Московский государственный университет имени М.В.Ломоносова
факультет Вычислительной математики и кибернетики

СУПЕРКОМПЬЮТЕРЫ
И ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ

ВВЕДЕНИЕ В ТЕОРИЮ АНАЛИЗА СТРУКТУРЫ ПРОГРАММ И АЛГОРИТМОВ

Вл.В.Воеводин

Директор НИВЦ МГУ,
Директор филиала МГУ в г.Сарове,
Зав.кафедрой Суперкомпьютеров и квантовой информатики ВМК МГУ,
чл.-корр. РАН, д.ф.-м.н., профессор

voevodin@parallel.ru

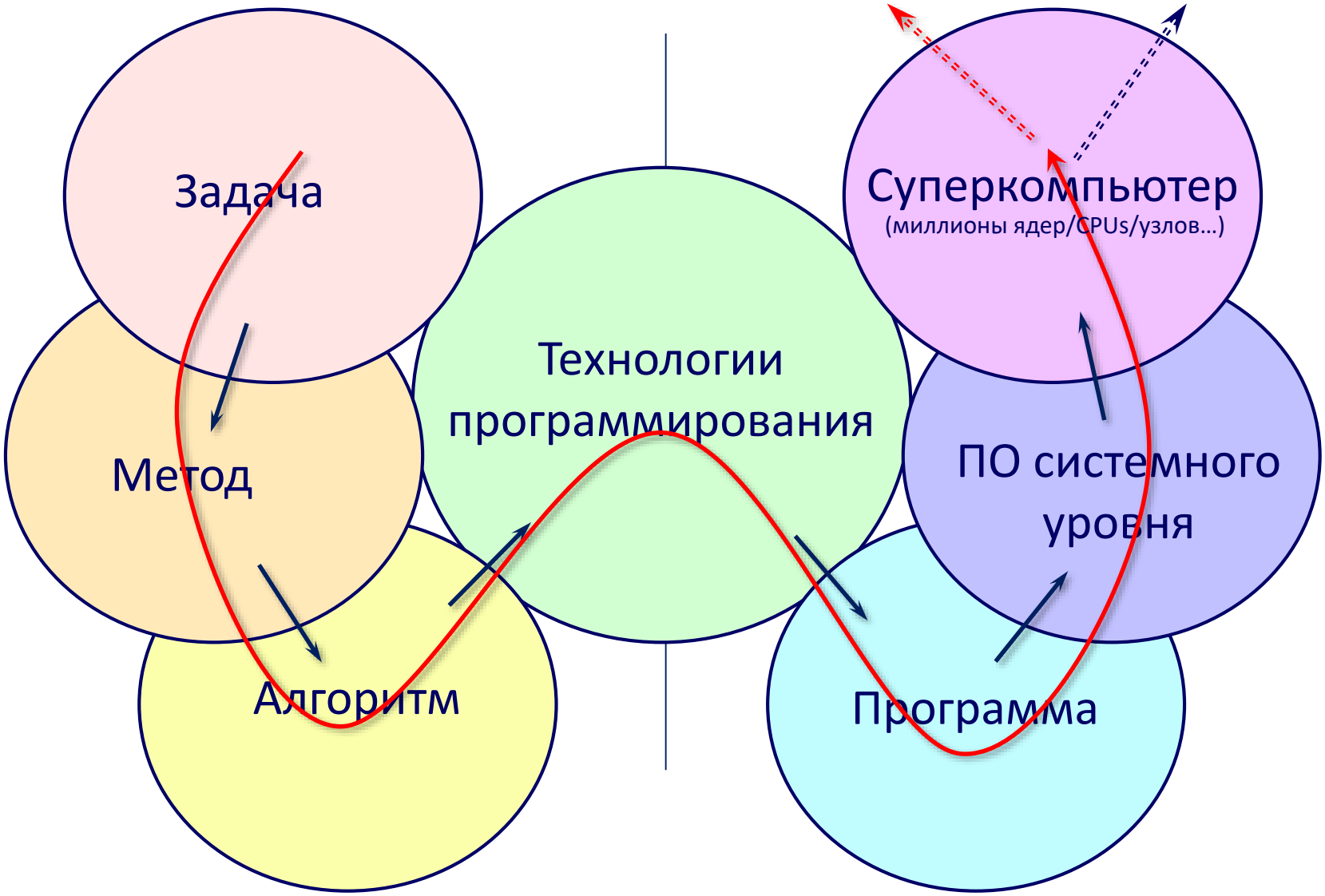
ВМК МГУ, 2025

Решение задач и вычислительные системы

(Реальность)
Реальная производительность

Огромный разрыв !

(Ожидания)
Пиковая производительность



Алгоритмическая сторона

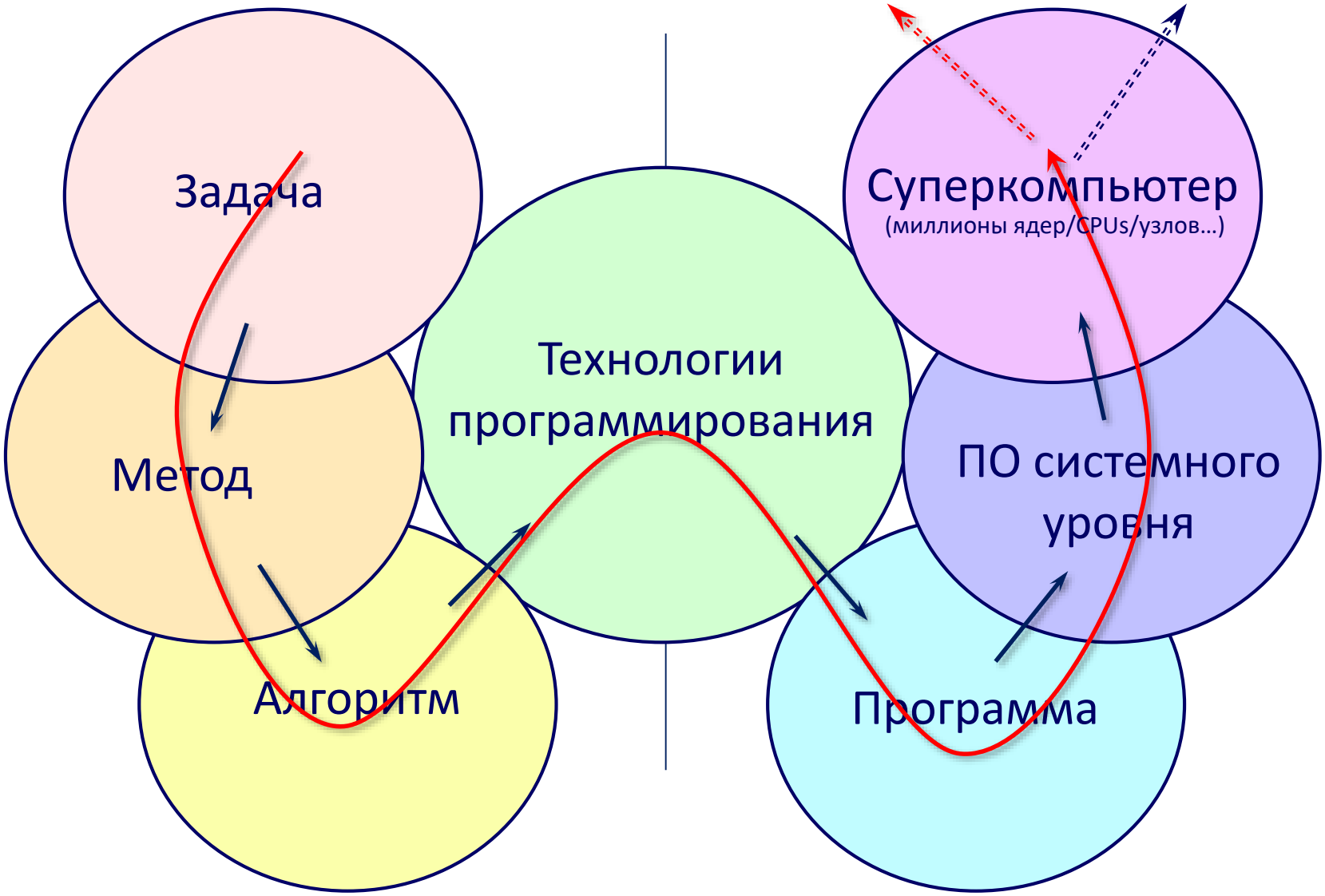
Компьютерная сторона

Решение задач и вычислительные системы

(Реальность)
Реальная производительность

Огромный разрыв !

(Ожидания)
Пиковая производительность



Алгоритмическая сторона

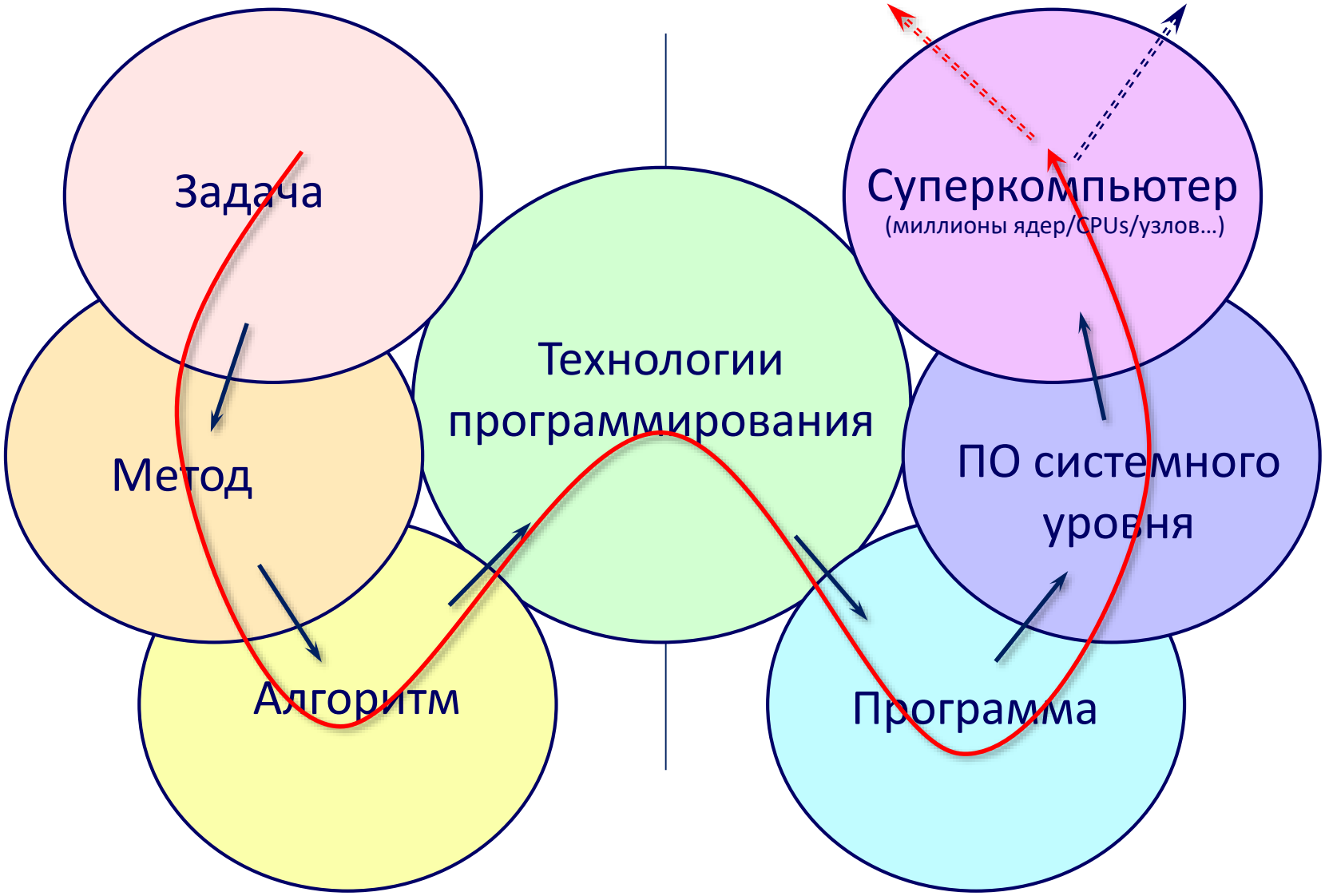
Компьютерная сторона

Суперкомпьютерный кодизайн

(Реальность)
Реальная производительность

Огромный разрыв !

(Ожидания)
Пиковая производительность



Алгоритмическая сторона

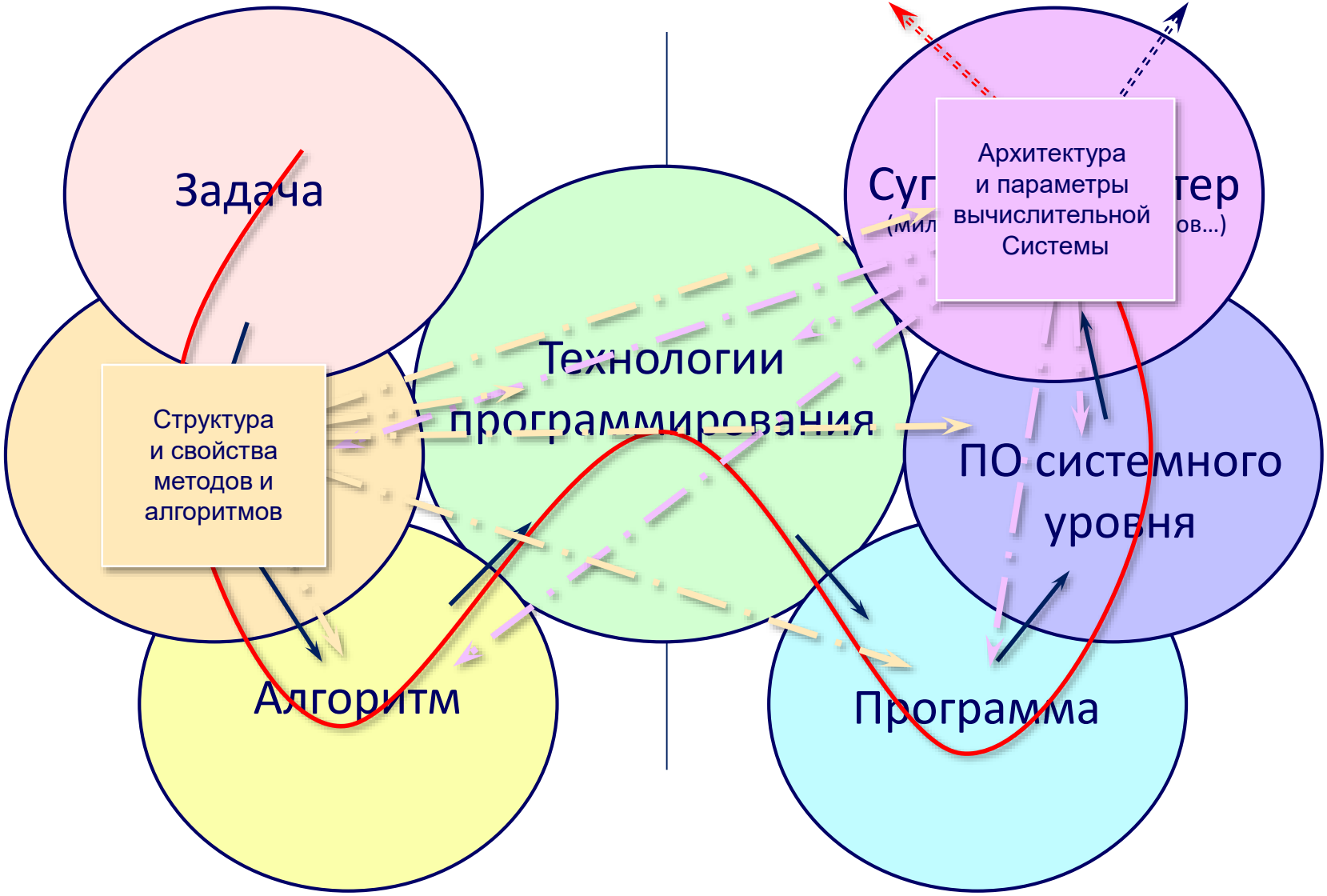
Компьютерная сторона

Суперкомпьютерный кодизайн

(Реальность)
Реальная производительность

Огромный разрыв !

(Ожидания)
Пиковая производительность



Алгоритмическая сторона

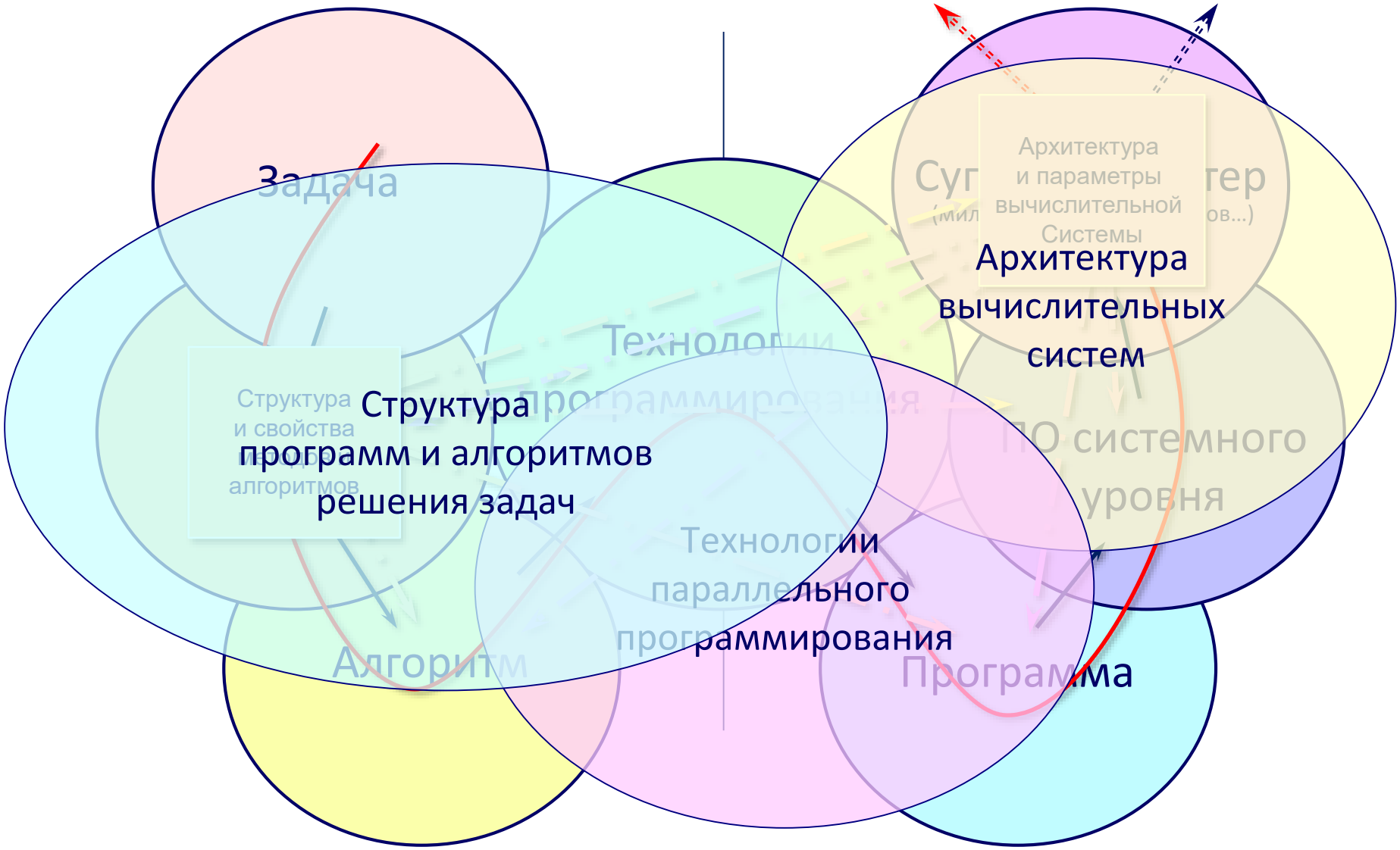
Компьютерная сторона

Суперкомпьютерный кодизайн

(Реальность)
Реальная производительность

Огромный разрыв !

(Ожидания)
Пиковая производительность



Алгоритмическая сторона

Компьютерная сторона

Почему важно понимать как написаны программы?

Pascal? Fortran? C? C++?

Пользователь: почему?

$A_{ijk} = A_{i-1jk} + B_{jk} + B_{jk}$, i=1,40; j=1,40; 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?

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

Почему важно понимать как написаны программы?

Pascal? Fortran? C? C++?

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]
```

Порядок циклов: (i, j, k)

Возможен ли порядок:

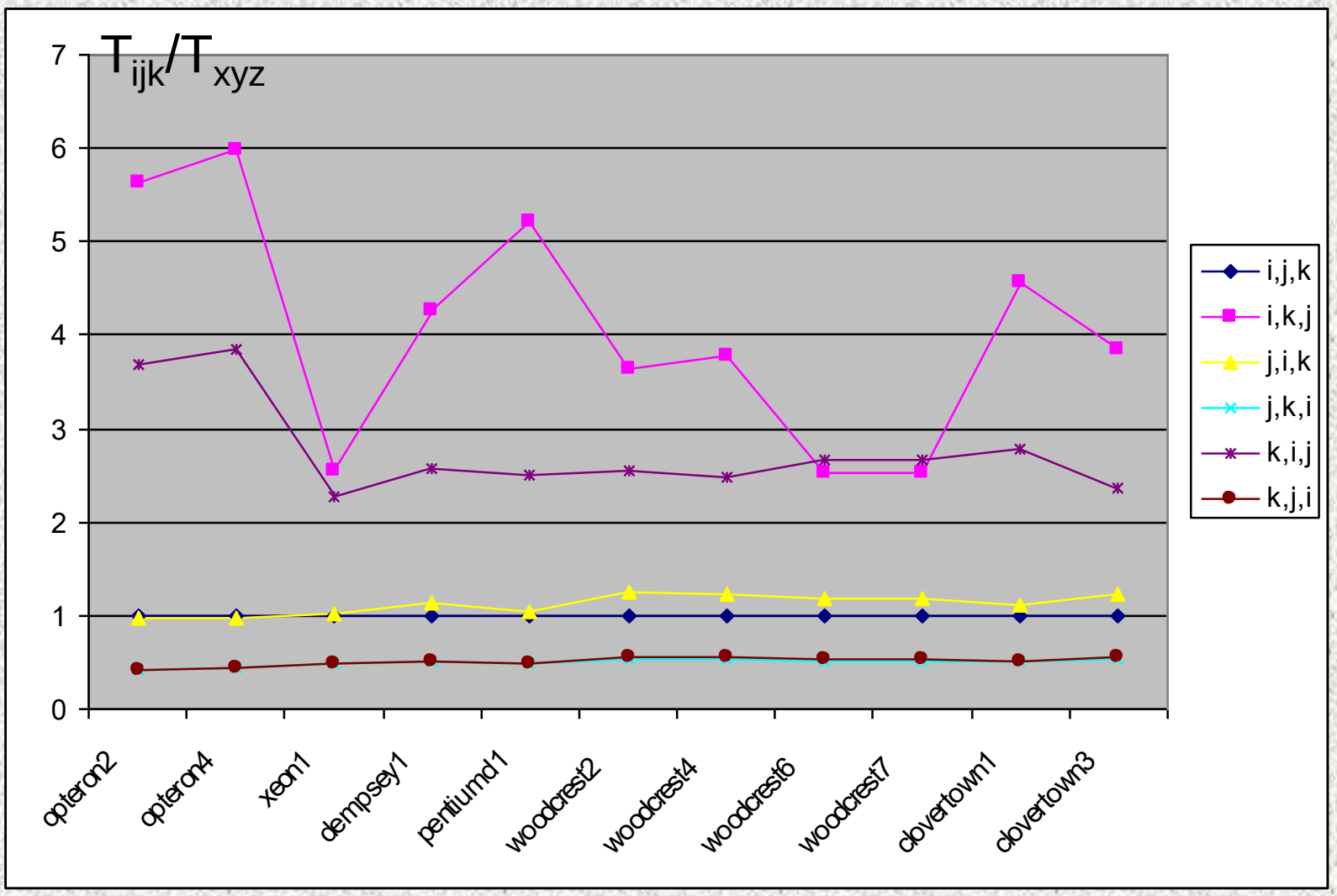
(i, k, j) - ?	ДА
(k, i, j) - ?	ДА
(k, j, i) - ?	ДА
(j, i, k) - ?	ДА
(j, k, i) - ?	ДА

Почему возможен другой порядок?

А зачем нужен другой порядок?

Умножение матриц: все ли просто?

(сравнение с порядком (i, j, k))

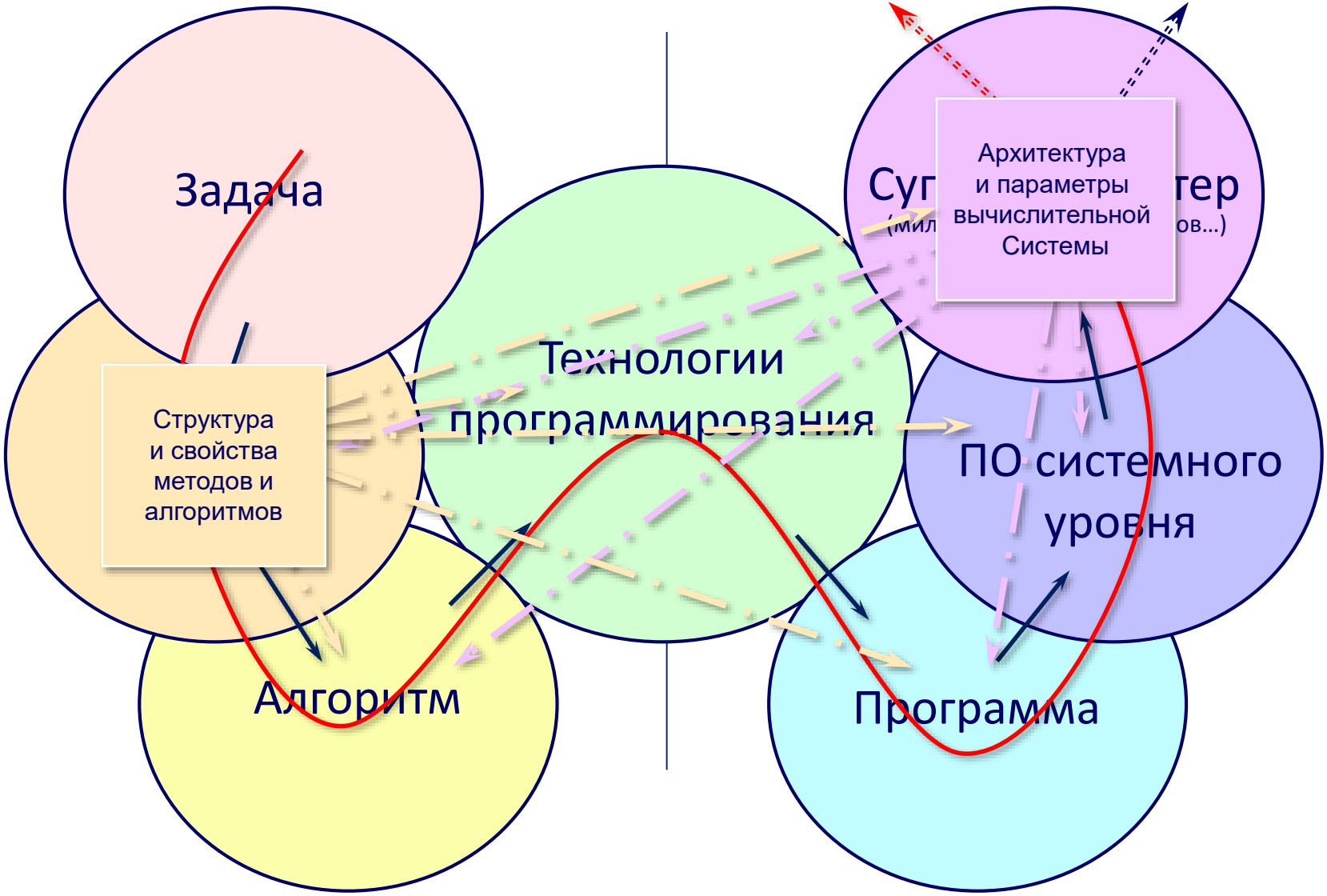


Суперкомпьютерный кодизайн

(Реальность)
Реальная производительность

Огромный разрыв !

(Ожидания)
Пиковая производительность



Алгоритмическая сторона

Компьютерная сторона

Графовые модели программ

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

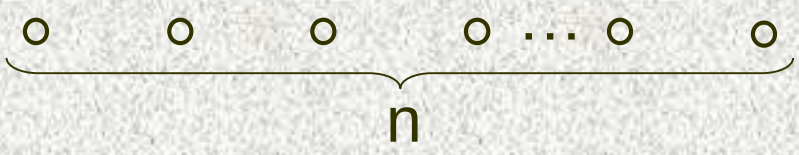
Вершины: процедуры, циклы, линейные участки, операторы, итерации циклов, срабатывания операторов…

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];
}
```



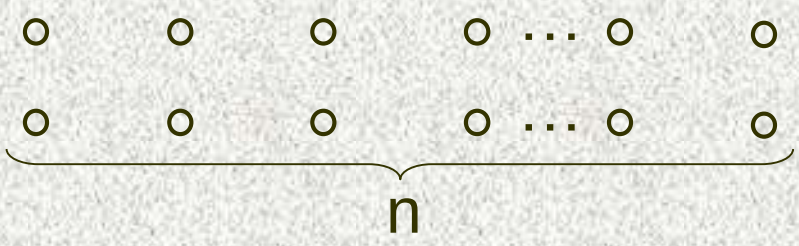
Каждая вершина соответствует двум операторам (телу цикла), выполненным на одной и той же итерации цикла.

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) (1)
y(i) = 2*x(i) – 3 (2)
t1 = y(i)*y(i) + 1 (3)
t2 = b(i) – y(i)*a (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.

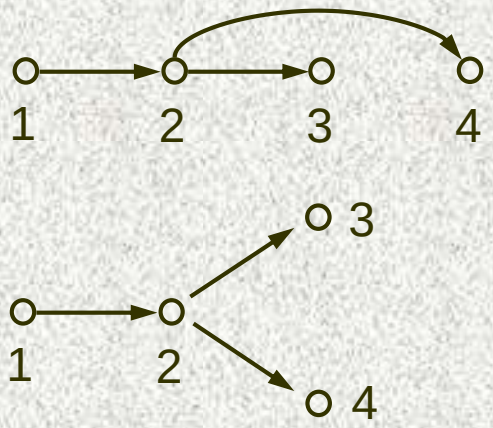
Дуги: информационное отношение:

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

$y(i) = 2*x(i) – 3$ (2)

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

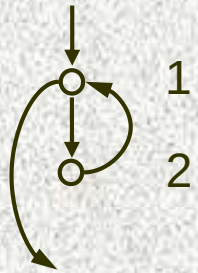
$t2 = b(i) – y(i)*a$ (4)



Четыре основные модели программ

Граф управления программы.
Вершины: операторы
Дуги: операционное отношение

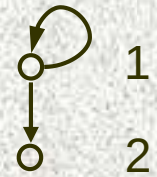
```
for( i = 0; i < n; ++i) {
    A[i] = A[i – 1] + 2;          (1)
    B[i] = B[i] + A[i];           (2)
}
```



Четыре основные модели программ

Информационный граф программы.
Вершины: операторы
Дуги: информационное отношение

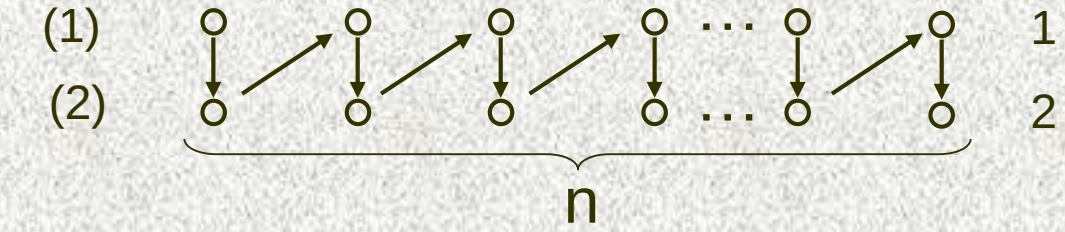
```
for( i = 0; i < n; ++i) {
    A[i] = A[i – 1] + 2;        (1)
    B[i] = B[i] + A[i];         (2)
}
```



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];
}
```

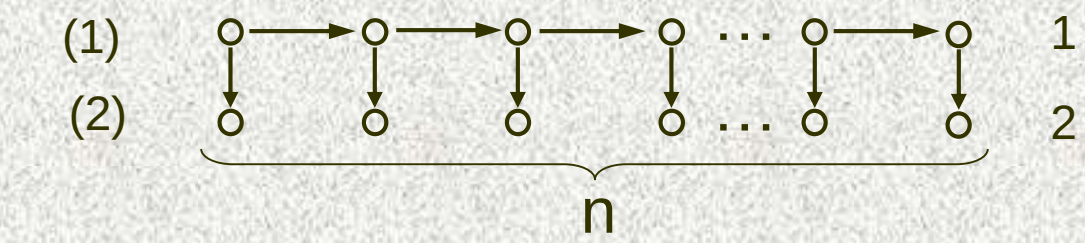


Свойства операционной истории:
- одна начальная вершина, у которой нет входящей дуги,
- одна конечная вершина, у которой нет исходящей дуги,
- у всех остальных вершин есть ровно одна входящая дуга и одна исходящая дуга.

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];
}
```



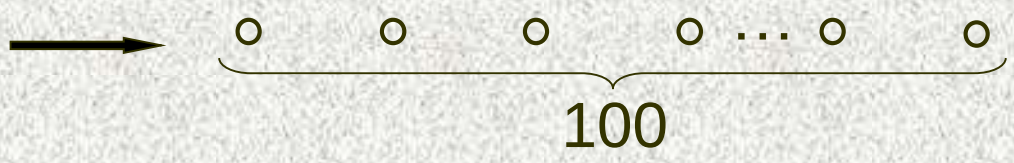
Свойства информационной истории:
- ациклический граф,
- нет кратных дуг.

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;
```

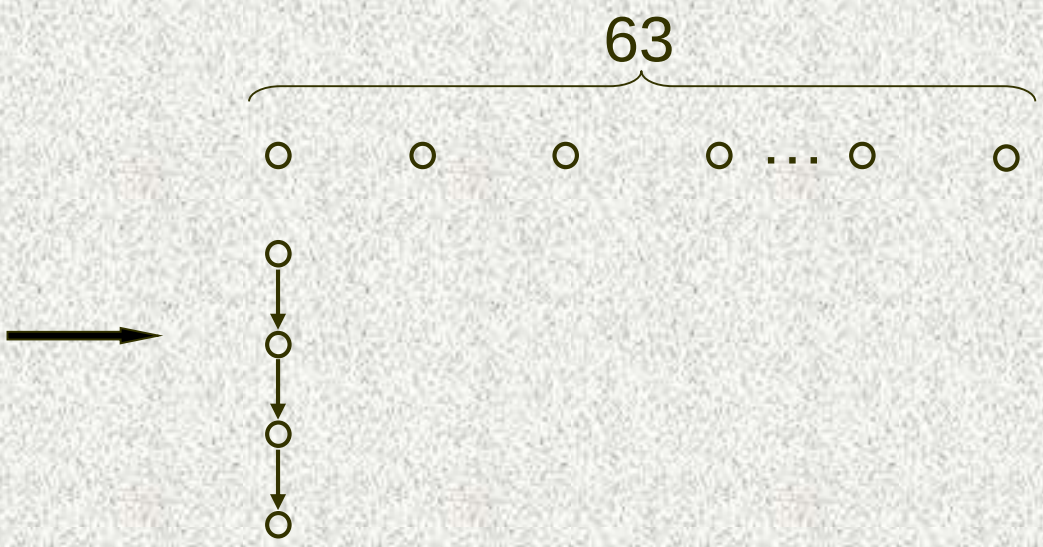


Несколько вопросов…

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

ДА.

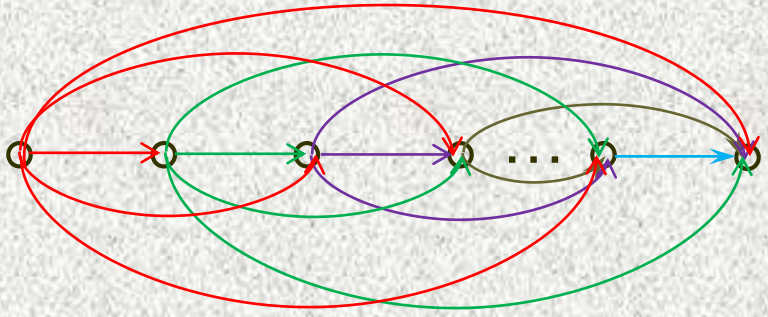
```
for( i = 0; i < 63; ++i)
    A[i] = B[i] + C[i]*x;
x1 = 10;
x2 = x1+1;
x3 = x2+2;
x4 = x3+3;
```



Несколько вопросов…

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

Вспомним свойства информационной истории:
- ациклический граф,
- нет кратных дуг.



Макс.число дуг: (n-1) + (n-2) + (n-3) + … + 2 + 1 = n(n-1)/2*

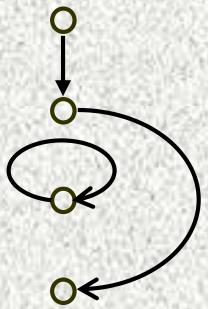
Ответ: НЕТ

Несколько вопросов…

Может ли граф управления некоторого фрагмента программы состоять из нескольких компонент связности?

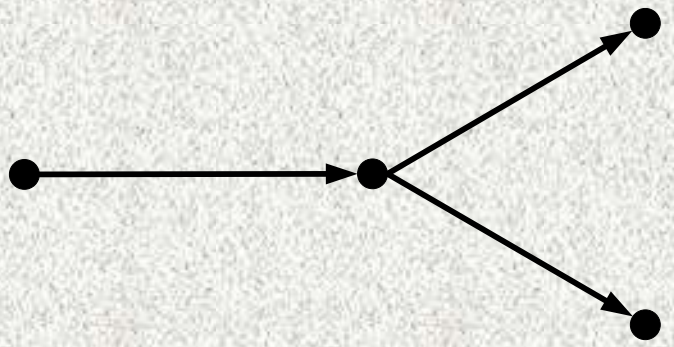
ДА.

```
x1 = 10;
x2 = x1+1; goto A;
B: x3 = x2+2; goto B;
A: x4 = x2+3;
```



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)

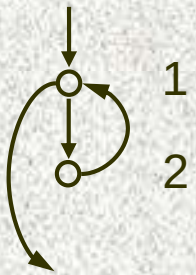
Операционные модели

Информационные модели

Компактные модели

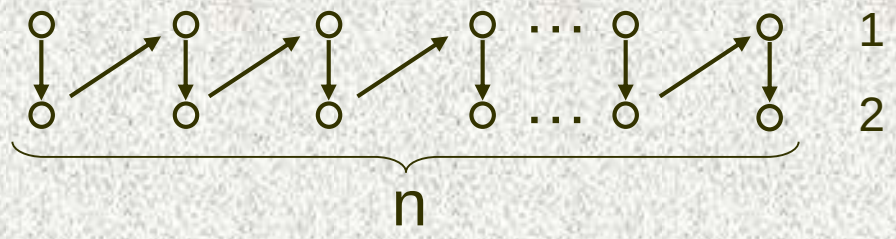
Истории

Граф управления

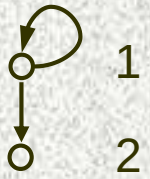


…

Операционная история

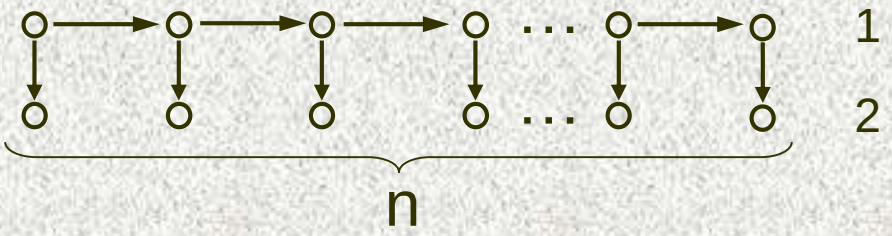


Информационный граф



…

Информационная история



Какое отношение выбрать для описания свойств программ?

Операционное отношение?

x(i) = a + b(i) (1)
y(i) = 2*x(i) – 3 (2)
t1 = y(i)*y(i) + 1 (3)
t2 = b(i) – y(i)*a (4)

Какое отношение выбрать для описания свойств программ?

Информационная структура *– это основа анализа свойств программ и алгоритмов.*

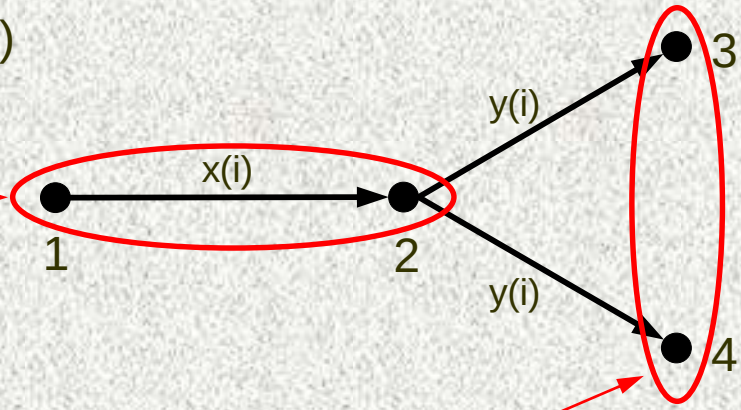
$$x(i) = a + b(i) \quad (1)$$

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

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

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

Исполнять только последовательно !

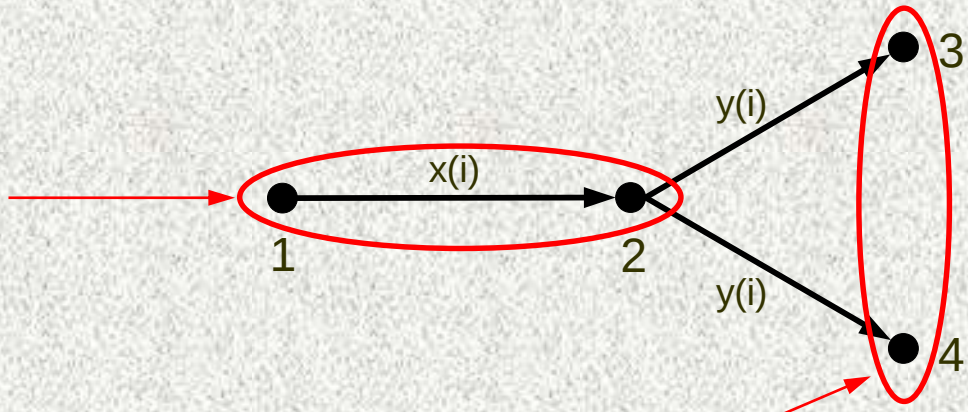


Можно исполнять параллельно !

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?

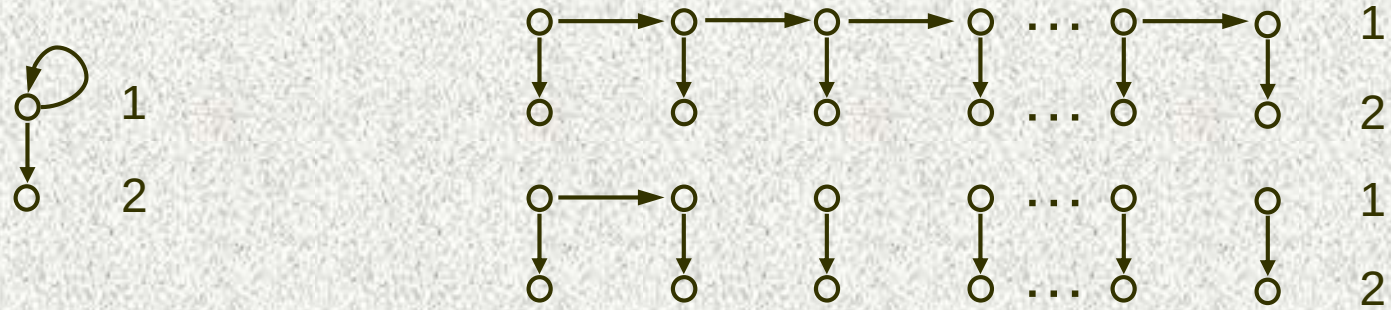
Аргументы для выбора степени компактности модели:

- *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?

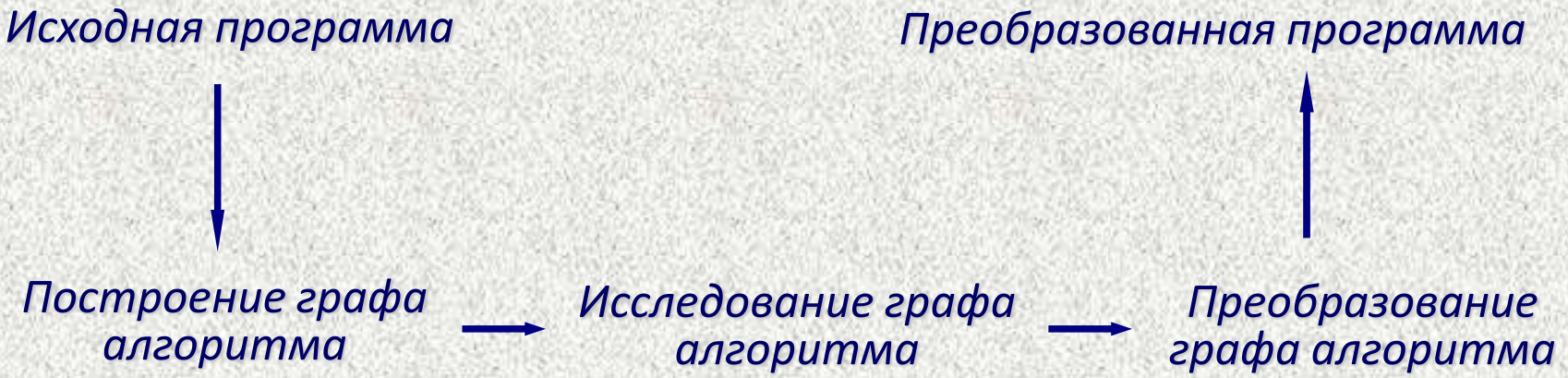
Аргументы для выбора степени компактности модели:

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

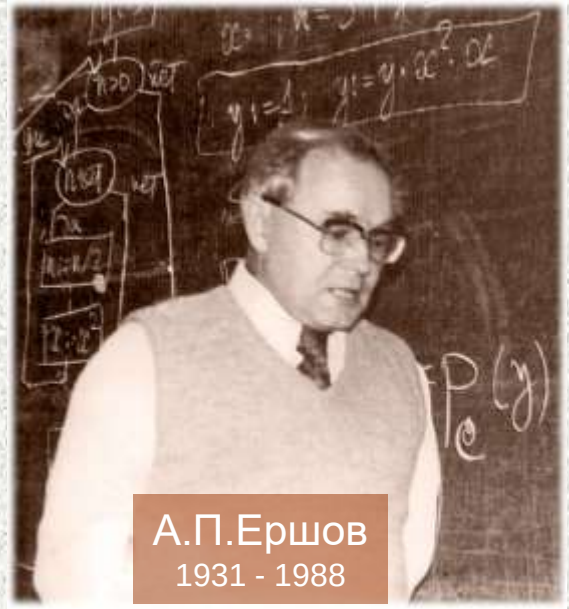
Граф алгоритма – это параметризованная информационная история:

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

Общая схема анализа и преобразования структуры программ



Основатели теории анализа структуры программ и алгоритмов



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 распределенной 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.

Теорема о построении графа алгоритма

<u>Теорема.</u> 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.

Теорема о построении графа алгоритма

(…простыми словами…)

<u>Теорема.</u> 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,
Δ(N) – линейный выпуклый многогранник в пространстве итераций фрагмента,
F(Δ, N) – линейная векторная функция, описывающая входящие дуги.

аргументы операторов

входные данные программы

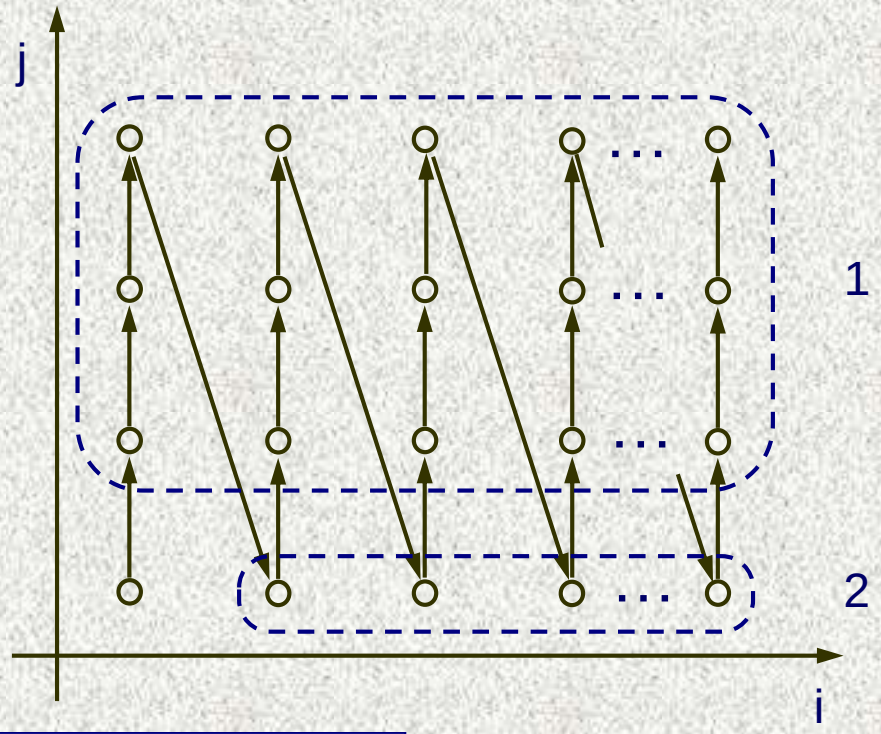
срабатывания операторов

информационное отношение

Как выполняется описание структуры программ на практике?

Программы и их графы алгоритма

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



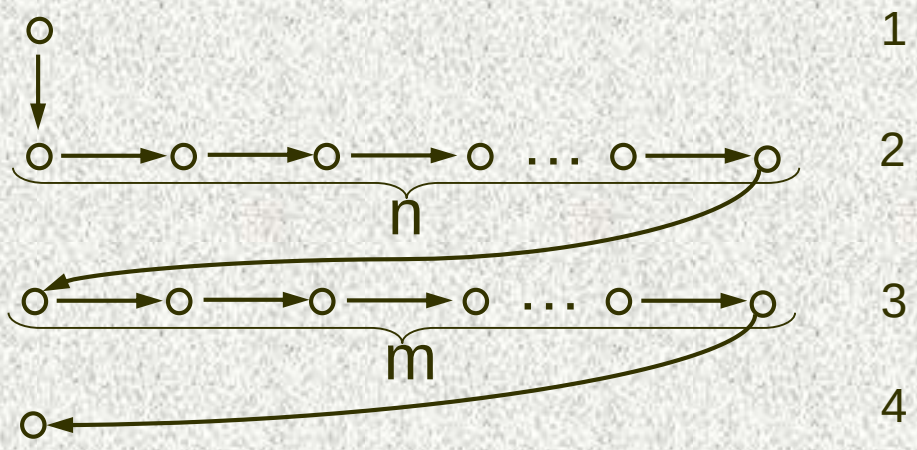
Для входа s:

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

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

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)
```



$$\begin{cases} m \geq 1 \\ j_1 = m \\ из\ 3 \end{cases} \qquad \begin{cases} m < 1 \\ n \geq 1 \\ j_1 = n \\ из\ 2 \end{cases} \qquad \begin{cases} m < 1 \\ n < 1 \\ из\ 1 \end{cases}$$

Вспомним прошлые лекции…

(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]
```

Порядок циклов: (i, j, k)

Возможен ли порядок:

(i, k, j) - ?	ДА
(k, i, j) - ?	ДА
(k, j, i) - ?	ДА
(j, i, k) - ?	ДА
(j, k, i) - ?	ДА

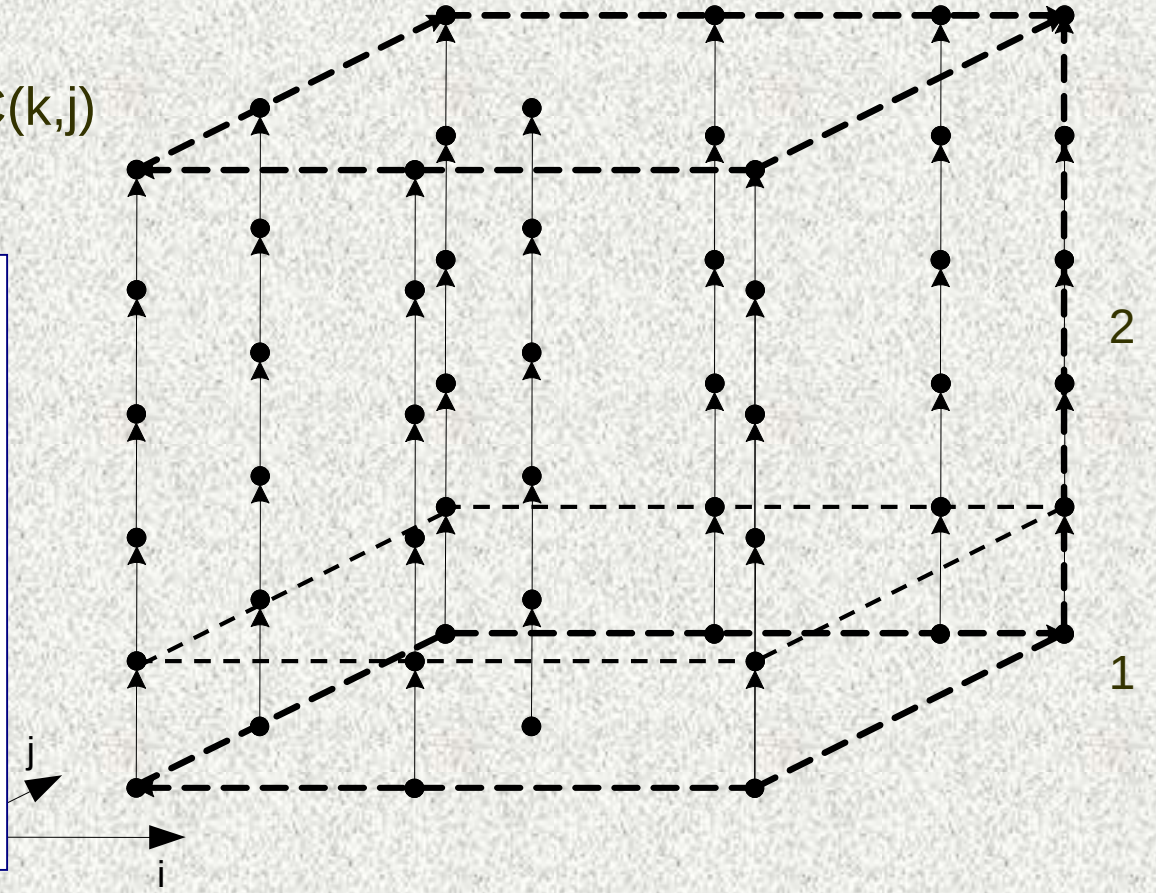
Почему возможен другой порядок?

А зачем нужен другой порядок?

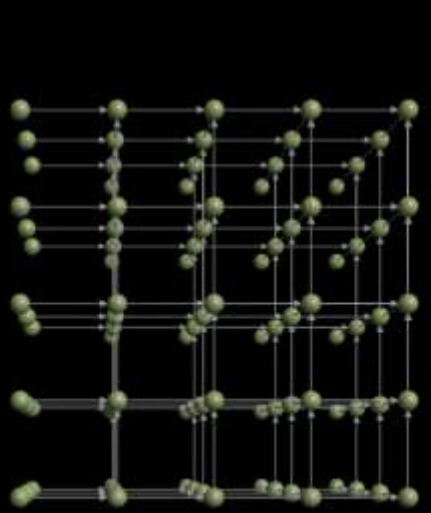
Программы и их графы алгоритма
(умножение матриц)

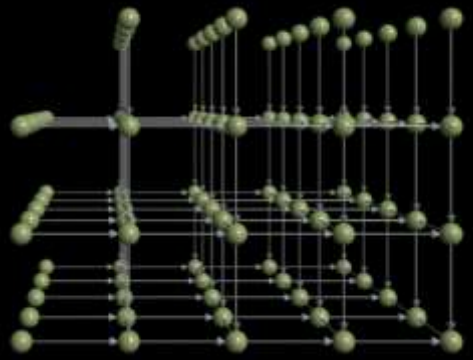
```
    Do i = 1, n
       Do j = 1, n
1        A(i,j) = 0
          Do k = 1, n
2           A(i,j) = A(i,j) + B(i,k)*C(k,j)
```

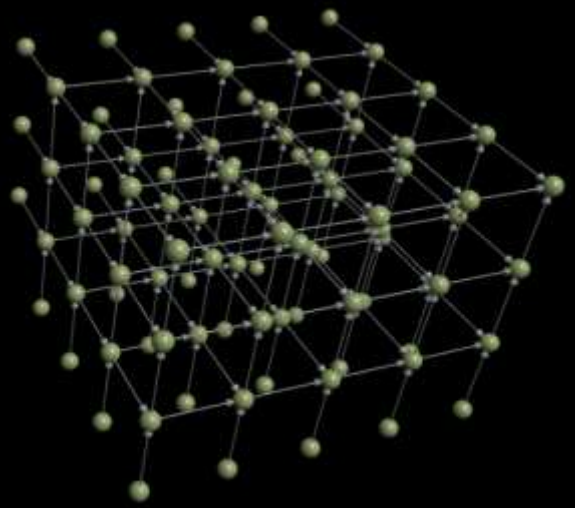
$$\begin{cases} 1 \le i \le n \\ 1 \le j \le n \\ k = 1 \\ \begin{cases} i_1 = i \\ j_1 = j \end{cases} \\ из\ (1) \end{cases} \qquad \begin{cases} 1 \le i \le n \\ 1 \le j \le n \\ 2 \le k \le n \\ \begin{cases} i_1 = i \\ j_1 = j \\ k_1 = k - 1 \end{cases} \\ из\ (2) \end{cases}$$

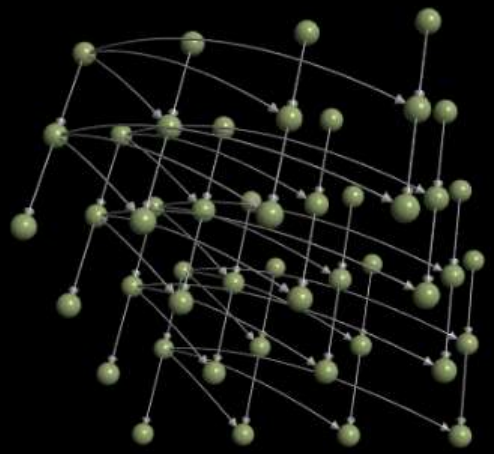


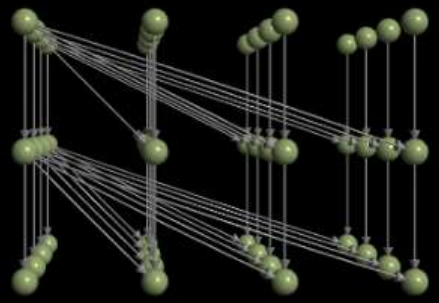
Информационная структура алгоритмов и программ

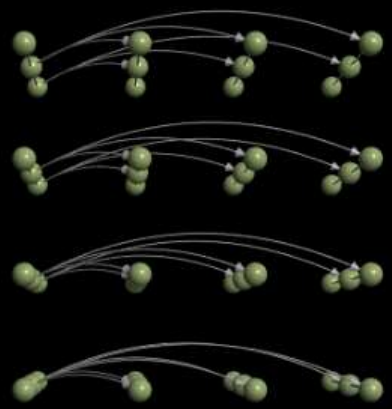




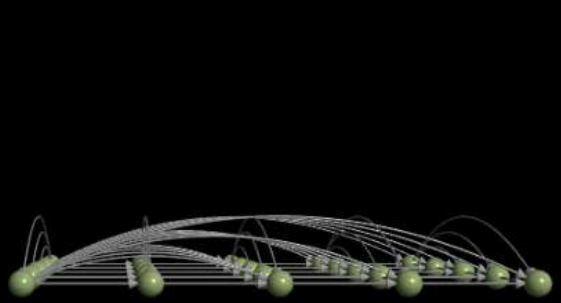


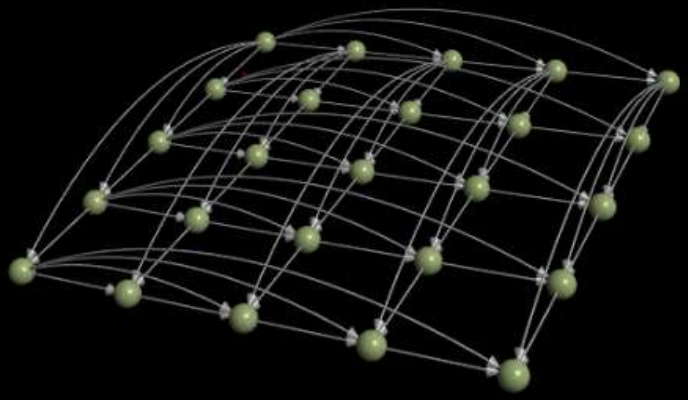


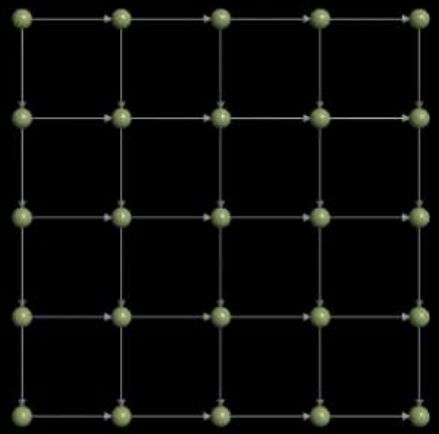




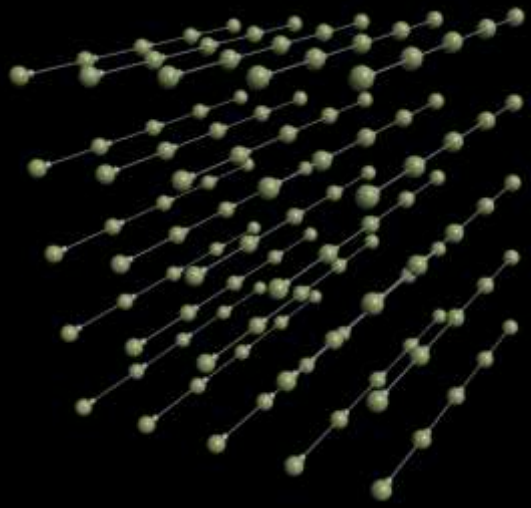
Информационная структура алгоритмов и программ



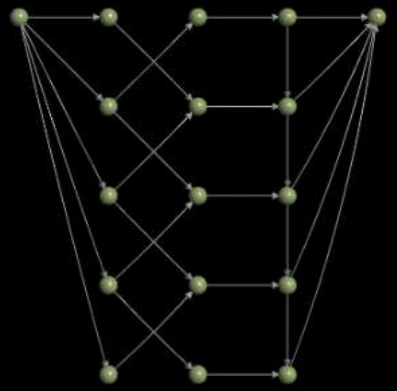




Типовые алгоритмические структуры



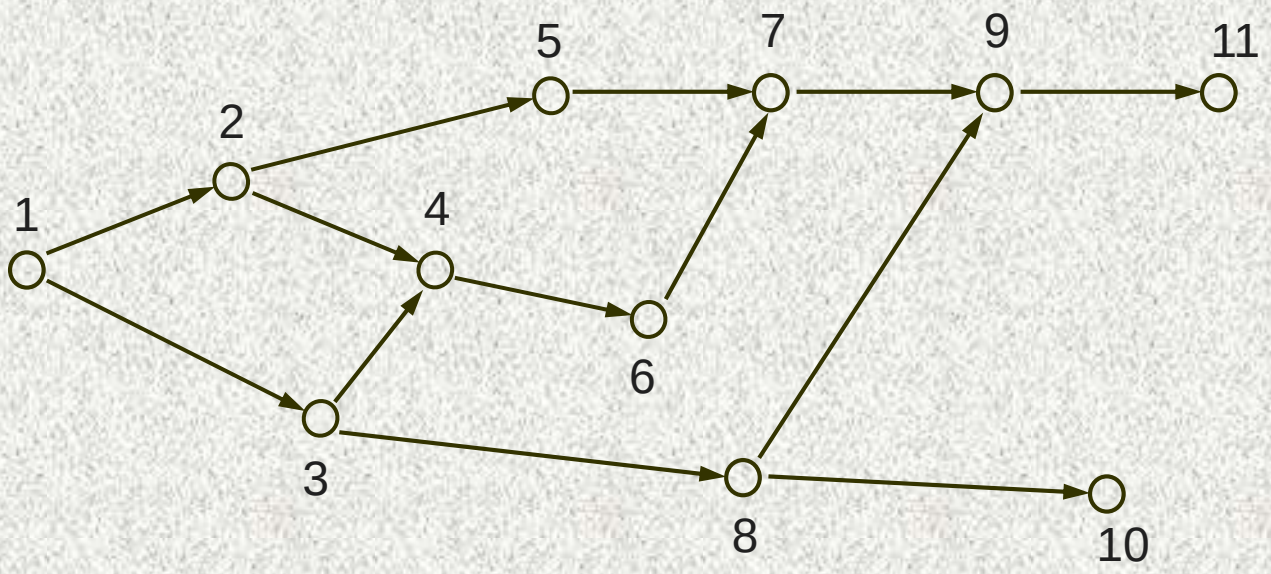
Огромный ресурс параллелизма



А такой структуры быть не может …

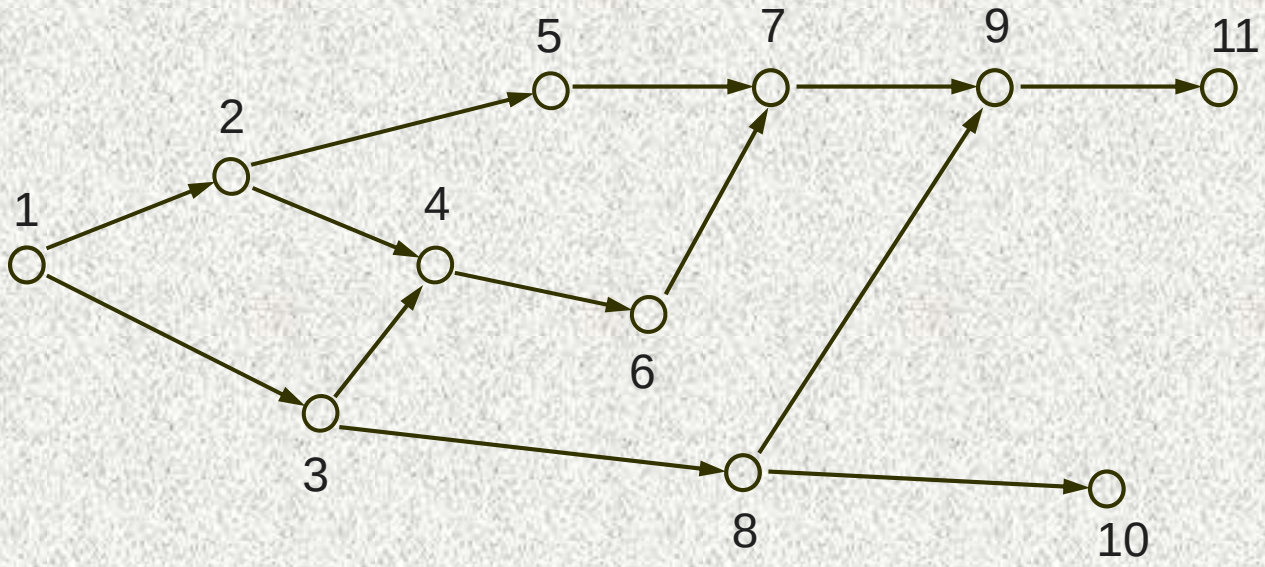
Как описать ресурс параллелизма программ и алгоритмов?

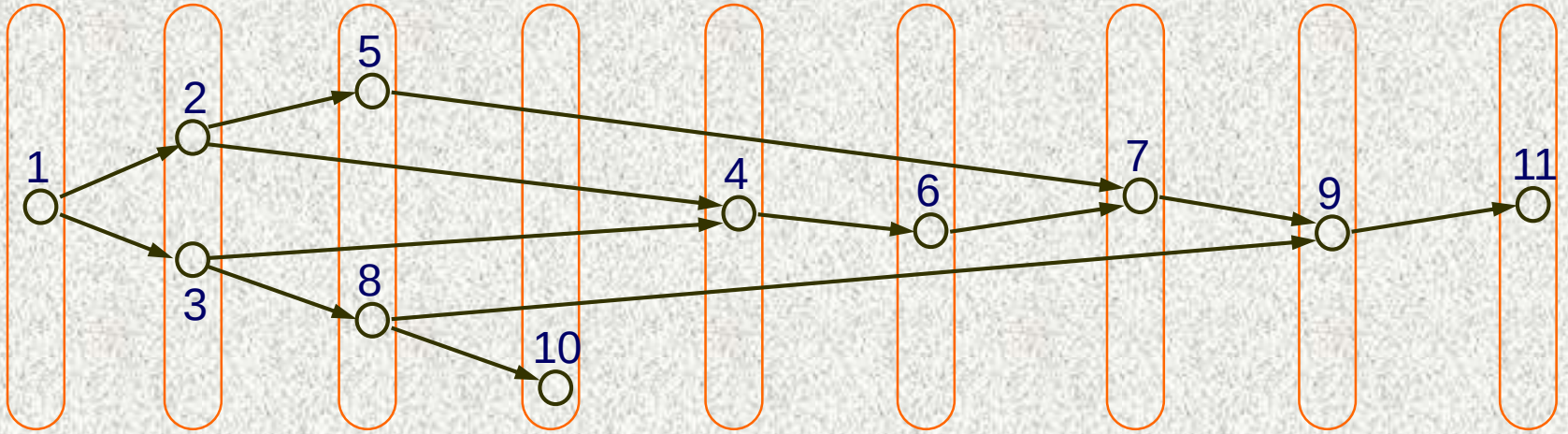
Ярусно-параллельная форма графа алгоритма



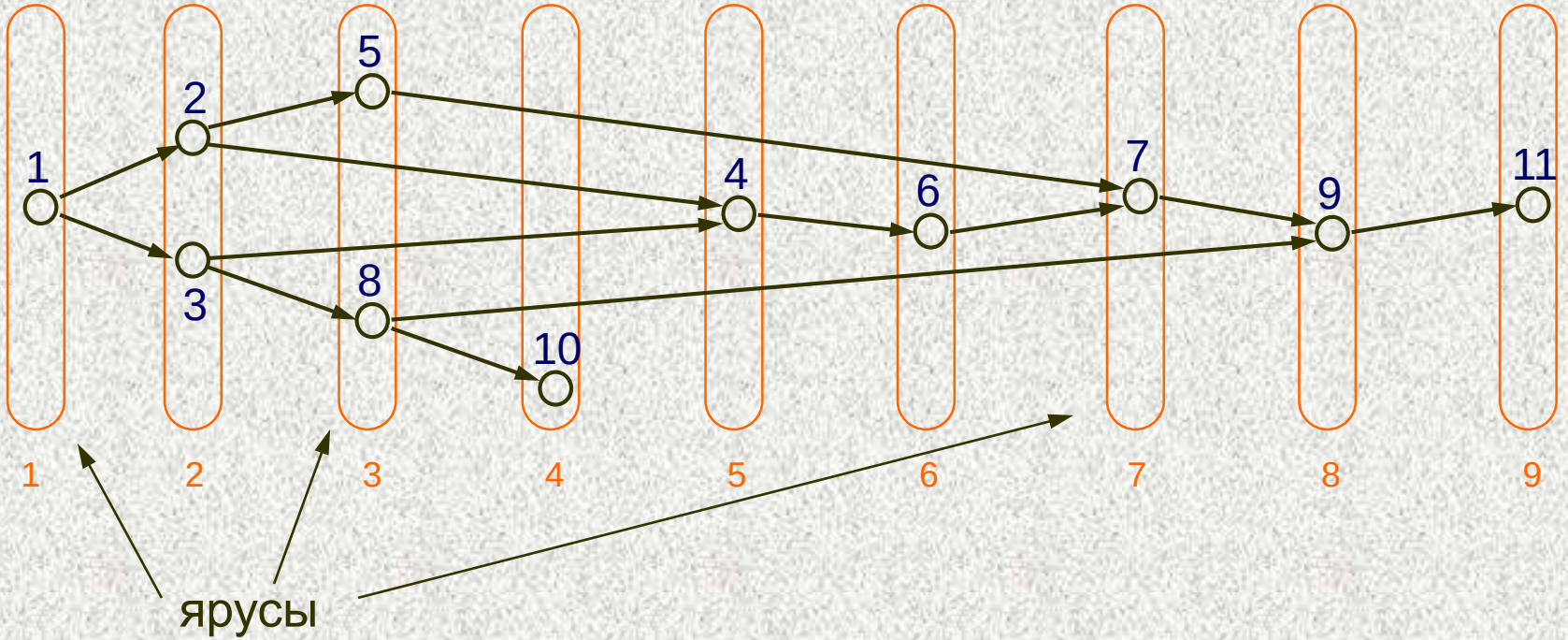
Как определить и сделать понятным ресурс параллелизма в графе алгоритма (в программе, в алгоритме) ?

Ярусно-параллельная форма графа алгоритма





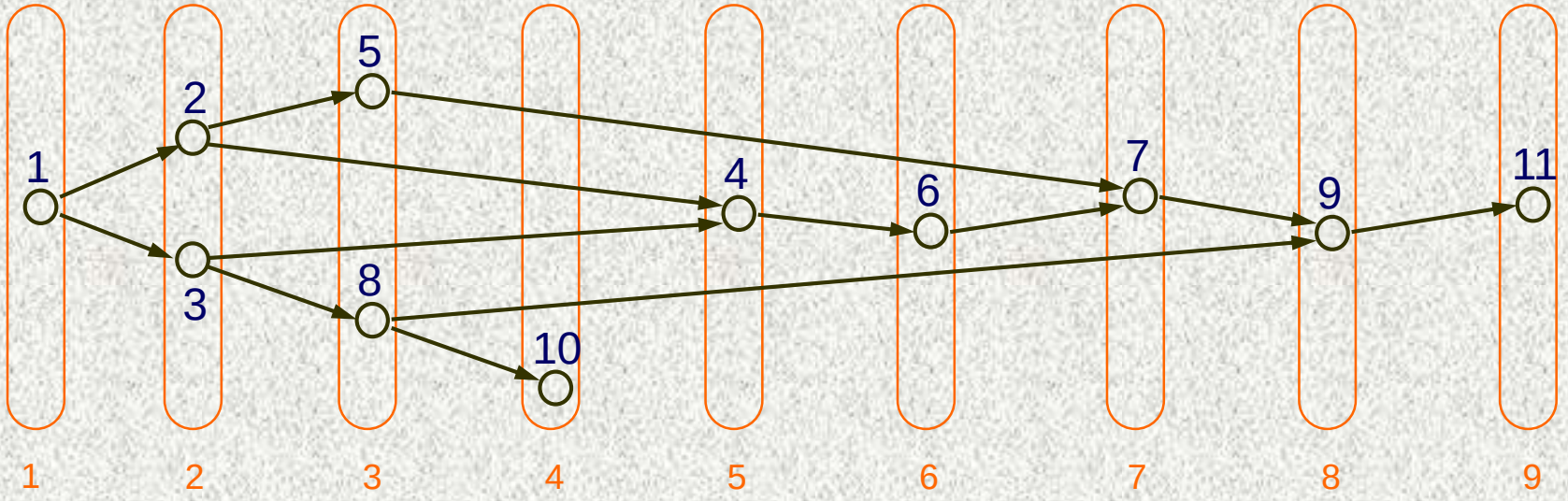
Ярусно-параллельная форма графа алгоритма



- 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



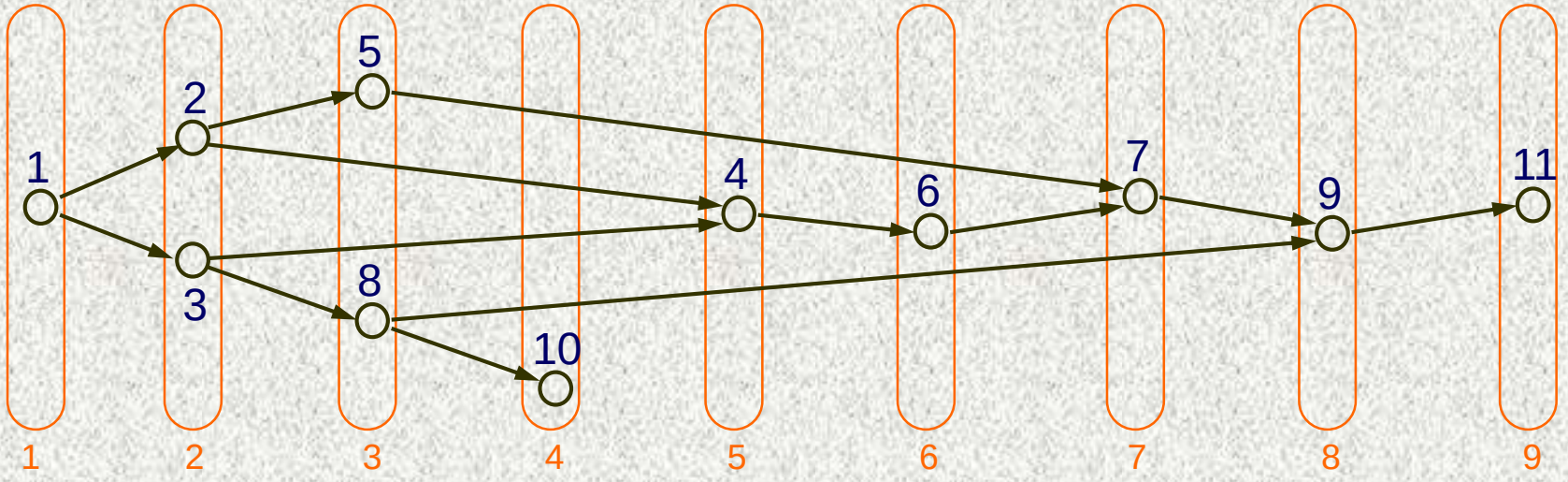
Высота ЯПФ – это число ярусов,

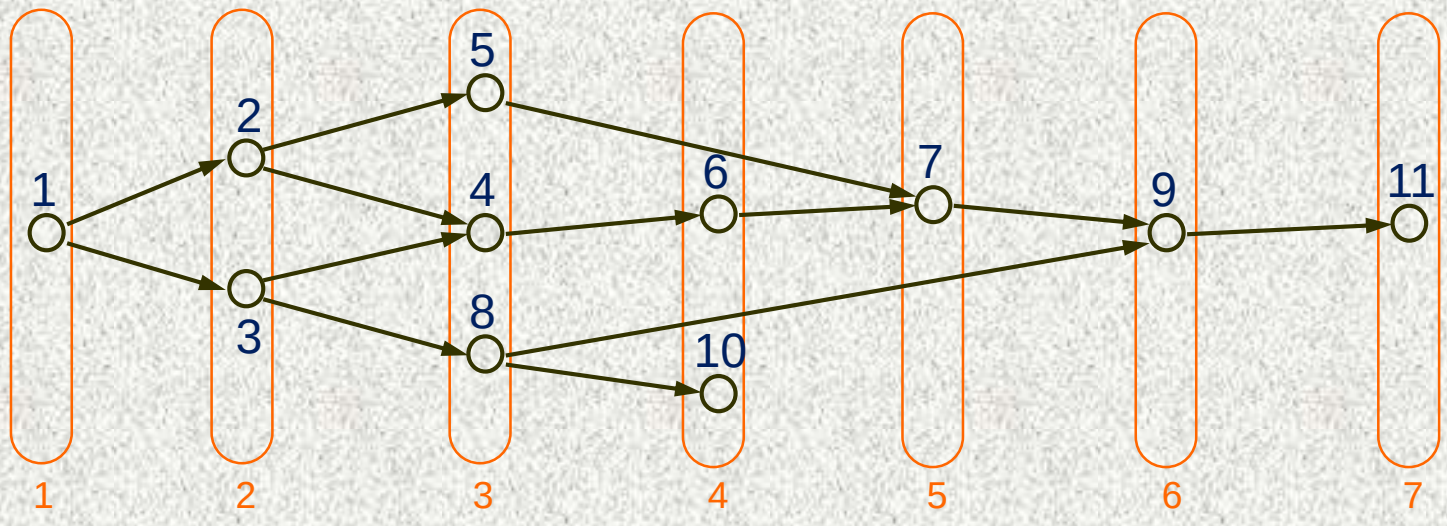
Ширина яруса – число вершин, расположенных на ярусе,

Ширина ЯПФ – это максимальная ширина ярусов в ЯПФ.

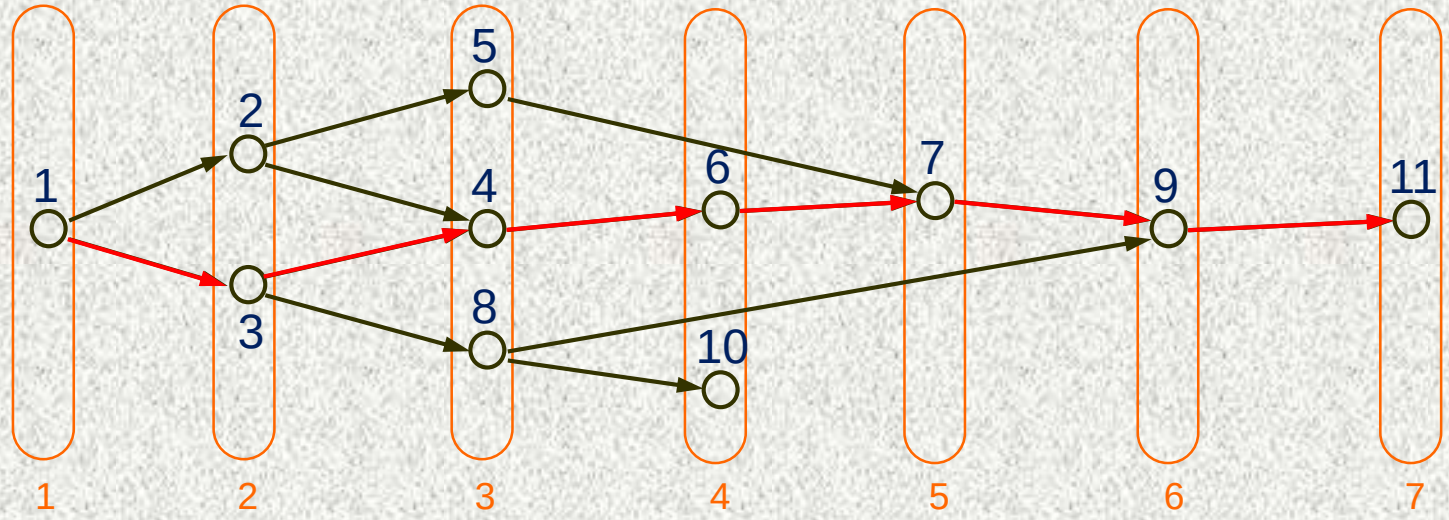
Высота ярусно-параллельной формы - это сложность параллельной реализации алгоритма/программы.

Ярусно-параллельная форма графа алгоритма определяется неоднозначно





Каноническая ярусно-параллельная форма графа алгоритма

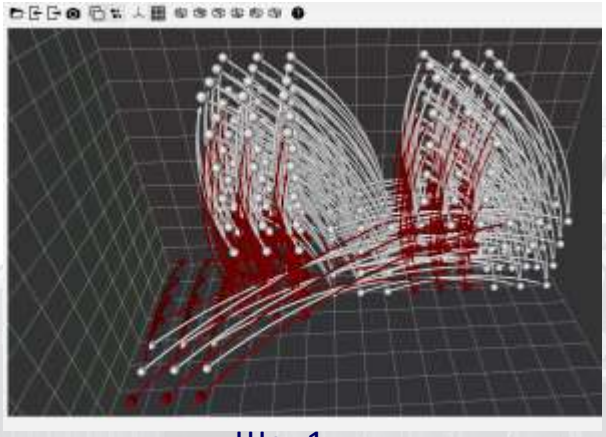


Ярусно-параллельная форма называется канонической, если у любой вершины, кроме вершин первого яруса, есть входная дуга, идущая с предыдущего яруса.

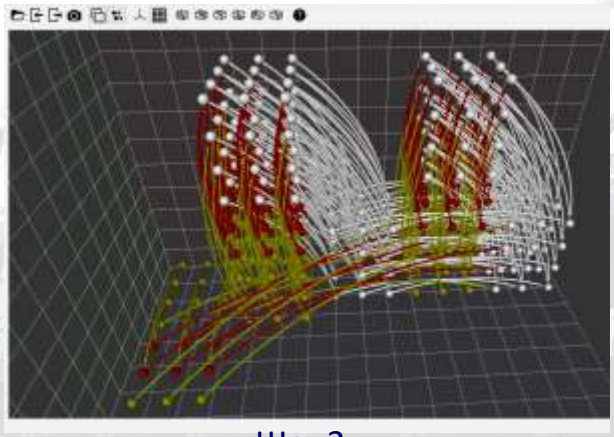
Высота канонической ЯПФ = длине критического пути + 1.

Критический путь в ориентированном ациклическом графе – это путь максимальной длины.

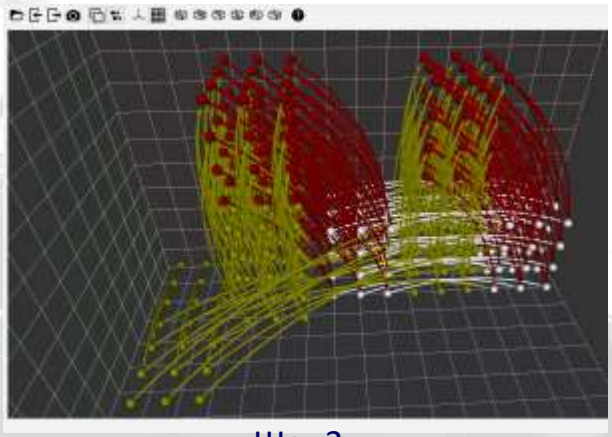
Информационная структура: как показывать?



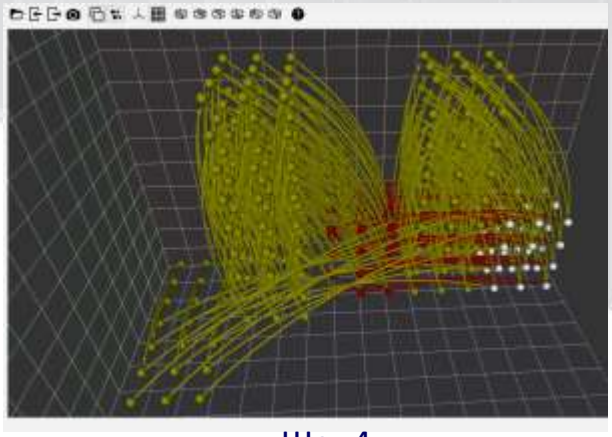
Шаг 1



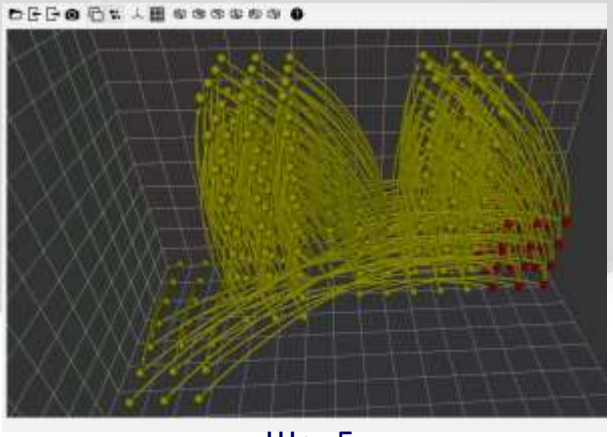
Шаг 2



Шаг 3



Шаг 4



Шаг 5

Цвета в ярусно-параллельной форме:

Красный – слой выполняющихся операций; **Зеленый** – уже выполненные операции;

Белый – операции должны выполниться позже.

Каноническая ЯПФ и степень параллелизма

(каков у фрагмента ресурс 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:

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

где:

α – доля последовательных операций,

p – число процессоров в системе.

Каноническая ЯПФ и степень параллелизма

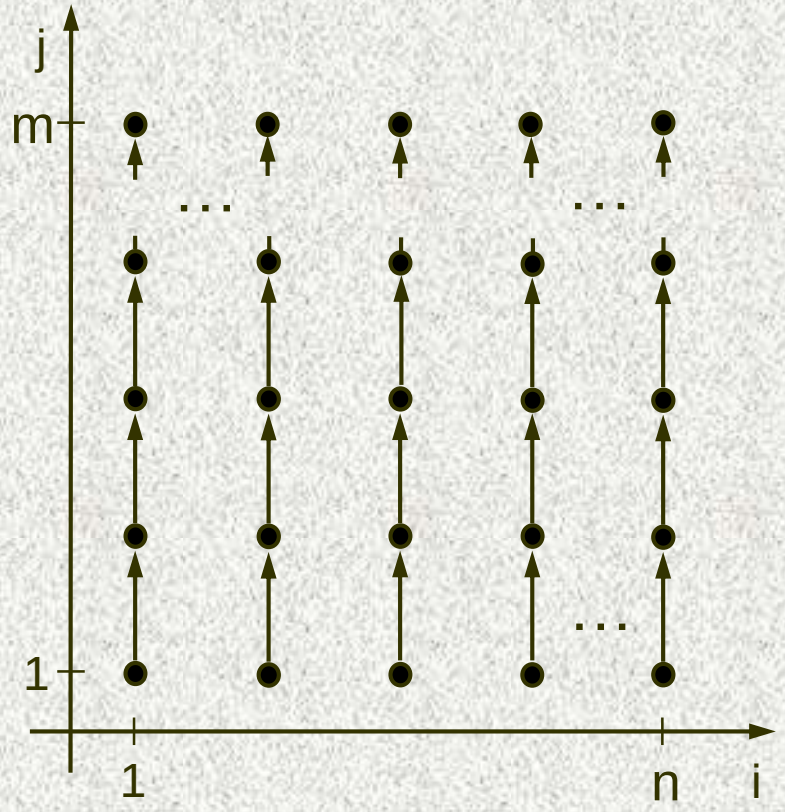
(каков у фрагмента ресурс параллелизма?)

```
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

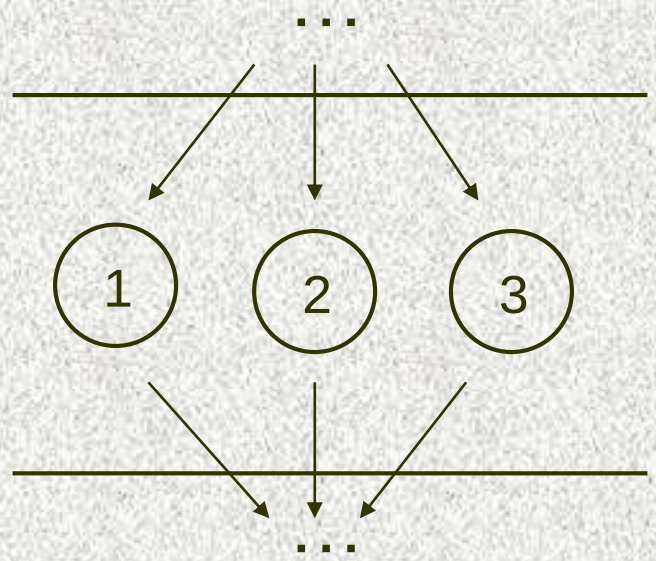
Конечный параллелизм.

```
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

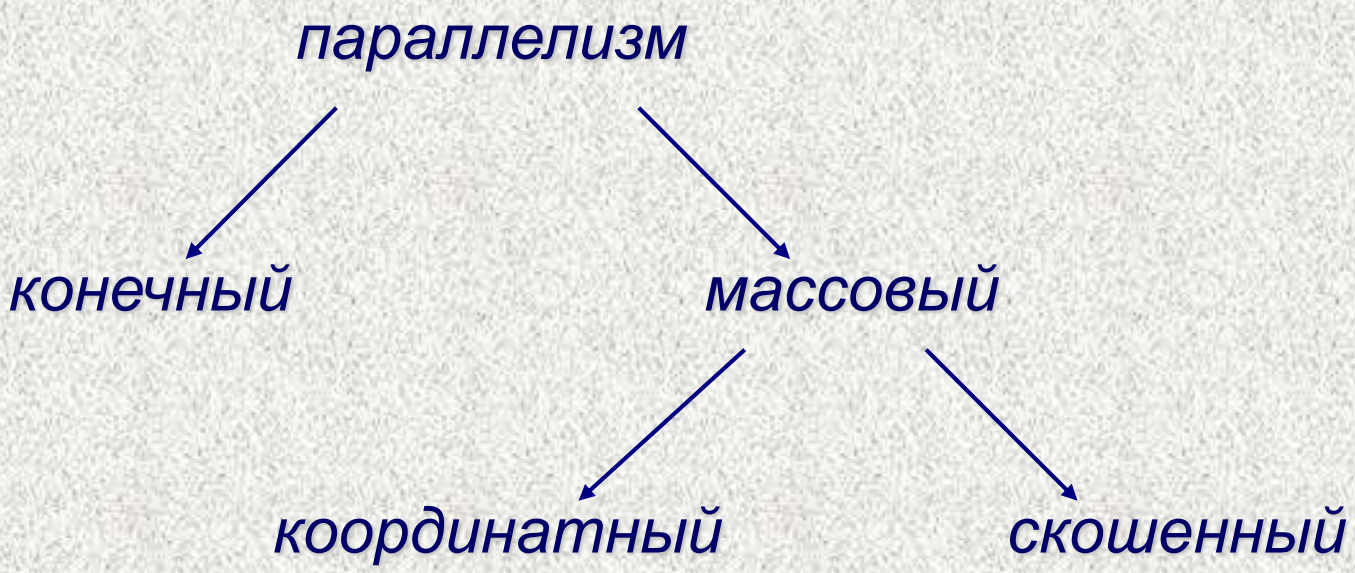


Виды параллелизма в алгоритмах и программах

Массовый параллелизм.

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

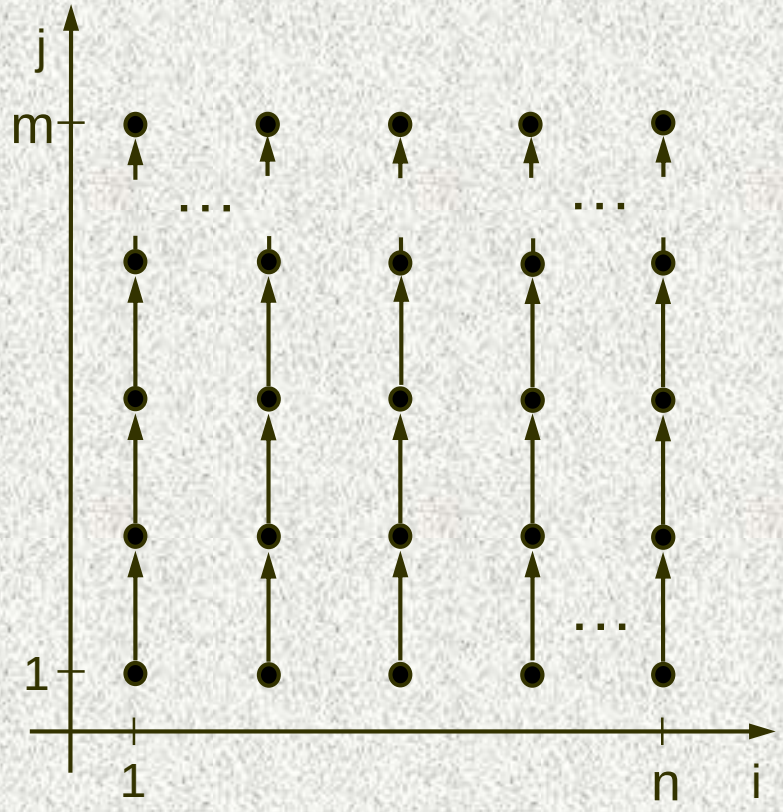
Виды параллелизма в алгоритмах и программах



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

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,*

$G_i = \{f_1 = i_1,$

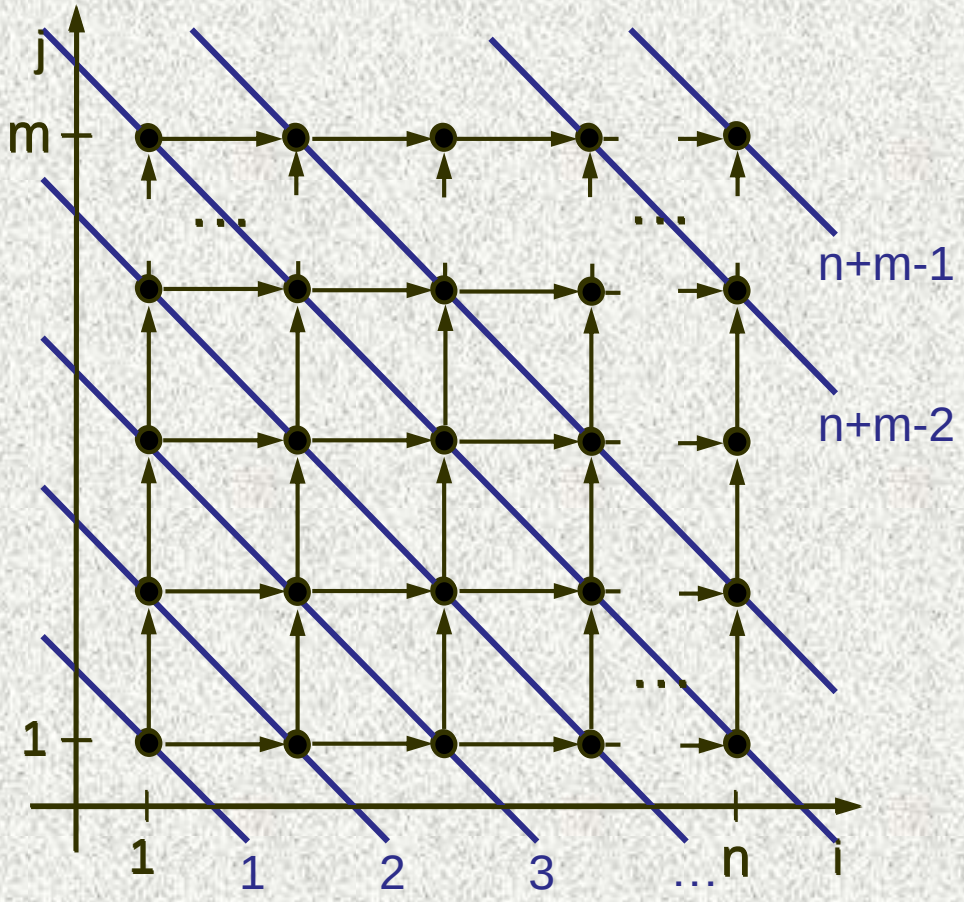
i_1 *— это параметр анализируемого цикла,*

f_1 *— это первая компонента векторной функции* F_i *из тройки.*

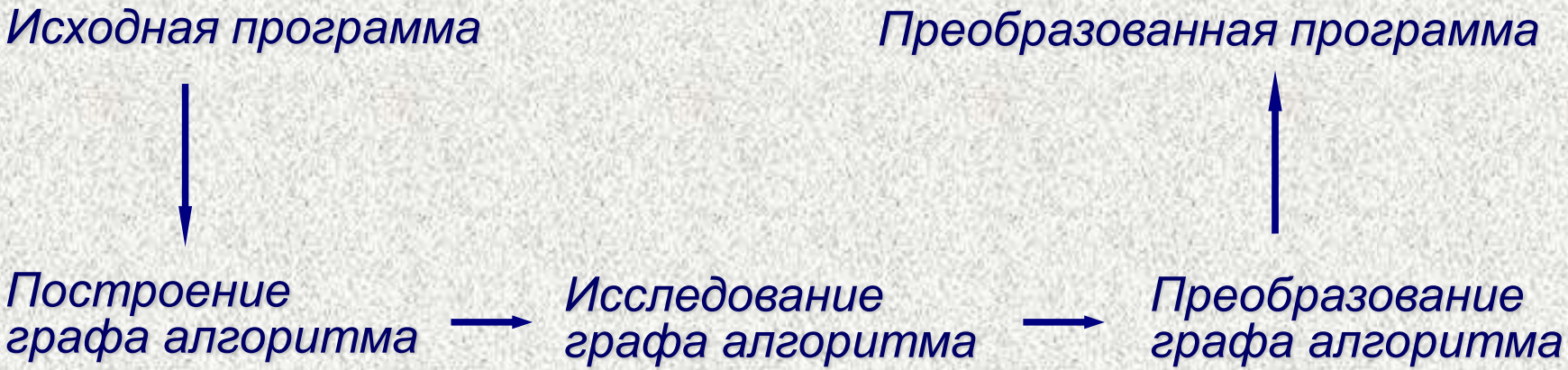
Виды параллелизма в алгоритмах и программах

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

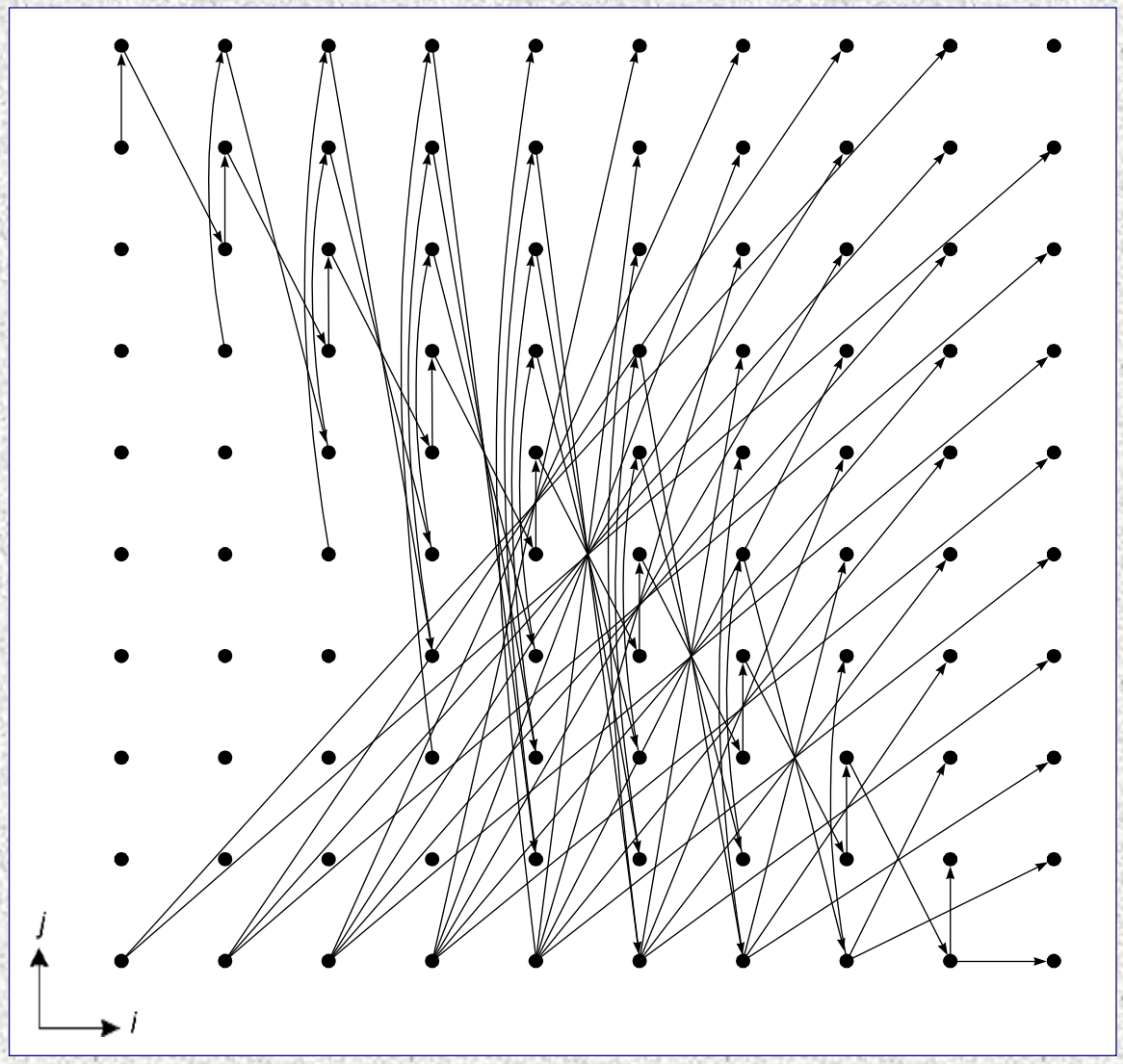


Означает ли малый размер программы простоту ее информационной структуры?

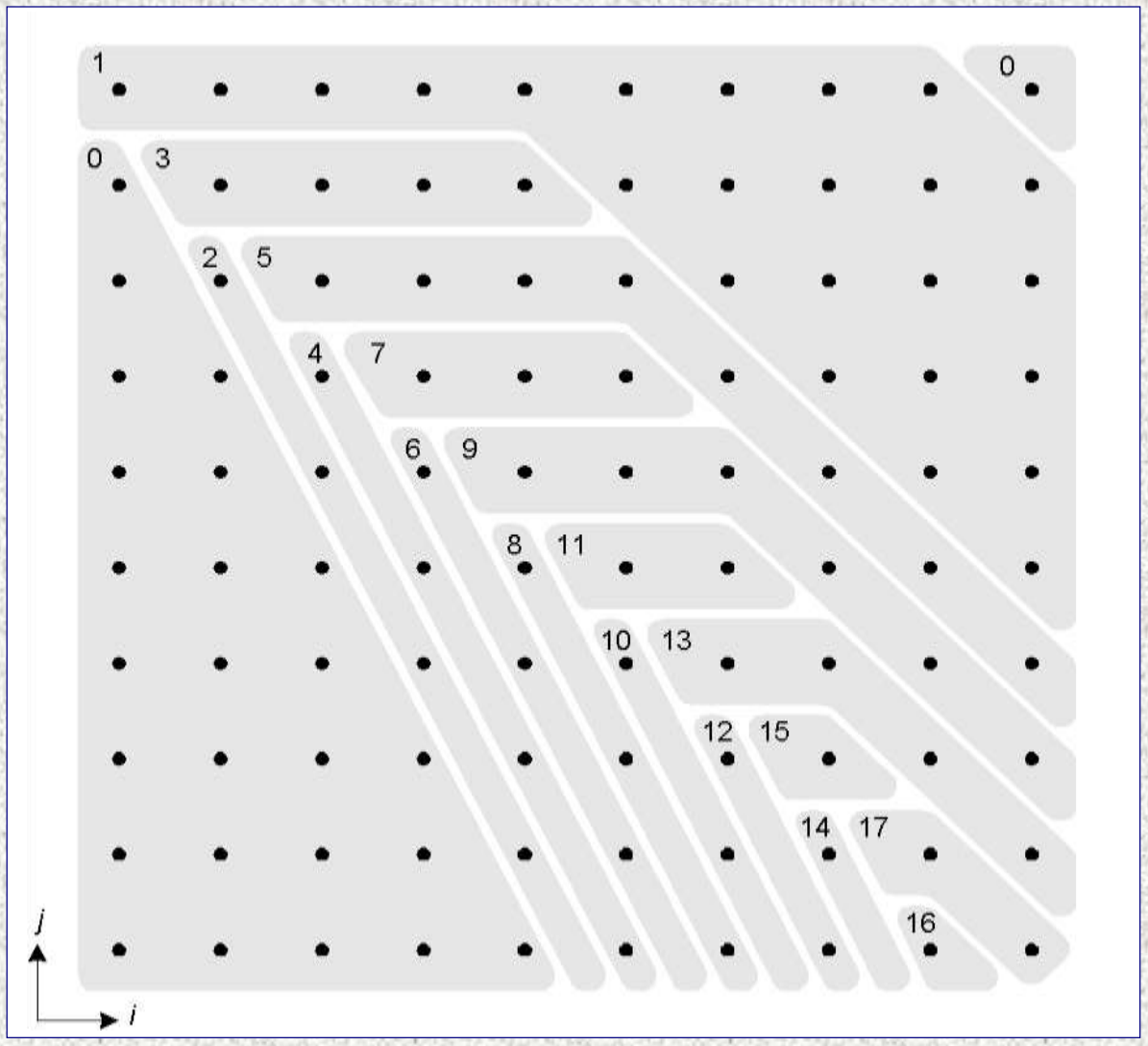
Простой пример…
(последовательный вариант)

```
DO  i = 1, n
    DO j = 1, n
        U( i + j ) = U( 2*n – i – j + 1)*q + p
    EndDO
EndDO
```

Информационная структура примера…



Ярусно-параллельная форма примера…



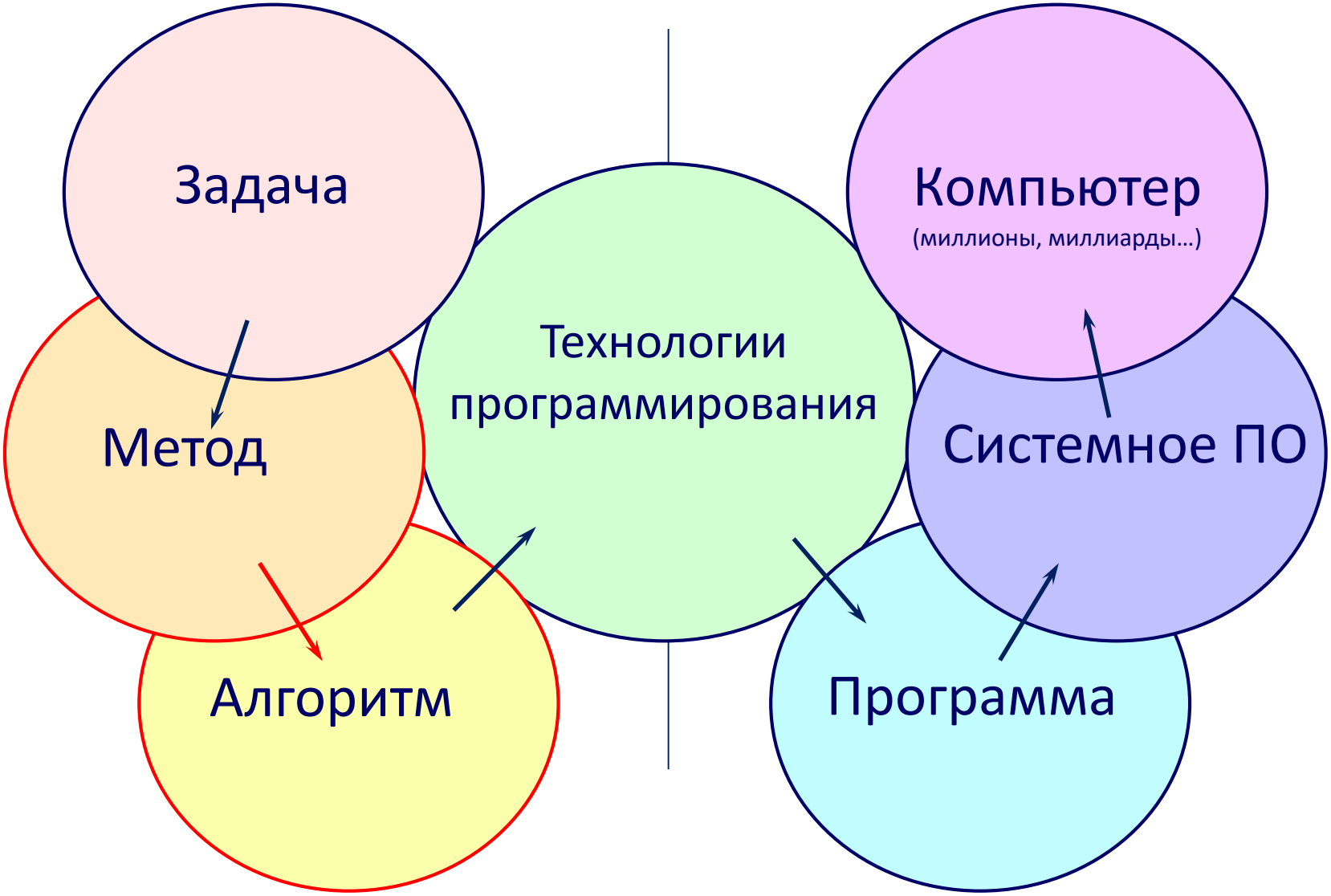
Совсем не простой пример…
(параллельный вариант)

```
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
```

Вспомним прошлые лекции…

Почему, говоря о решении задач на параллельных компьютерах, мы разделяем этапы “Метод” и “Алгоритм” ?

Решение задачи на компьютере



Рассмотрим решение верхней треугольной системы методом Гаусса…

Решение СЛАУ: от метода к алгоритму

(информационная структура)

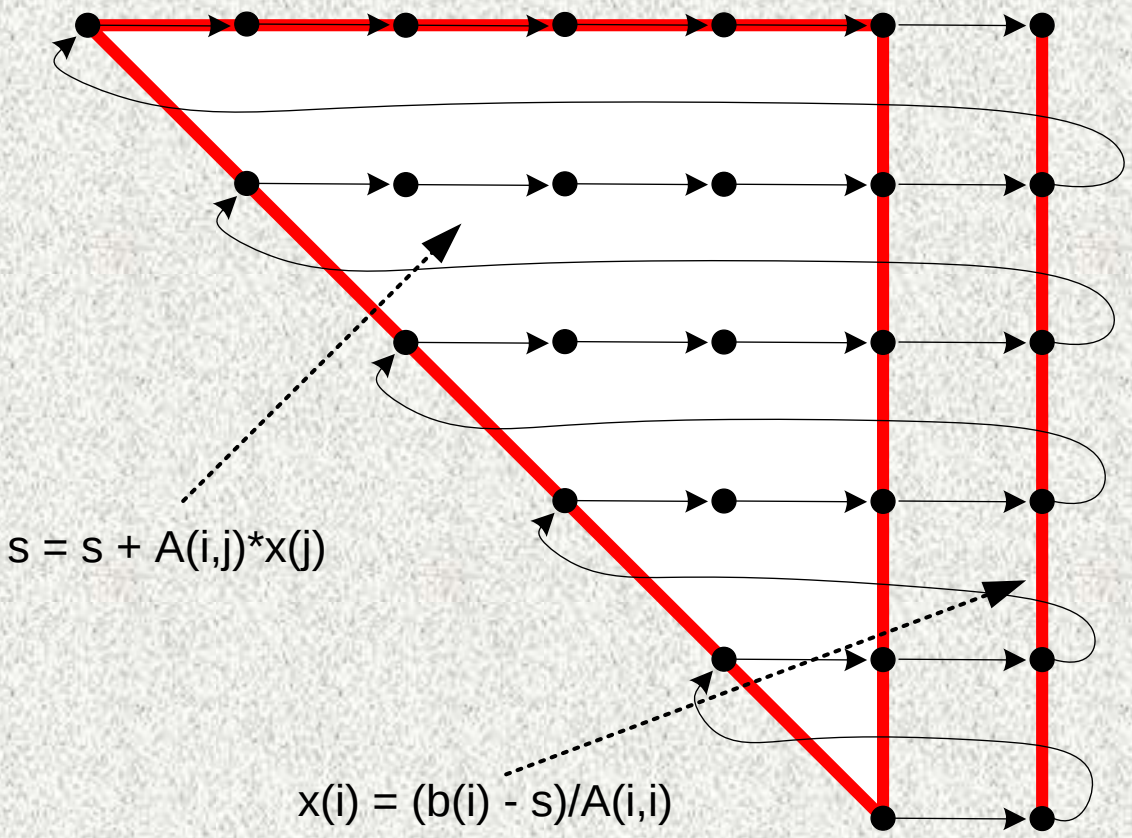


Схема программы:

```
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
```

Критический путь графа алгоритма проходит через все вершины, следовательно такую программу нет смысла исполнять на параллельной вычислительной системе!

С точки зрения метода Гаусса
не имеет никакого значения,
в каком порядке выполнять итерации
внутреннего цикла по j (суммирование)

Изменим в программе только этот порядок
и определим информационную структуру.

Решение СЛАУ: от метода к алгоритму

(информационная структура)

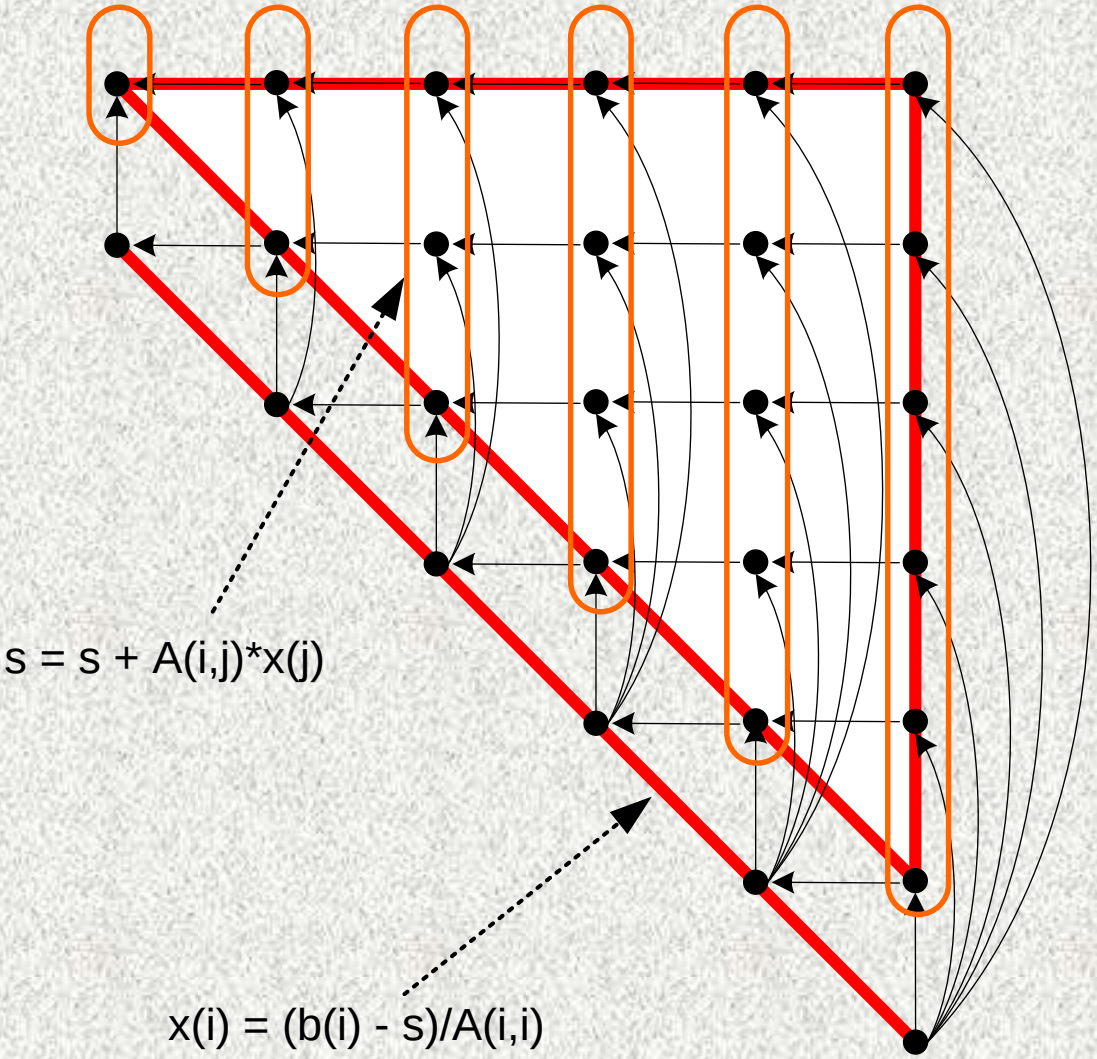


Схема программы:

```
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
```

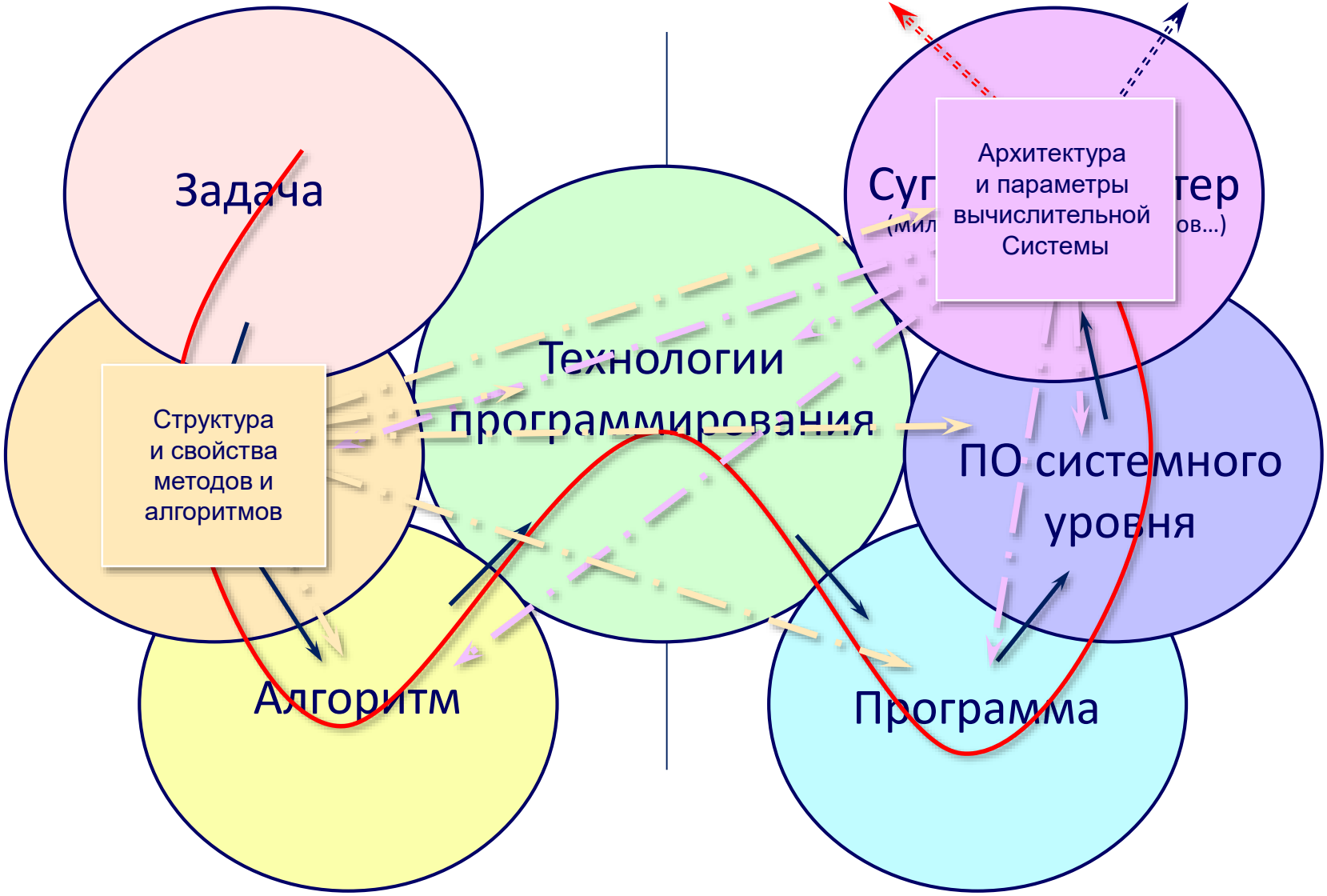
Критический путь графа алгоритма имеет длину o(n), следовательно данная программа обладает хорошим ресурсом параллелизма!

Суперкомпьютерный кодизайн

(Реальность)
Реальная производительность

Огромный разрыв !

(Ожидания)
Пиковая производительность



Алгоритмическая сторона

Компьютерная сторона

AlgoWiki

Файл Правка Вид Журнал Закладки Инструменты Справка

Алговики

https://algowiki-project.org/ru/Открытая_энциклопедия_свойств_алгоритмов

Поиск

Создать учётную запись Войти

AlgoWiki

Main page Обсуждение

Читать Просмотр История Поиск

Открытая энциклопедия свойств алгоритмов

Заглавная страница
Общий форум
Технический форум
Справка
Случайная статья
Свежие правки

Хранилище файлов
Новые файлы
Загрузить файл

Инструменты
Ссылки сюда
Связанные правки
Спецстраницы
Версия для печати
Постоянная ссылка
Сведения о странице

На других языках
English

Добро пожаловать! Присоединяйтесь!

AlgoWiki - это открытая энциклопедия по **свойствам алгоритмов и особенностям их реализации** на различных программно-аппаратных платформах от мобильных платформ до экзафлопсных суперкомпьютерных систем с возможностью коллективной работы всего мирового вычислительного сообщества.

Цель **AlgoWiki** - дать исчерпывающее описание алгоритма, которое поможет оценить его потенциал применительно к конкретной параллельной вычислительной платформе. Кроме классических свойств алгоритмов, например, последовательной сложности, в AlgoWiki представлены дополнительные сведения, составляющие в совокупности полную картину об алгоритме: параллельная сложность, параллельная структура, детерминированность, оценки локальности данных, эффективность и масштабируемость, коммуникационный профиль конкретных реализаций и многие другие.

Читать подробнее: О проекте

Структура проекта

Классификация алгоритмов - основной раздел AlgoWiki, содержащий описания всех алгоритмов. Алгоритмы добавляются в подходящий раздел классификации, при необходимости классификация расширяется за счет новых разделов.

Образцовая статья

Разложение Холецкого (метод квадратного корня)

1 Свойства и структура алгоритма

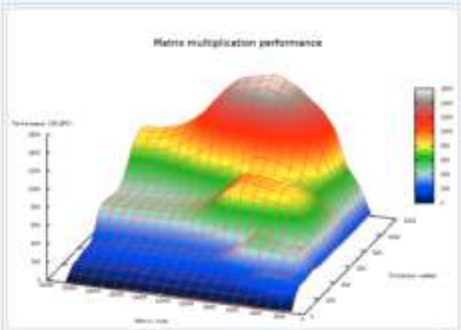
1.1 Общее описание алгоритма

Разложение Холецкого впервые предложено французским офицером и математиком Андре-Луи Холецким в конце Первой Мировой войны, незадолго до его гибели в бою в августе 1918 г. Идея этого разложения была опубликована в 1924 г. его сослуживцем. Потом оно было использовано поляком Т. Банашевичем

Свойства алгоритма:

- Последовательная сложность алгоритма: $O(n^3)$
- Высота ярусно-параллельной формы: $O(n)$
- Ширина ярусно-параллельной формы: $O(n^2)$
- Объём входных данных: $\frac{n(n+1)}{2}$
- Объём выходных данных: $\frac{n(n+1)}{2}$

Изображение дня



Производительность умножения плотных матриц

Организация работы

- Структура описания свойств алгоритмов
- Руководства по заполнению разделов описания
- Готовность статей
- Результаты прогона алгоритмов
- Глоссарий
- Помощь

Участники проекта

Руководители:

http://AlgoWiki-Project.org

Московский государственный университет имени М.В.Ломоносова
факультет Вычислительной математики и кибернетики

СУПЕРКОМПЬЮТЕРЫ
И ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ

ВВЕДЕНИЕ В ТЕОРИЮ АНАЛИЗА СТРУКТУРЫ ПРОГРАММ И АЛГОРИТМОВ

Вл.В.Воеводин

Директор НИВЦ МГУ,
Директор филиала МГУ в г.Сарове,
Зав.кафедрой Суперкомпьютеров и квантовой информатики ВМК МГУ,
чл.-корр. РАН, д.ф.-м.н., профессор

voevodin@parallel.ru

ВМК МГУ, 2025