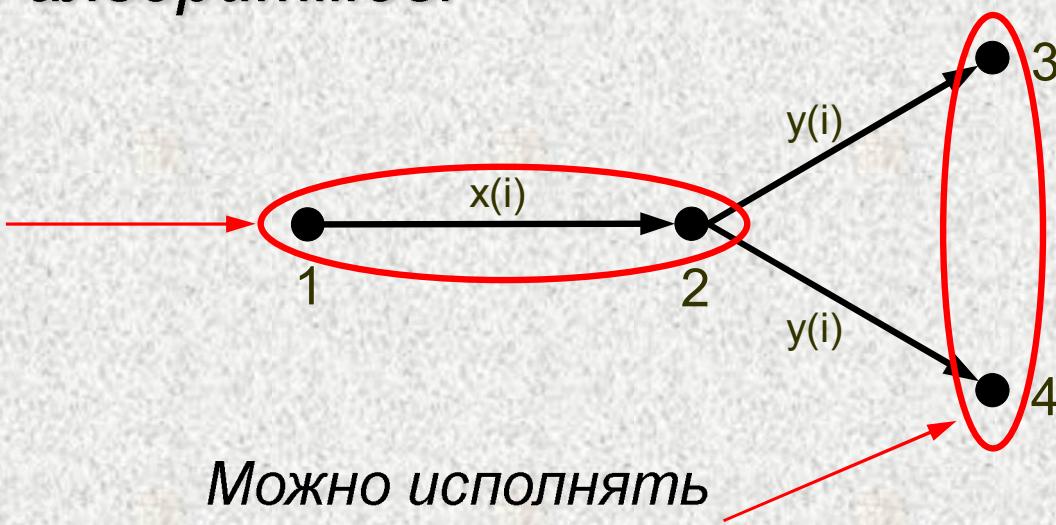


Можно исполнять параллельно !

Информационная структура программ и алгоритмов

Информационная структура – это основа анализа
свойств программ и алгоритмов.

Исполнять только
последовательно !



Информационная зависимость определяет критерий
эквивалентности преобразований программ.

Информационная независимость определяет ресурс
параллелизма программы.

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

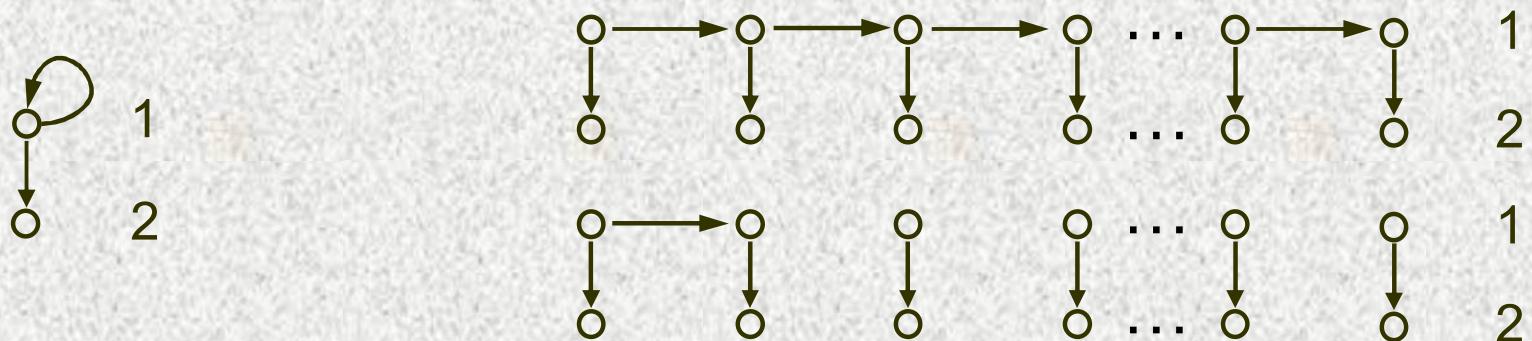
Аргументы для выбора степени компактности модели:

- *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?

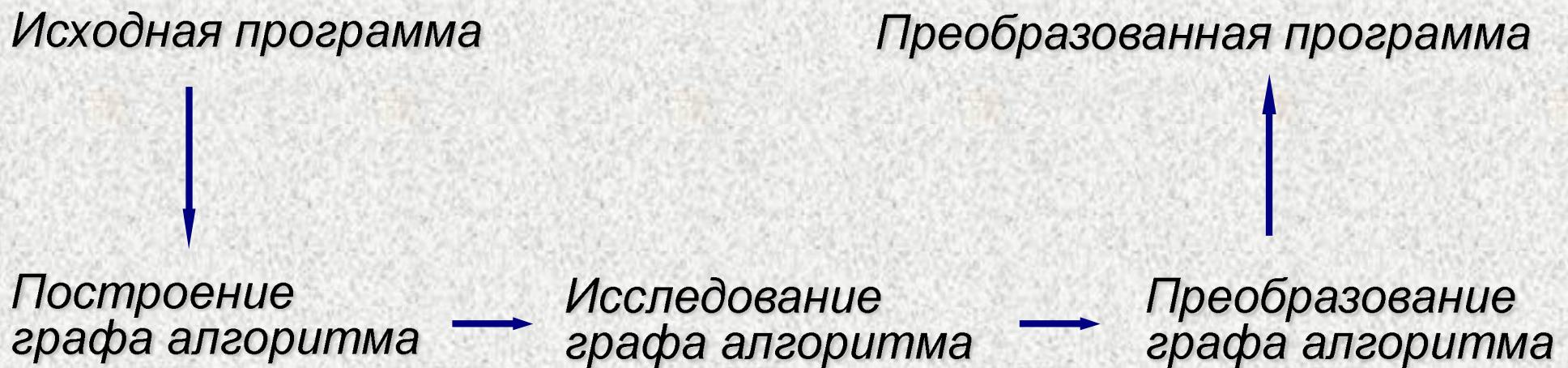
Аргументы для выбора степени компактности модели:

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

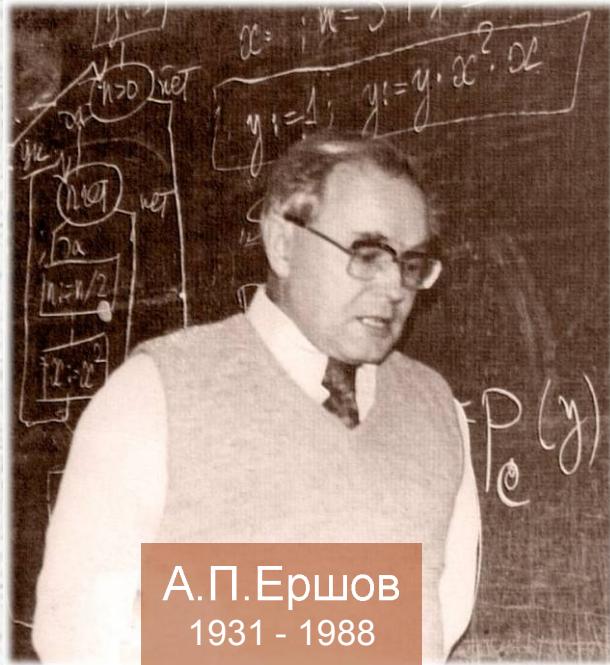
Граф алгоритма – это параметризованная информационная история:

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

Схема анализа и преобразования структуры программ



Основатели теории анализа структуры программ и алгоритмов



А.П. Ершов
1931 - 1988

Ершов Андрей Петрович, академик, создатель сибирской школы системного и теоретического программирования. Многие его работы посвящены методам изучения свойств и структуры программ. Еще в 60-х годах он рассматривал задачу преобразования схем программ над общей и分散ной памятью, изучал фундаментальные основы графовых моделей программ.

Воеводин Валентин Васильевич, академик, создатель математической теории информационной структуры программ и алгоритмов. Разработал методы нахождения и описания информационной структуры программ по их исходному тексту, методы определения потенциала параллелизма и эквивалентного преобразования программ.

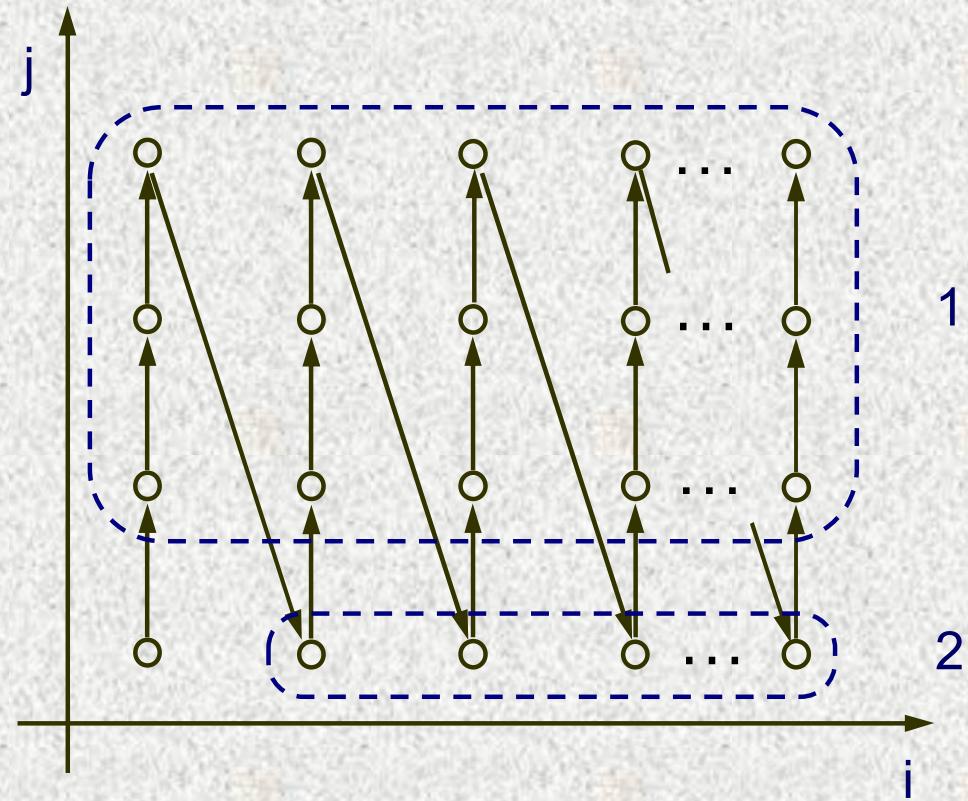


В.В. Воеводин
1934 - 2007

*Как выполняется описание
структур программы
на практике?*

Программы и их графы алгоритма

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



Для входа s:

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

Программы и их графы алгоритма

$s = 0$ (1)

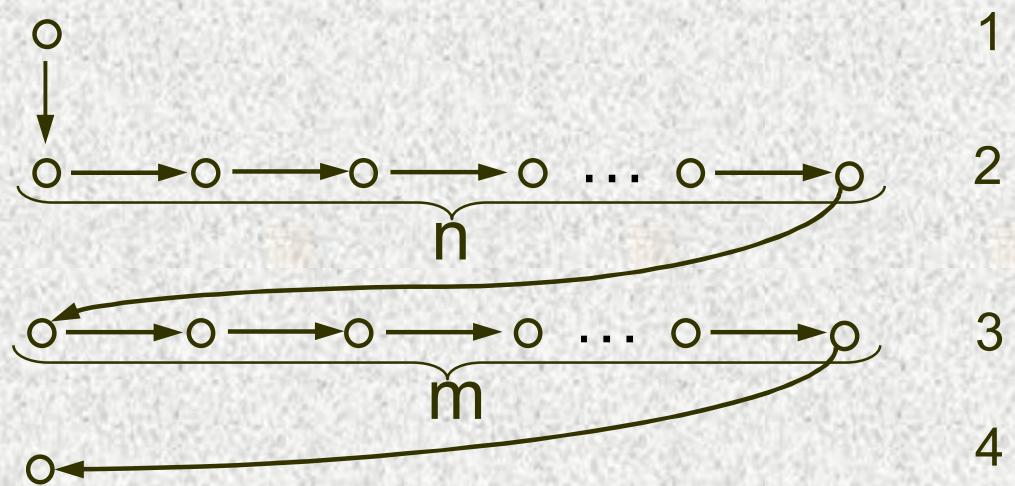
Do $i = 1, n$

$s = s + 1$ (2)

Do $i = 1, m$

$s = s + 1$ (3)

$s = s + 1$ (4)



$$\begin{cases} m \geq 1 \\ j_1 = m \\ u3 3 \end{cases}$$

$$\begin{cases} m < 1 \\ n \geq 1 \\ j_1 = n \\ u3 2 \end{cases}$$

$$\begin{cases} m < 1 \\ n < 1 \\ u3 1 \end{cases}$$

Программы и их графы алгоритма (умножение матриц)

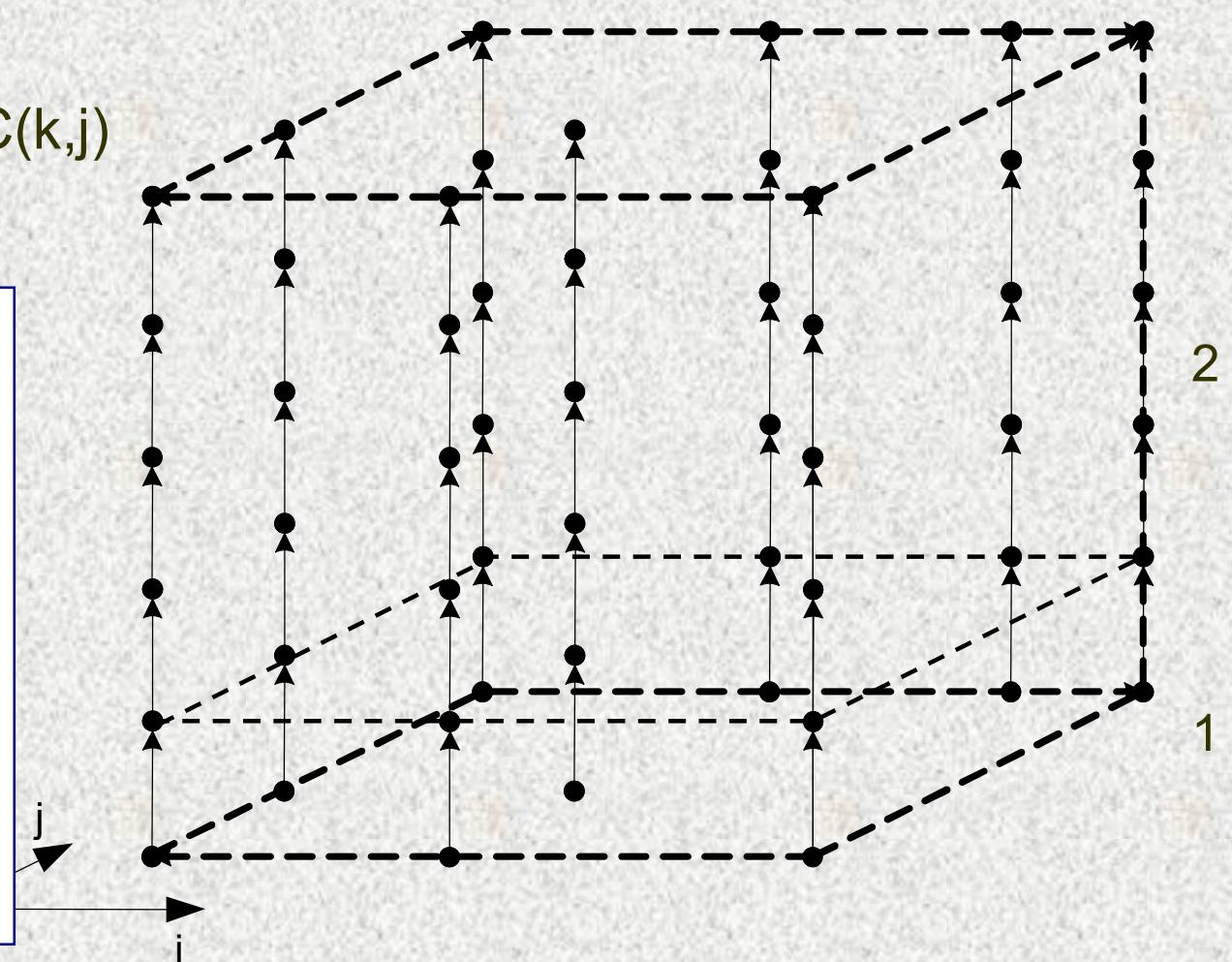
```

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

1

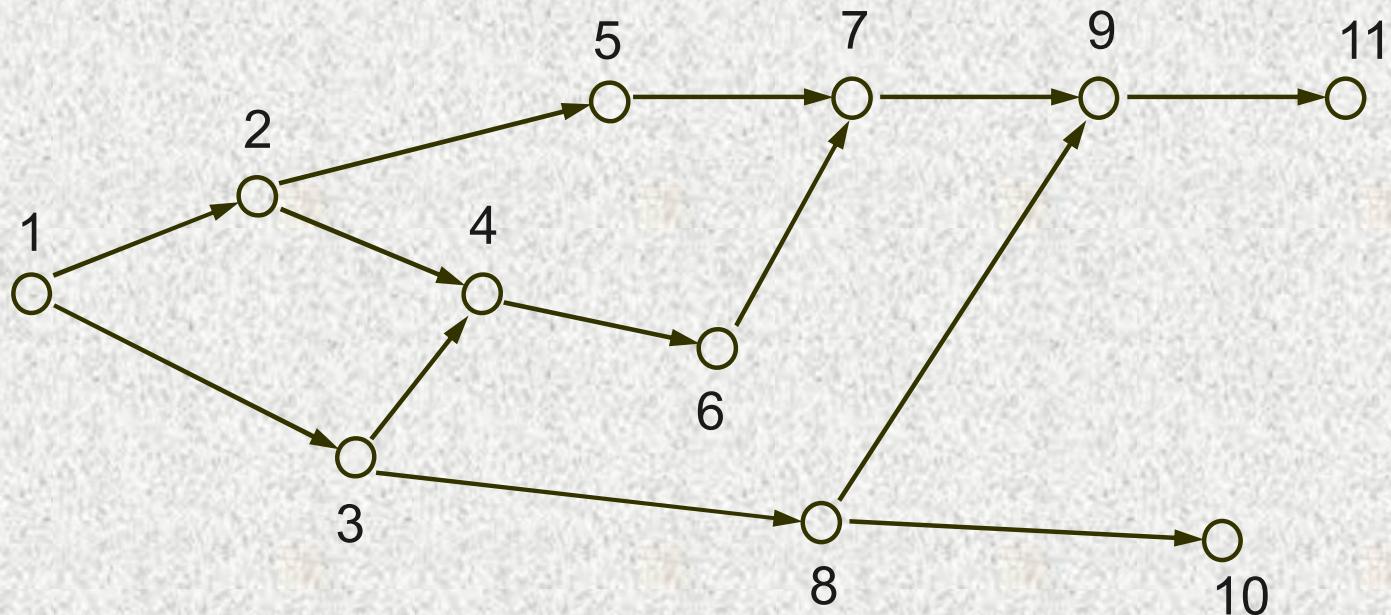
2

$$\begin{cases}
 \begin{cases}
 1 \leq i \leq n \\
 1 \leq j \leq n \\
 k = 1 \\
 \begin{cases}
 i_1 = i \\
 j_1 = j
 \end{cases} \\
 u_3(1)
 \end{cases} \\
 \begin{cases}
 1 \leq i \leq n \\
 1 \leq j \leq n \\
 2 \leq k \leq n \\
 \begin{cases}
 i_1 = i \\
 j_1 = j \\
 k_1 = k - 1
 \end{cases} \\
 u_3(2)
 \end{cases}
 \end{cases}$$



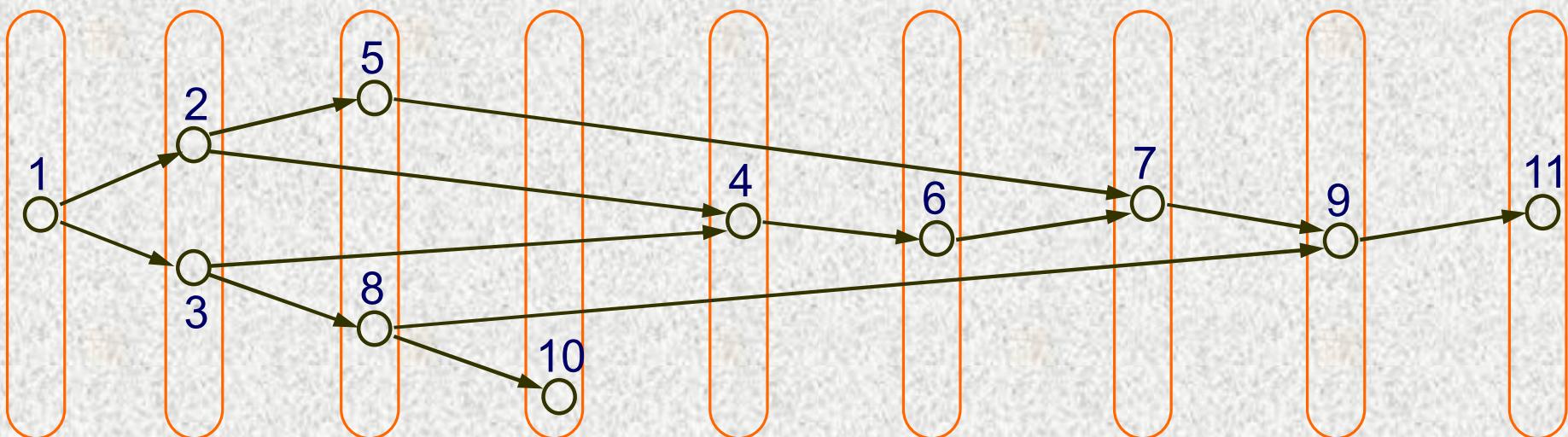
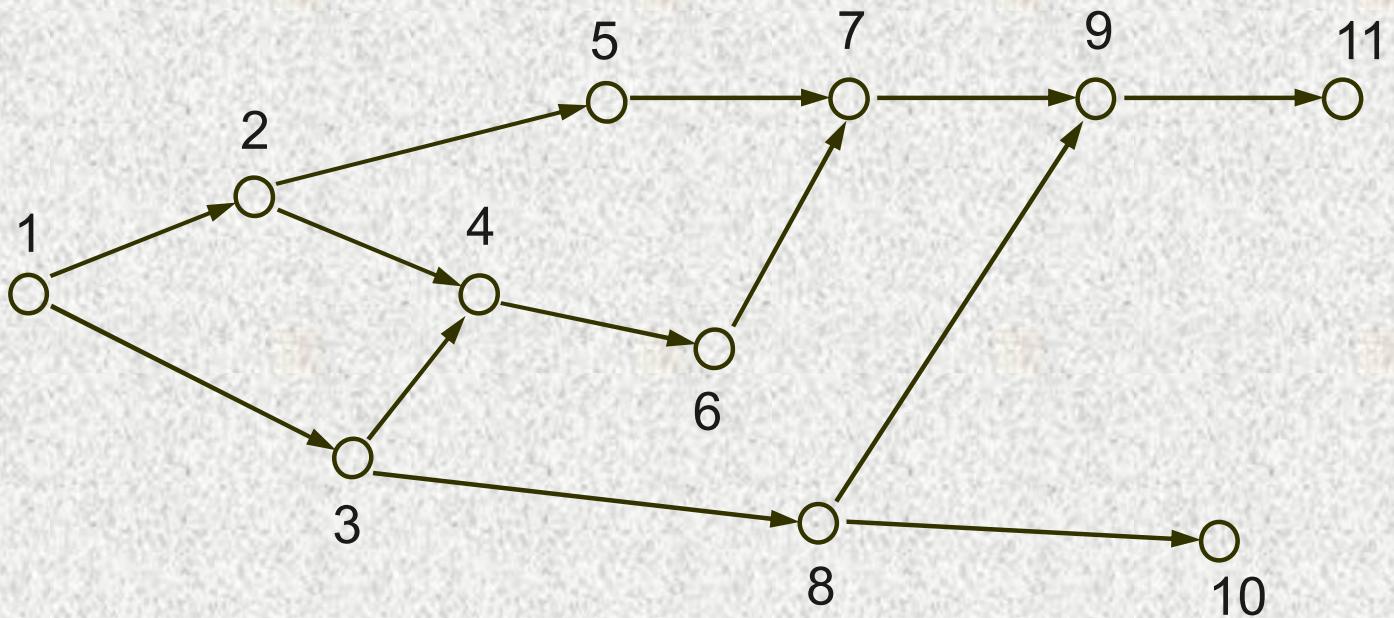
*Как описать ресурс параллелизма
программ и алгоритмов?*

Ярусно-параллельная форма графа алгоритма

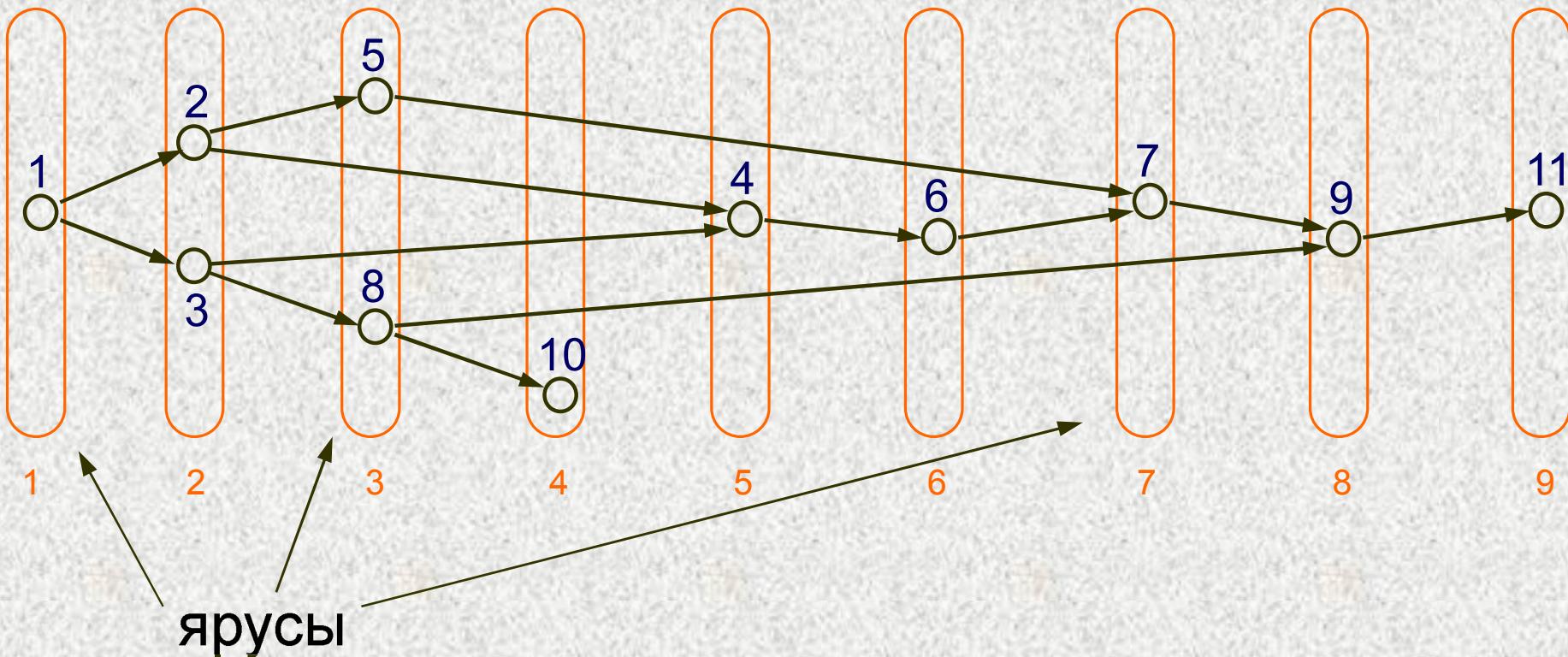


Как определить и сделать понятным ресурс
параллелизма в графе алгоритма (в программе, в
алгоритме) ?

Ярусно-параллельная форма графа алгоритма

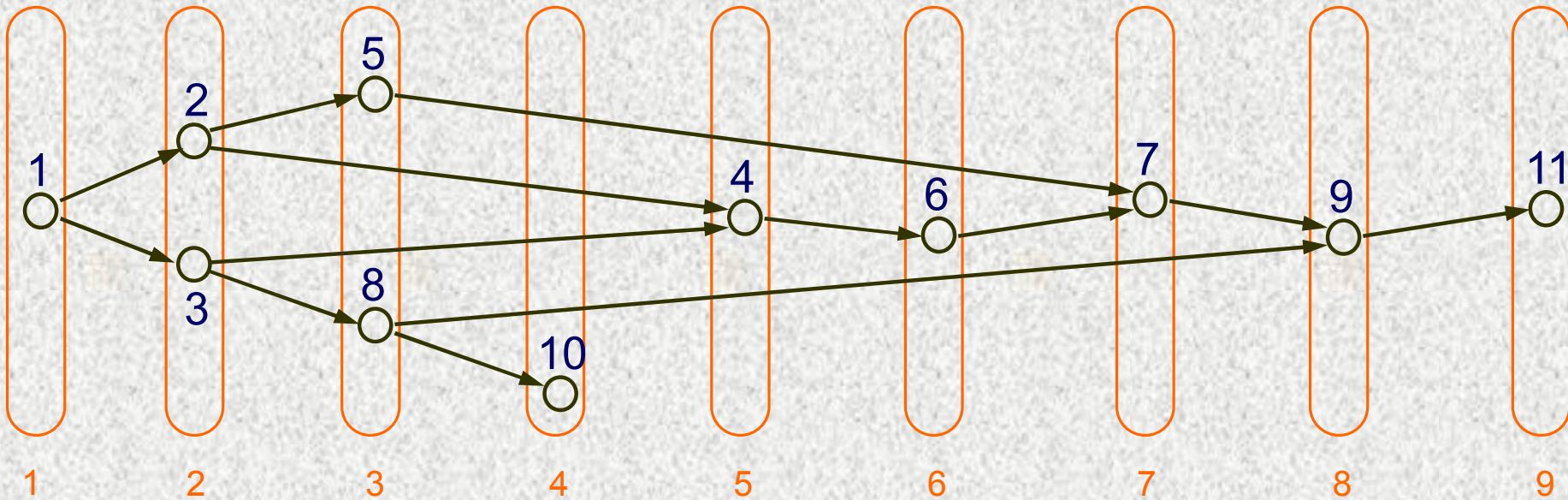


Ярусно-параллельная форма графа алгоритма



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

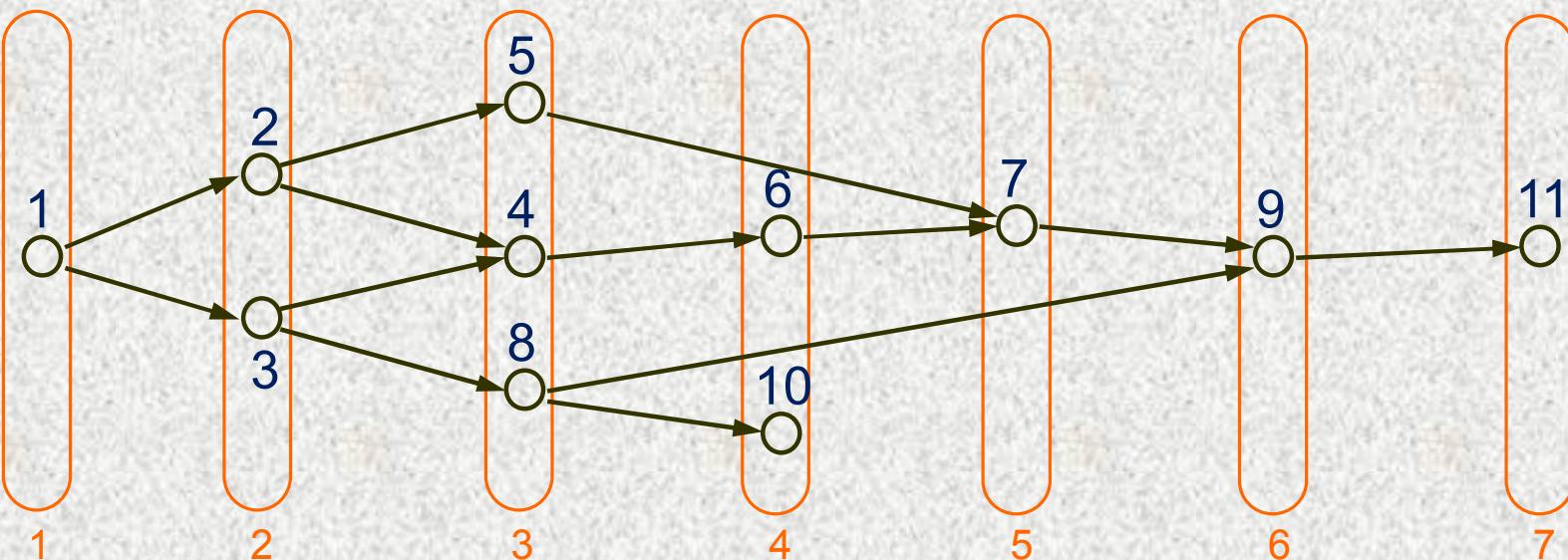
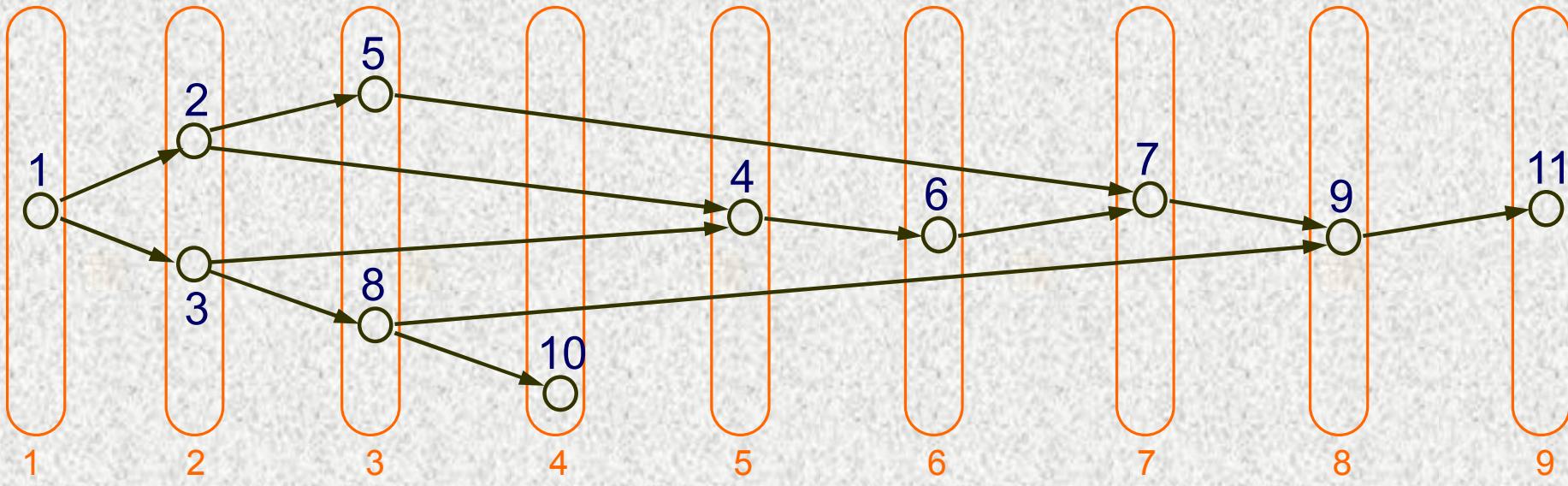
Ярусно-параллельная форма графа алгоритма



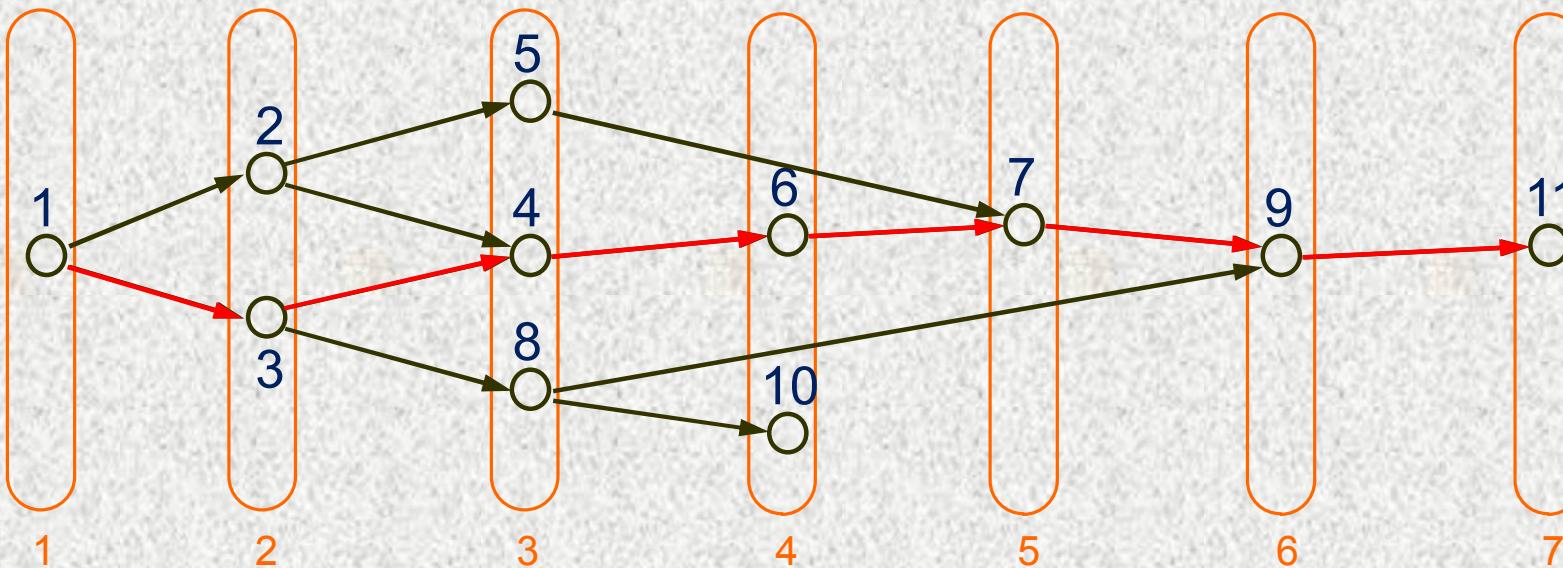
Высота ЯПФ – это число ярусов,
Ширина яруса – число вершин, расположенных на ярусе,
Ширина ЯПФ – это максимальная ширина ярусов в ЯПФ.

Высота ЯПФ = сложность параллельной реализации
алгоритма/программы.

Ярусно-параллельная форма графа алгоритма определяется неоднозначно



Каноническая ярусно-параллельная форма графа алгоритма



Высота канонической ЯПФ = длине критического пути + 1.

Каноническая ЯПФ и степень параллелизма

```
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.

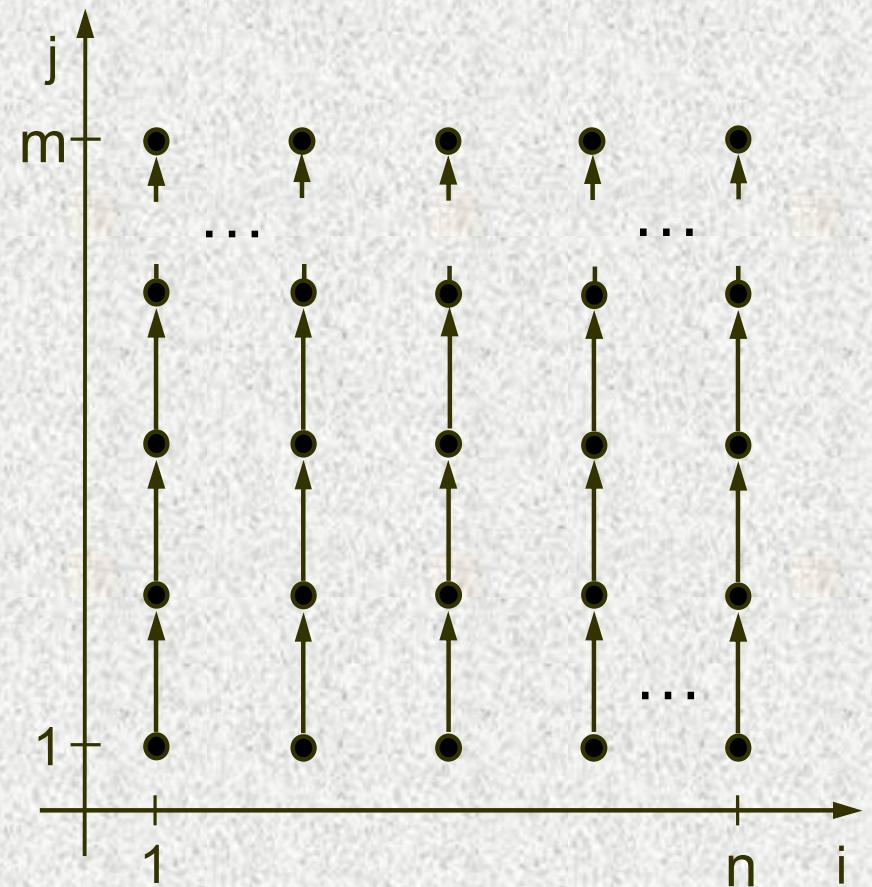
Каноническая ЯПФ и степень параллелизма

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

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

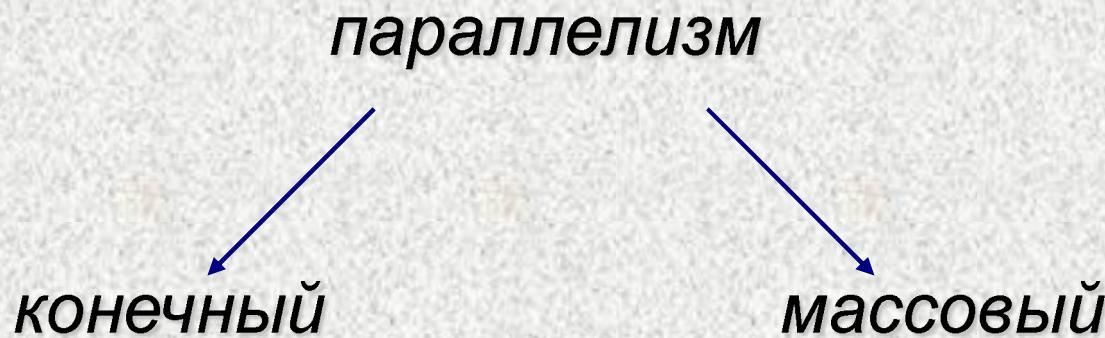
$$\alpha = \frac{\text{Число последовательных операций}}{\text{Общее число операций}} = \frac{m}{n*m} = \frac{1}{n}$$

$$S \approx n$$



*Какого рода параллелизм
встречается в программах?*

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.

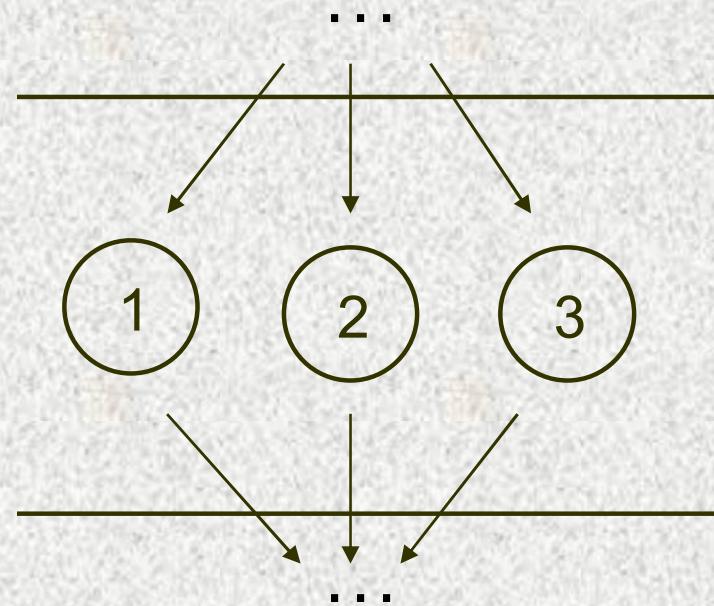
```
cout << "N=" << N << endl;
cycleTestWithUnroll_KJI("k");
cycleTestWithUnroll_KJI("j");
cycleTestWithUnroll_KJI("i");
cycleTestWithUnroll_KJI_3("k");
cycleTestWithUnroll_KJI_3("j");
cycleTestWithUnroll_KJI_3("i");
cycleTest("j,i,k");
cycleTest("i,k,j");
cycleTest("k,j,i");
cycleTest("i,j,k");
cycleTest("k,i,j");
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][j][i] + P[i][j]*A[k][j][i-1] +
                P[i][k]*A[k][j-1][i] + P[j][k]*A[k-1][j][i];
        }
...

```

1

2

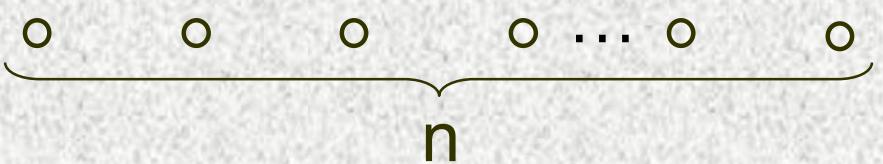
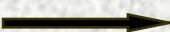
3



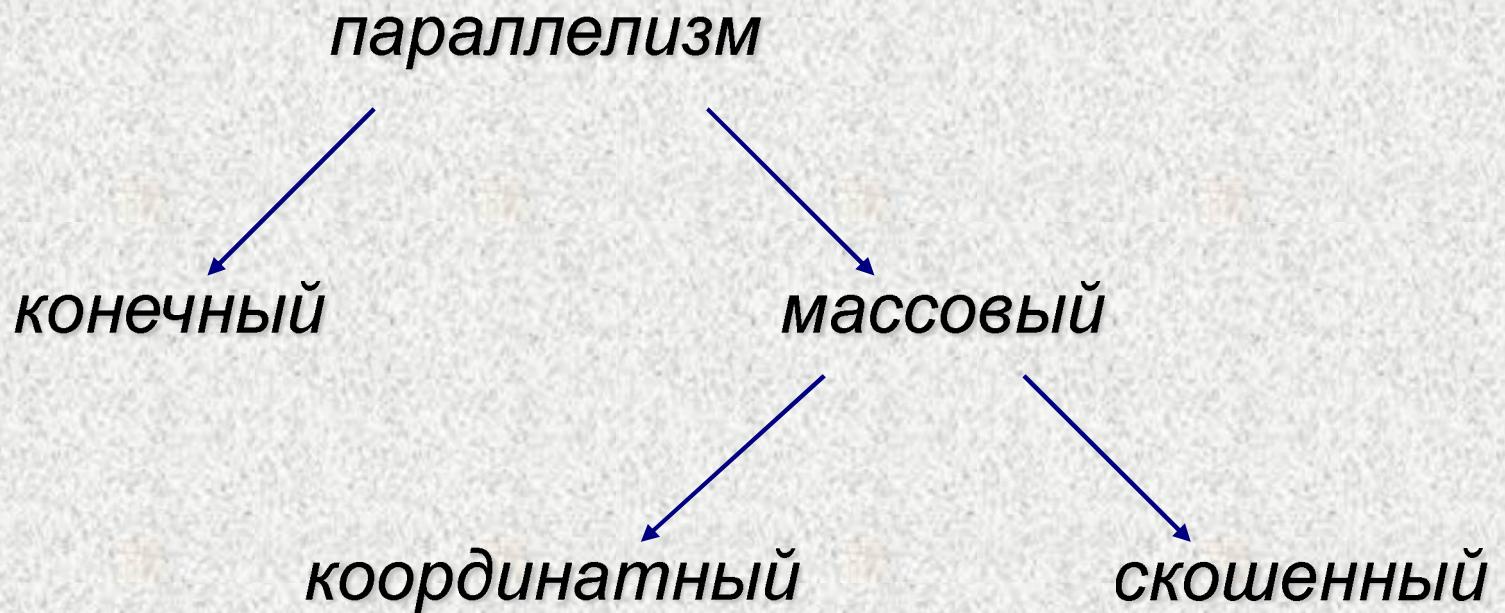
Виды параллелизма в алгоритмах и программах

Массовый параллелизм.

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



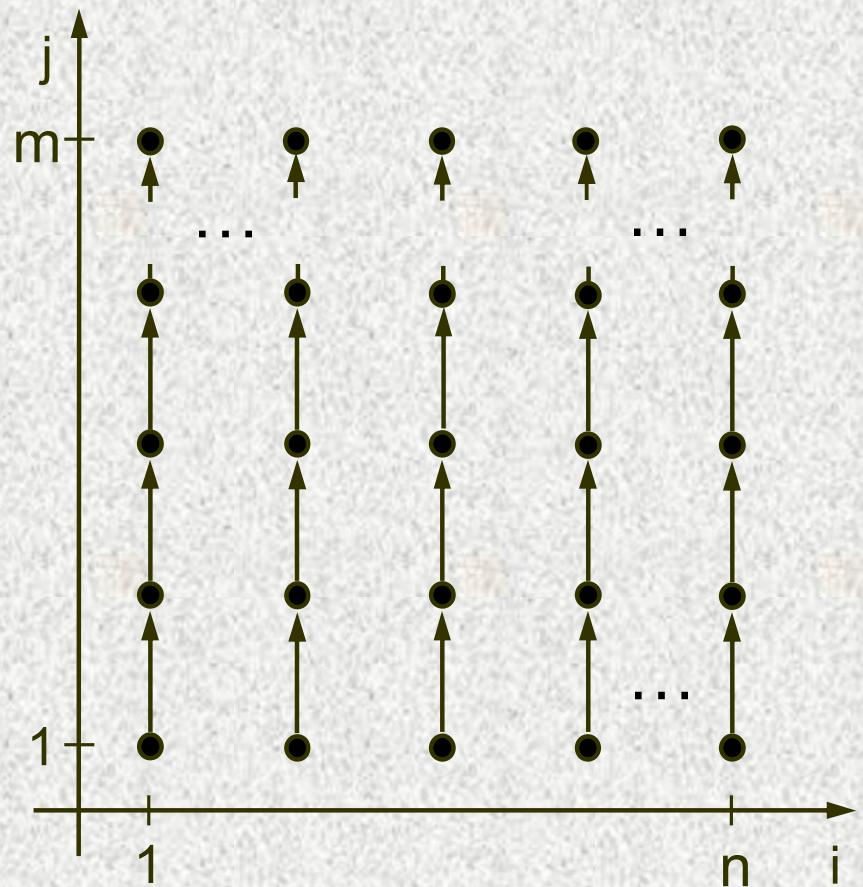
Виды параллелизма в алгоритмах и программах



Виды параллелизма в алгоритмах и программах

Координатный параллелизм.

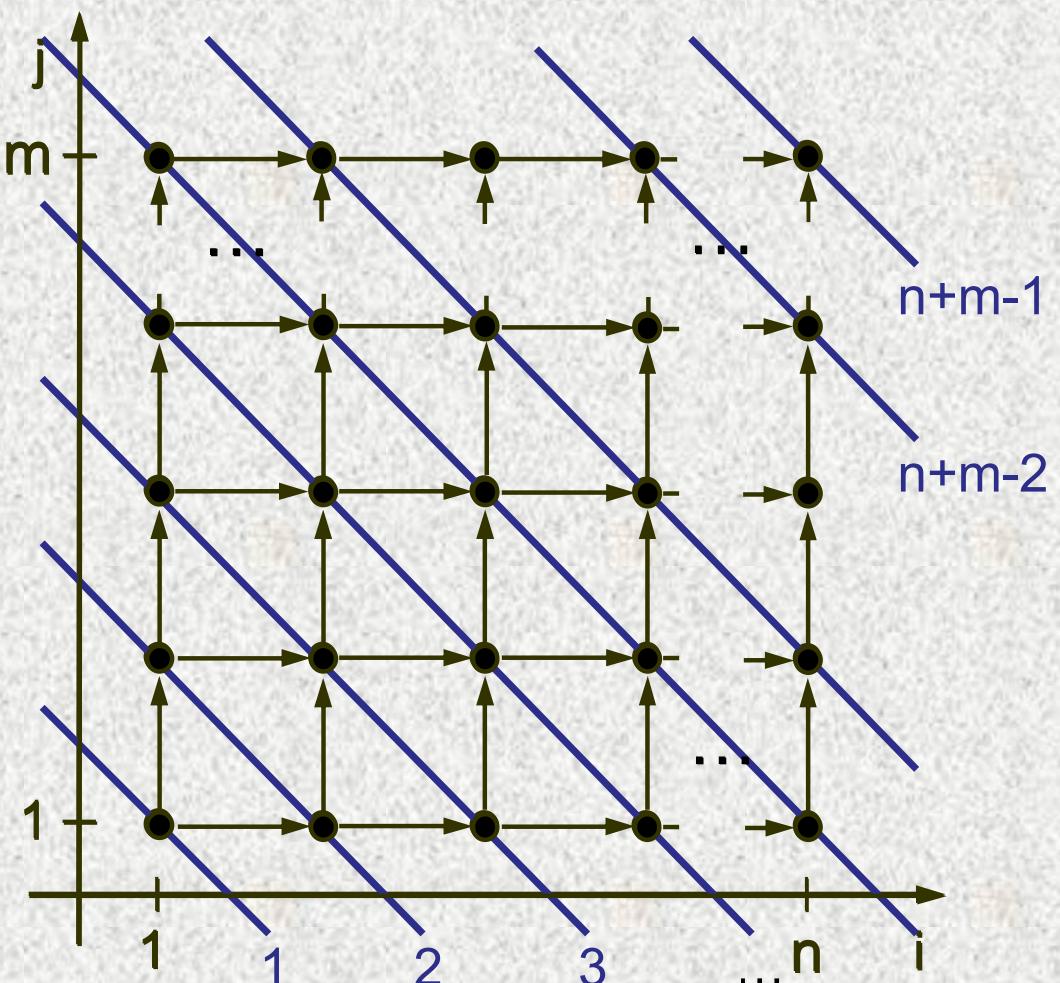
```
#pragma omp parallel for
for( i = 0; i < n; ++i)
    for( j = 0; j < m; ++j)
        A[i][j] = A[i][j-1] + C[i][j]*x;
```



Виды параллелизма в алгоритмах и программах

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

Преобразованная программа

Построение
графа алгоритма

Исследование
графа алгоритма

Преобразование
графа алгоритма



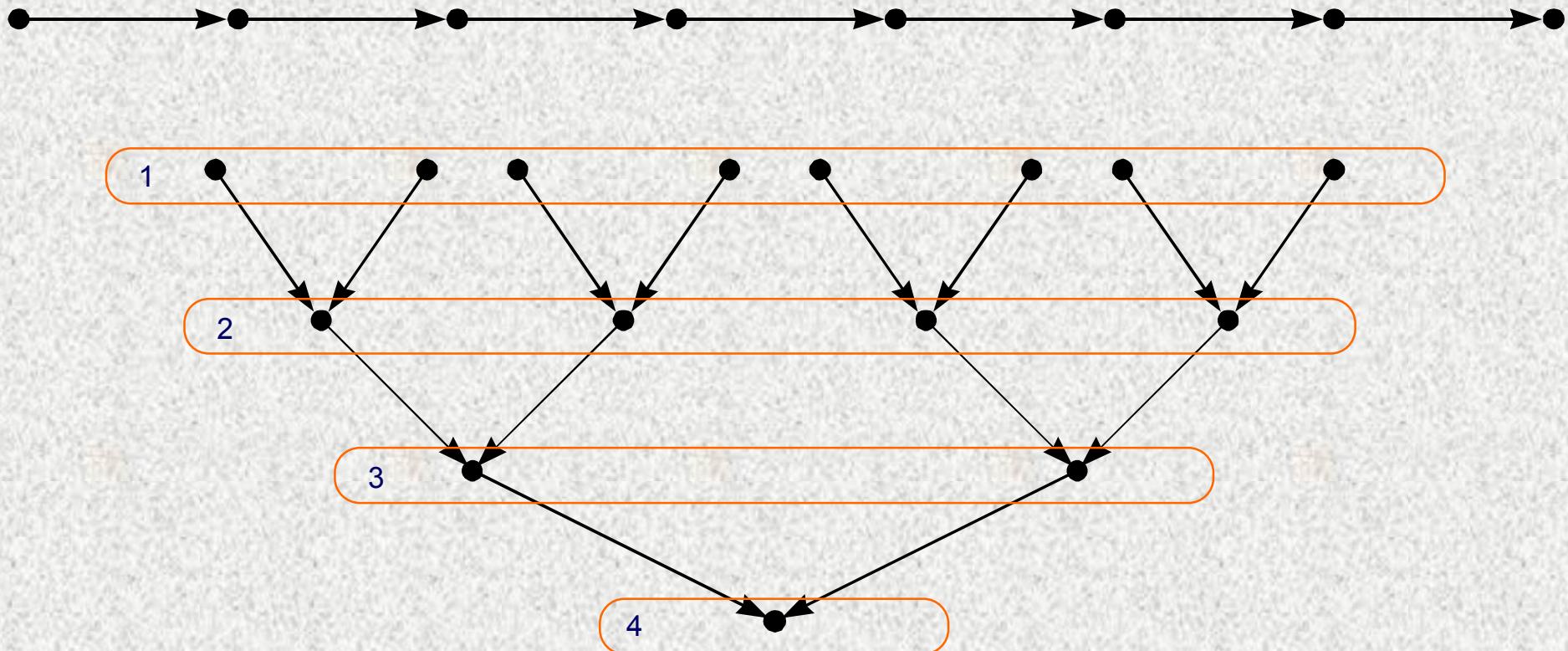
Эквивалентные преобразования программ (суммирование элементов массива)

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



Чисто последовательный алгоритм. Что делать? Не использовать суммирования на параллельных вычислительных системах? Невозможно...

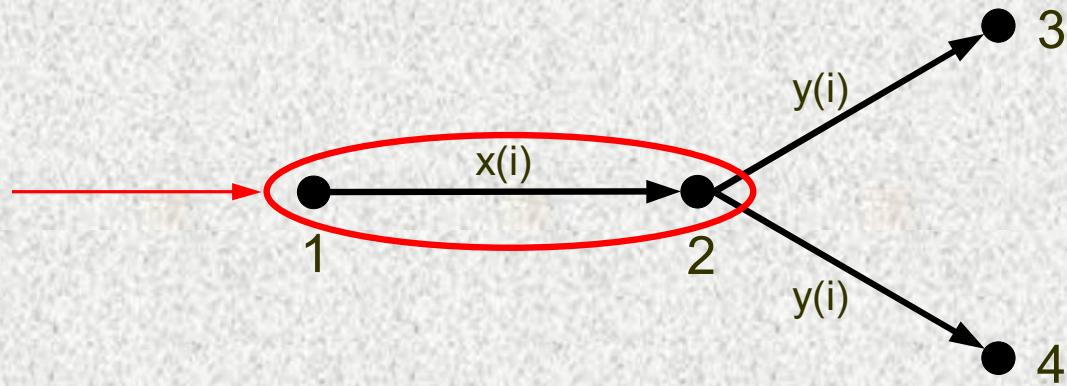
Эквивалентные преобразования программ (суммирование элементов массива)



В данном случае это не есть эквивалентное преобразование! Использовано два разных метода с разной информационной структурой, разной параллельной сложностью, разными ошибками округления...

Эквивалентные преобразования программ

Исполнять только
последовательно !



Информационная зависимость определяет критерий эквивалентности преобразований программ.

Информационная независимость определяет ресурс параллелизма программы.