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

СУПЕРКОМПЬЮТЕРЫ И ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ
(консультация к гос. экзамену)

А.С.Антонов

Вед. н.с. НИВЦ МГУ, к.ф.-м.н.

asa@parallel.ru

ВМК МГУ, 2026

Виды параллельной обработки данных, их особенности. Компьютеры с общей и распределенной памятью. Производительность вычислительных систем, методы оценки и измерения.

Закон Амдала, его следствия. Граф алгоритма. Критический путь графа алгоритма, ярусно-параллельная форма графа алгоритма. Этапы решения задач на параллельных вычислительных системах.

Виды параллельной обработки данных, их особенности. Компьютеры с общей и распределенной памятью. Производительность вычислительных систем, методы оценки и измерения.

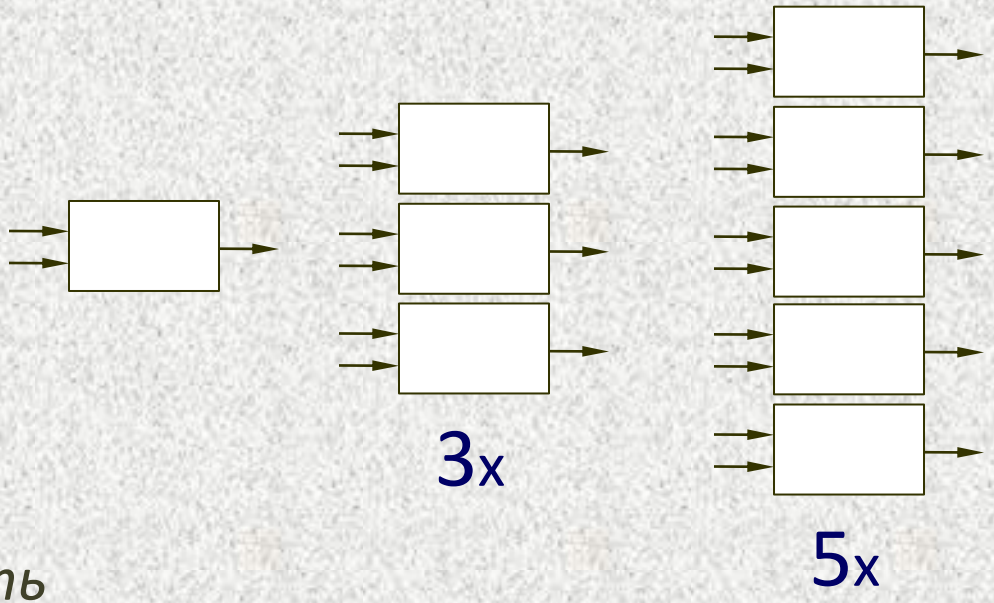
Виды параллельной обработки данных, их особенности

Два вида параллельной обработки данных

- *Параллелизм*
- *Конвейерность*

Параллелизм в архитектуре компьютеров

- *Параллелизм*



- *Конвейерность*

Длина конвейера = число ступеней конвейера.

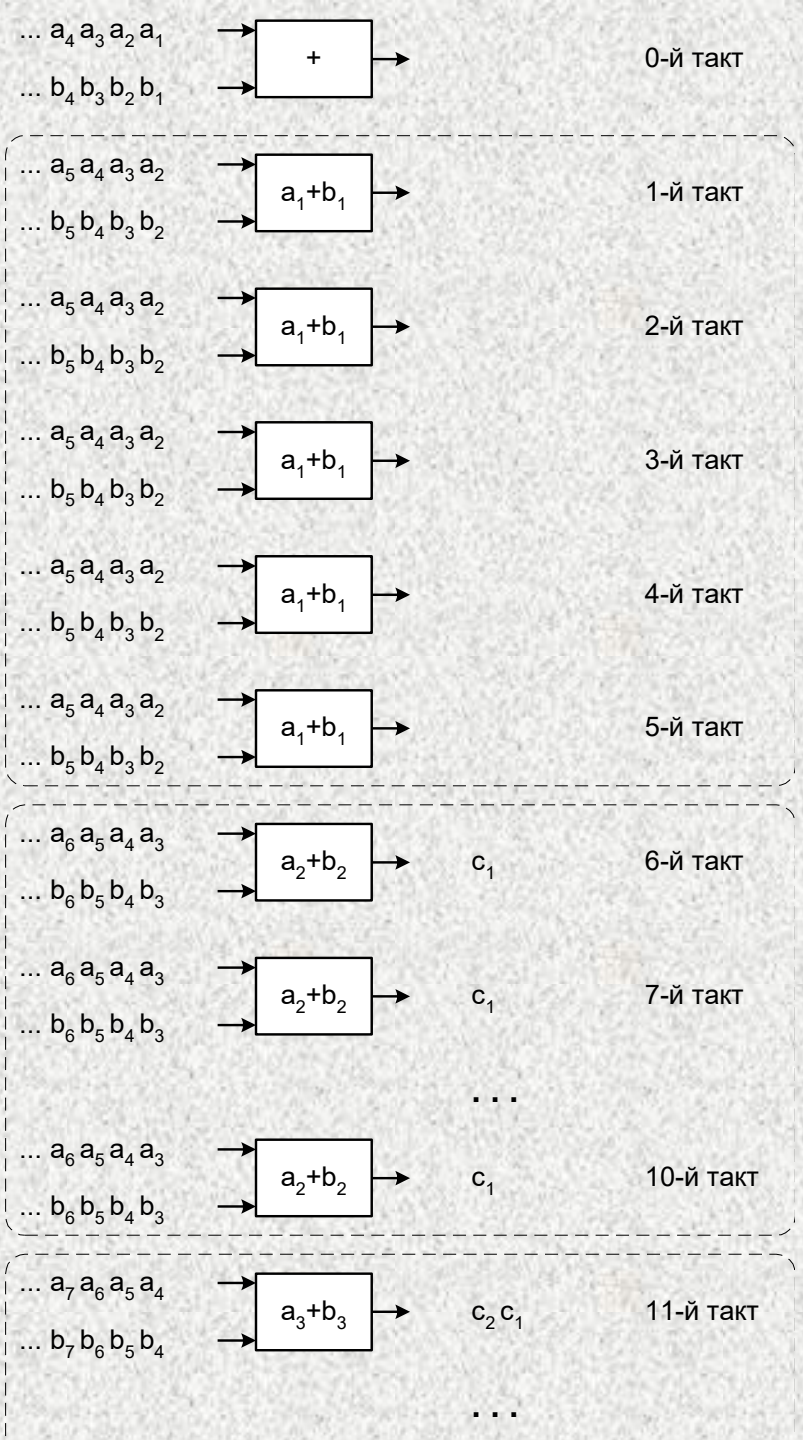
Последовательная обработка данных

Суммирование векторов C = A + B с помощью 1 последовательного устройства.

Устройство выполняет 1 операцию за 5 тактов.

Векторы A и B содержат по 100 элементов.

Время выполнения данной операции: 500 тактов.



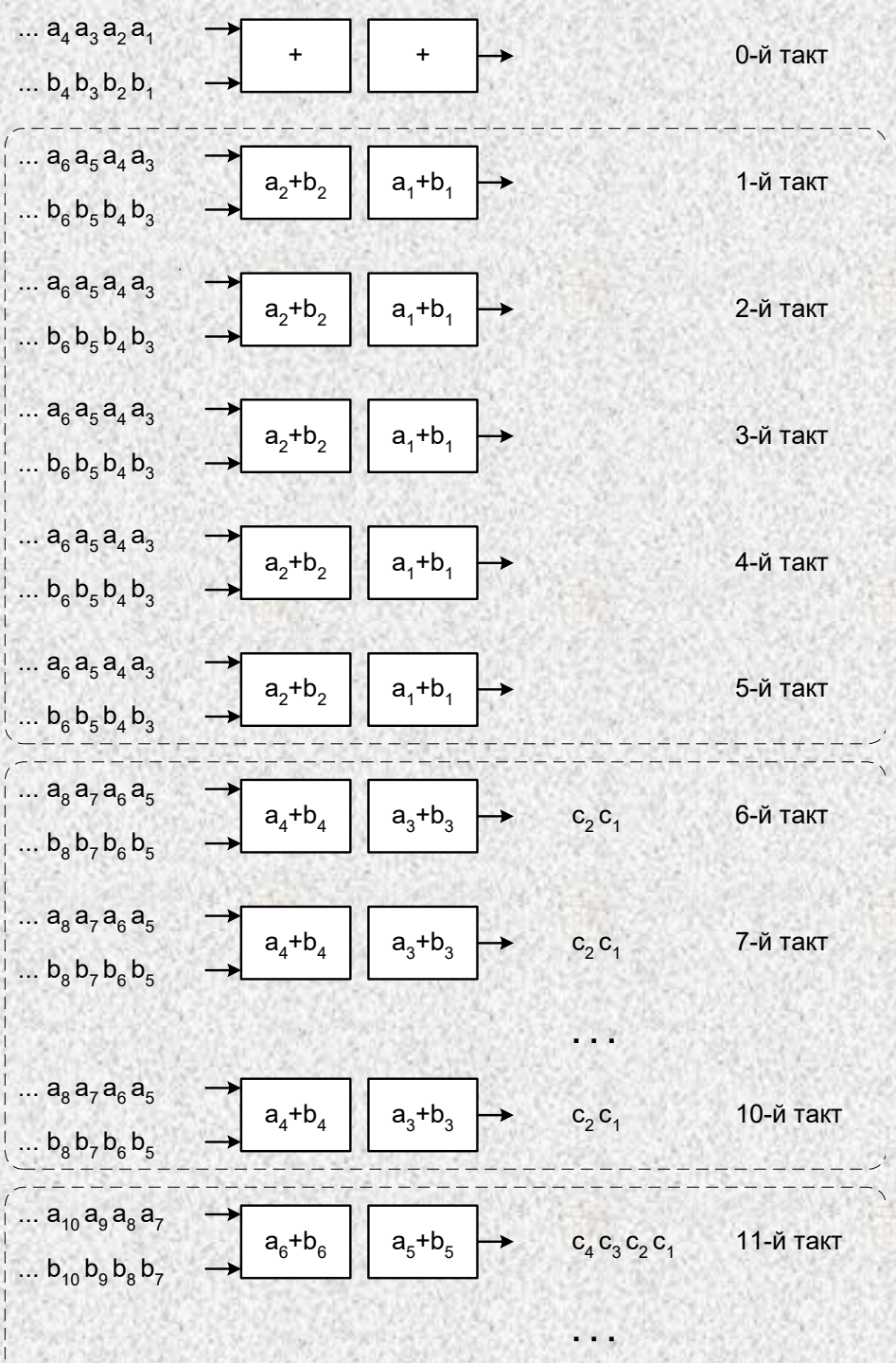
Параллельная обработка данных

Суммирование векторов C = A + B с помощью 2 одинаковых последовательных устройств.

Каждое устройство выполняет по 1 операции за 5 тактов.

Векторы A и B содержат по 100 элементов.

Время выполнения данной операции: 250 тактов.



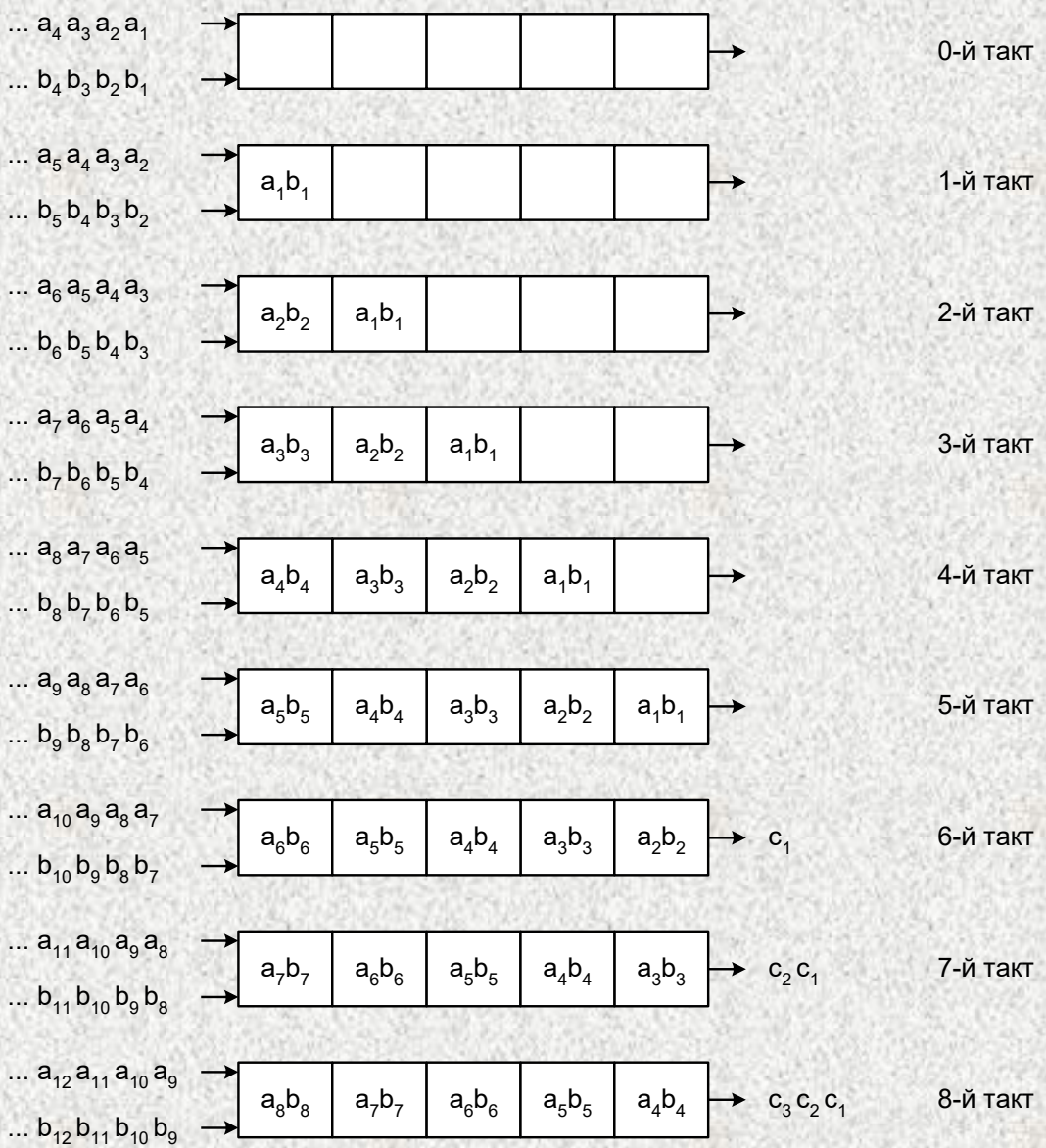
Конвейерная обработка данных

Суммирование векторов C = A + B с помощью конвейерного устройства.

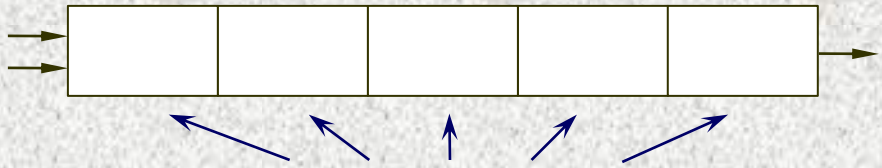
Конвейерное устройство состоит из 5 ступеней. Каждая из 5 ступеней срабатывает за 1 такт.

Векторы A и B содержат по 100 элементов.

Время выполнения данной операции: 104 такта.



Конвейерная обработка данных
(основные понятия)



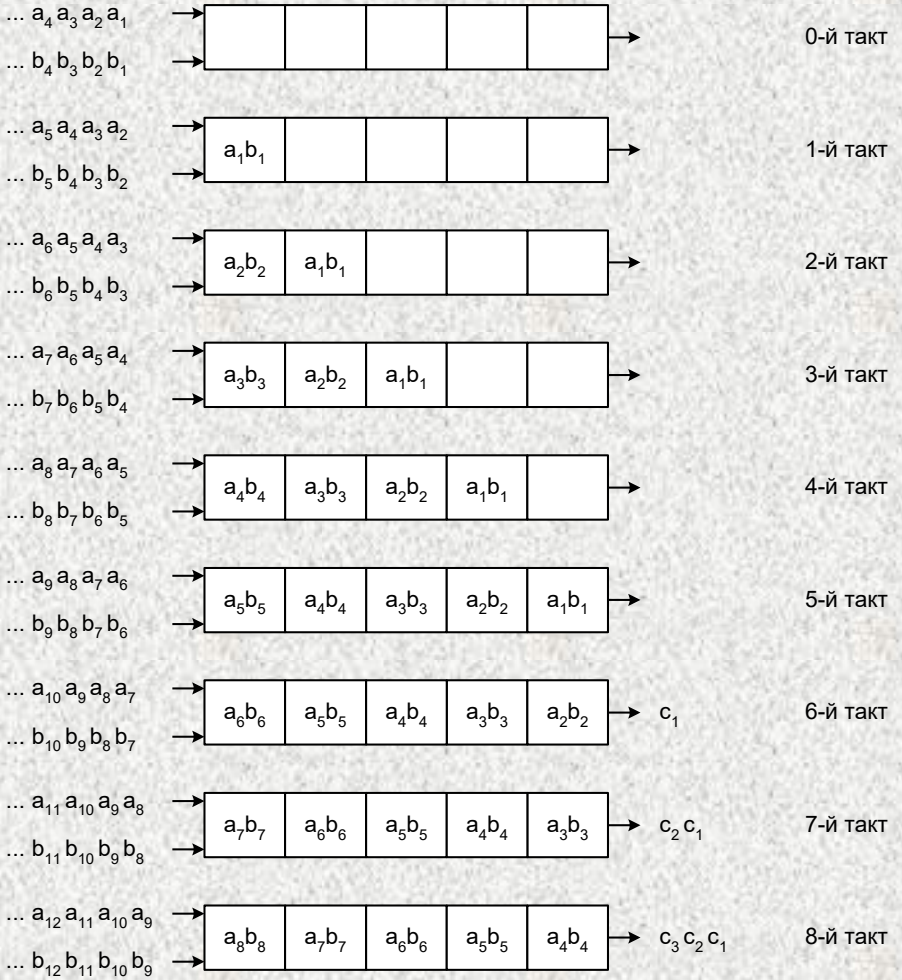
L - длина конвейера (число ступеней конвейера).
Каждая ступень срабатывает за 1 такт.
Темп выдачи результатов – каждый такт.

Время обработки всего набора из N входных данных:

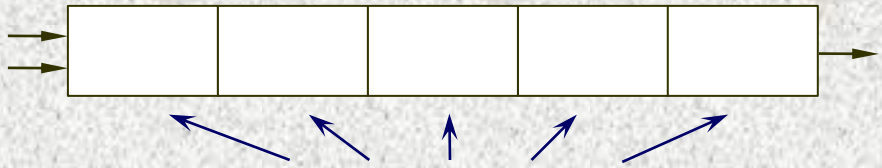
$$T = L + (N-1)$$

Время (число тактов) для заполнения всего конвейера (одновременно будет обработан первый входной аргумент).

Время (число тактов) для обработки оставшегося набора из N-1 входных аргументов.



Конвейерная обработка данных
(вопрос…)

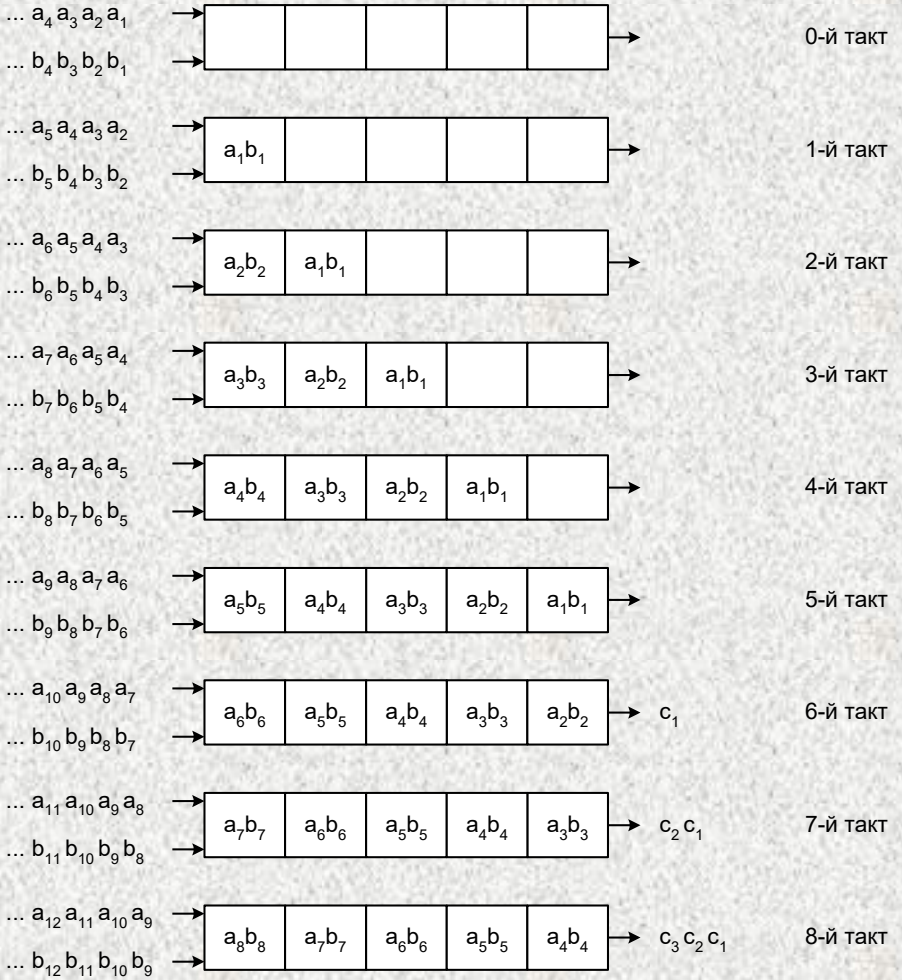


L - длина конвейера (число ступеней конвейера).

Каждая ступень срабатывает за T_i тактов.

С какой частотой будут выдаваться результаты в устоявшемся режиме?

$$\max_i(T_i)$$



Конвейерная обработка данных
(зацепление устройств – это часто на практике!)

L_1 - длина конвейера сложения.
L_2 - длина конвейера умножения.
Каждая ступень срабатывает за 1 такт.

Нужно выполнить операцию

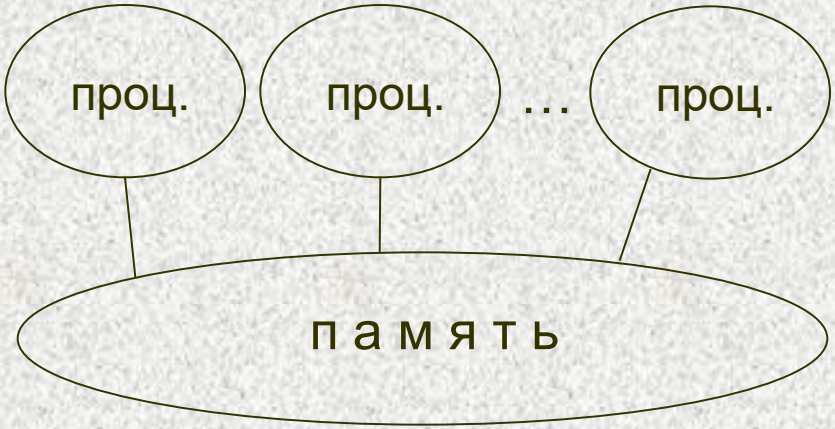
$$A_i = B_i + C_i * d,\ i=1..N$$

Зацепление функциональных устройств – это такой режим обработки, при котором выход одного функционального устройства подаётся на вход другого.

- Время обработки в режиме без зацепления: $T = L_2 + (N-1) + L_1 + (N-1)$
- Время обработки в режиме с зацеплением: $T = L_2 + L_1 + (N-1)$

Компьютеры с общей и распределенной памятью

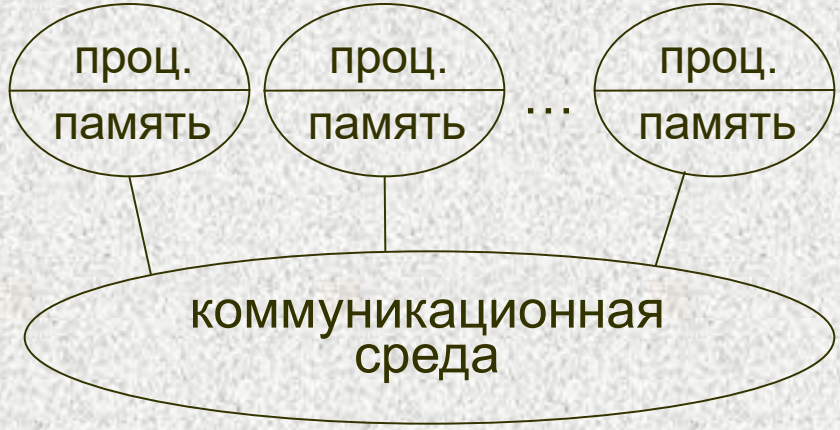
Компьютеры с общей памятью



SMP-компьютеры, два варианта расшифровки:
Shared Memory Processors
Symmetric MultiProcessor

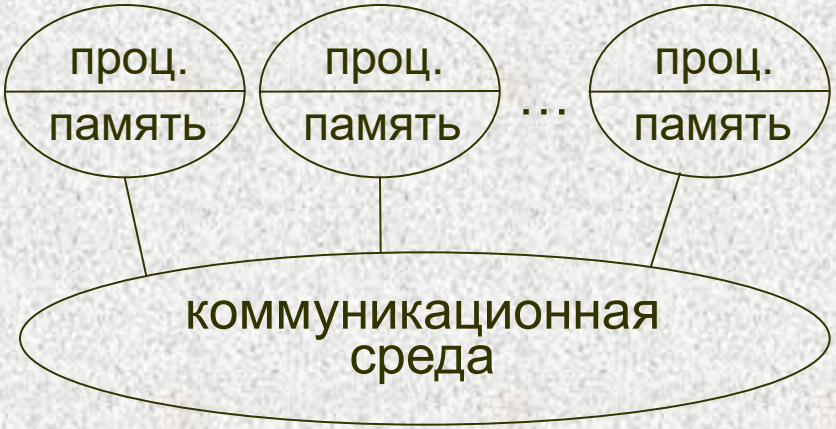
В SMP-компьютерах все, кроме процессоров, в одном экземпляре: образ операционной системы, память, подсистема ввода-вывода…

Компьютеры с распределенной памятью



Компьютеры с распределенной памятью состоят из вычислительных узлов, каждый из которых является полноценным компьютером со своей памятью, ОС, устройствами ввода-вывода и т.п., взаимодействующих друг с другом через коммуникационную среду.

Компьютеры с общей и распределенной памятью



\+ *относительная простота параллельного программирования,*
– *сложность увеличения числа процессоров (роста производительности)*

– *сложность параллельного программирования,*
\+ *относительная простота увеличения числа процессоров (роста производительности)*

Высокопроизводительные компьютеры и две задачи параллельных вычислений

Две задачи параллельных вычислений:

- *построение вычислительных систем с максимальной производительностью:*
  - *это компьютеры с распределенной памятью*

- *эффективное программирование параллельных вычислительных систем:*
  - *это компьютеры с общей памятью.*

Программирование на общей и распределенной памяти

Для эффективного программирования компьютеров С общей памятью необходимо:

1.Найти в программе ресурс параллелизма.
2.Распределить операции по исполнительным устройствам.
3.Разграничить доступ к данным в общей памяти.

Программирование на общей и распределенной памяти

Для эффективного программирования компьютеров с распределенной памятью необходимо:

1. *Найти в программе ресурс параллелизма.*
2. *Распределить данные по модулям памяти вычислительных узлов.*
3. *Распределить операции по исполнительным устройствам.*
4. *Согласовать распределение данных с параллелизмом вычислений.*
5. *Организовать необходимые пересылки данных.*

Производительность вычислительных систем, методы оценки и измерения

Производительность компьютера

(теория и практика)

• *Пиковая производительность компьютера,* R_{peak}

Определяется по архитектуре компьютера в предположении, что все устройства работают с полной загрузкой, без остановок, необходимые аргументы всегда готовы, а передача данных выполняется мгновенно. Это теоретически максимальная производительность компьютера.

• *Реальная производительность компьютера,* R_{max}

Индивидуальна для каждой программы и/или каждого компьютера. Определяется по параметрам работы программы на компьютере:

$$R_{max} = \frac{\text{число операций в программе}}{\text{время выполнения программы на компьютере}}$$

Пиковая производительность недостижима на практике $\Rightarrow$ $R_{max} < R_{peak}$

Методы оценки производительности

Какой компьютер выбрать?

В идеале было бы компьютеру однозначно сопоставить некое число.

***Пиковая производительность**: вычисляется просто и однозначно, но нет связи с реальной задачей пользователя. Даёт нижнюю оценку времени выполнения программы.*

Полезнее для пользователя оценка эффективности программно-аппаратной среды на некоторых задачах или наборе задач.

***Синтетические (или искусственные) тесты** не имеют отношения к реальным приложениям; предназначены для создания стрессовой нагрузки на отдельные подсистемы компьютера.*

***Реальные тесты** выполняют реальные задачи над реальными данными.*

Методы оценки производительности

Основные требования к тестам производительности:

- *Непротиворечивость и понятность результатов.*
- *Легкость в использовании.*
- *Масштабируемость.*
- *Переносимость.*
- *Репрезентативность.*
- *Доступность теста и его исходного кода.*
- *Воспроизводимость.*

Методы оценки производительности

Наиболее известные тесты (бенчмарки):

- *Linpack*
- *STREAM*
- *Ливерморские циклы*
- *Perfect Club Benchmarks*
- *SPEC*
- *HINT*
- *NAS Parallel Benchmarks (NPB)*
- *HPC Challenge*
- *Graph500*
- *HPCG*

Закон Амдала, его следствия. Граф алгоритма. Критический путь графа алгоритма, ярусно-параллельная форма графа алгоритма. Этапы решения задач на параллельных вычислительных системах.

Закон Амдала, его следствия

Закон Амдала

(формулируется просто, но влияет сильно)

p – число процессоров (ядер, вычислительных узлов),

T_1 – время работы программы на одном процессоре,

T_p – время работы программы на системе из p процессоров,

$S = \frac{T_1}{T_p}$ – это ускорение работы программы при переходе с одного процессора на систему из p процессоров (во сколько раз программа начинает работать быстрее),

f – доля последовательных операций в исходной программе ($0 \le f \le 1$).

Закон Амдала.

Ускорение работы программы при переходе с одного процессора на систему из p процессоров можно оценить следующим образом:

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

На практике: $S < p$; идеальный случай: $S = p$ – *линейное ускорение работы программы*.

Закон Амдала. Следствие

$$S \approx \frac{1}{f}$$ *(при большом числе процессоров)*

На практике. Если доля последовательных операций в некоторой программе равна 0.1, значит вне зависимости от числа используемых процессоров ускорение не превысит 10.

Закон Амдала. Следствие

В теории. Для того чтобы ускорить программу в q раз, необходимо ускорить не менее, чем в q раз не менее, чем (1-1/q)-ю часть программы.

На практике. Нужно ускорить работу программы в 100 раз. Значит необходимо ускорить не менее, чем в 100 раз не менее, чем 99% этой программы.

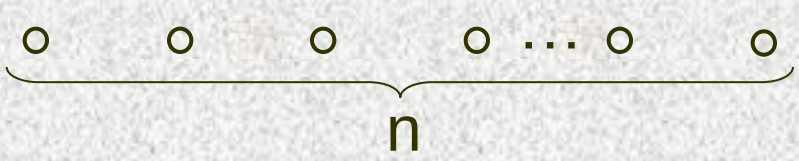
Граф алгоритма

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];
}
```



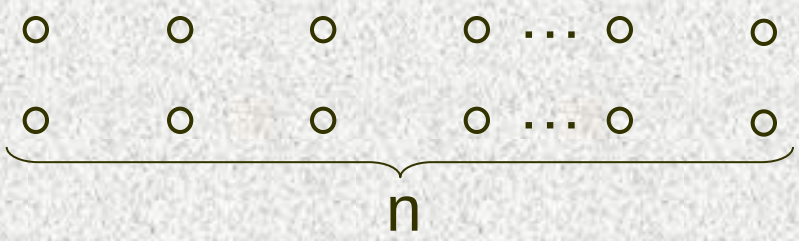
Каждая вершина соответствует двум операторам (телу цикла), выполненным на одной и той же итерации цикла.

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.

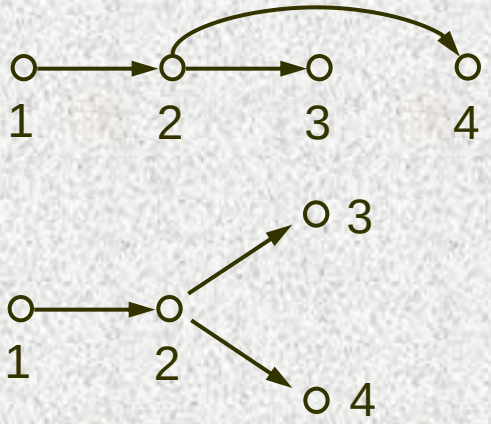
Информационное отношение = отношение по передаче данных.

Графовые модели программ

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

Дуги: информационное отношение:

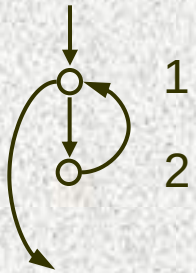
x(i) = a + b(i) (1)
y(i) = 2*x(i) – 3 (2)
t1 = y(i)*y(i) + 1 (3)
t2 = b(i) – y(i)*a (4)



Четыре основные модели программ

Граф управления программы.
Вершины: операторы
Дуги: операционное отношение

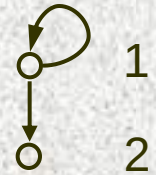
```
for( i = 0; i < n; ++i) {
    A[i] = A[i – 1] + 2;          (1)
    B[i] = B[i] + A[i];           (2)
}
```



Четыре основные модели программ

Информационный граф программы.
Вершины: операторы
Дуги: информационное отношение

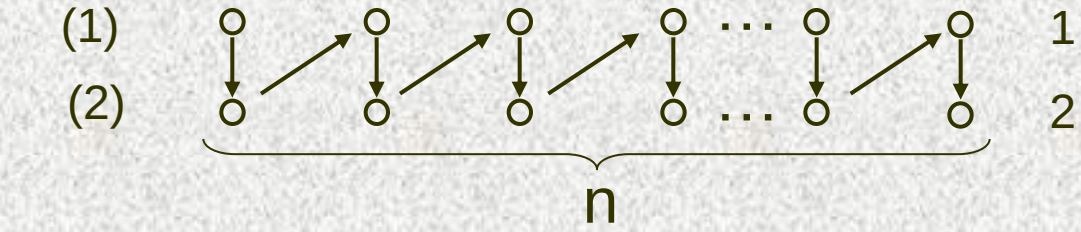
```
for( i = 0; i < n; ++i) {
    A[i] = A[i – 1] + 2;          (1)
    B[i] = B[i] + A[i];           (2)
}
```



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];
}
```

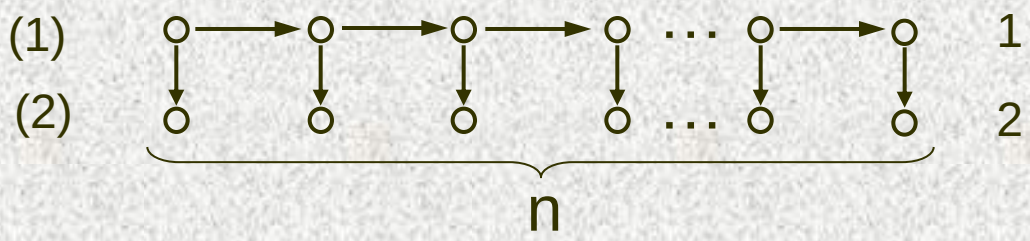


Свойства операционной истории:
- одна начальная вершина, у которой нет входящей дуги,
- одна конечная вершина, у которой нет исходящей дуги,
- у всех остальных вершин есть ровно одна входящая дуга и одна исходящая дуга.

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.

Какое отношение выбрать для описания свойств программ?

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

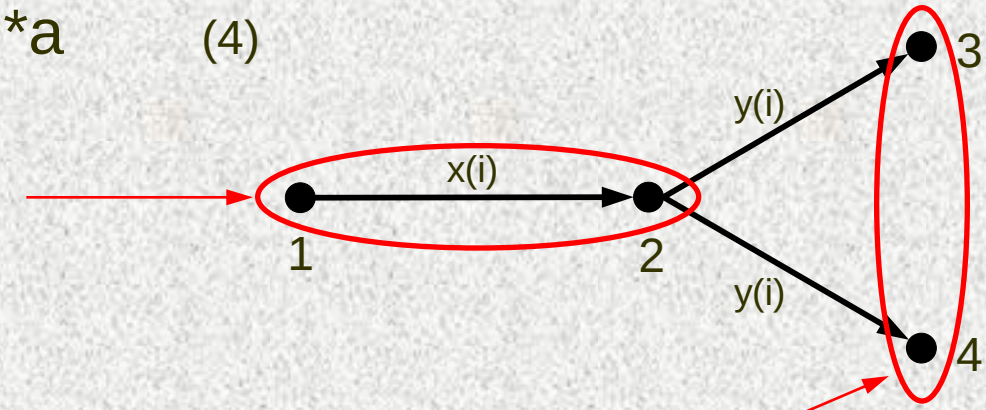
$$x(i) = a + b(i) \quad (1)$$

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

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

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

Исполнять только последовательно !

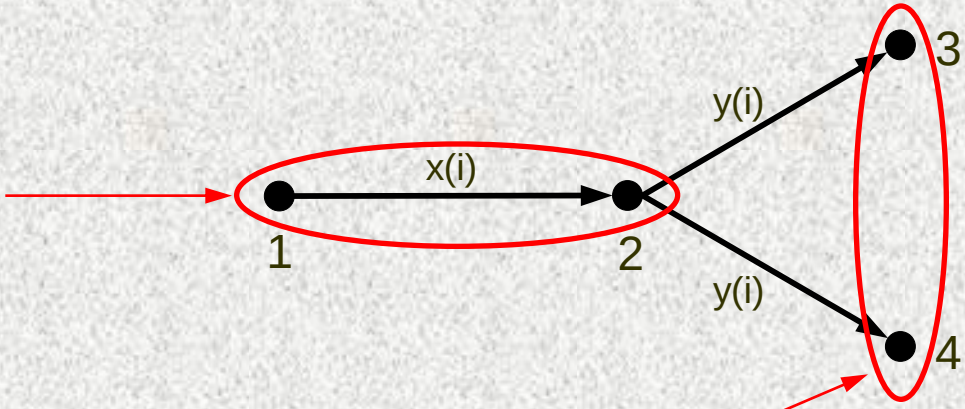


Можно исполнять параллельно !

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?

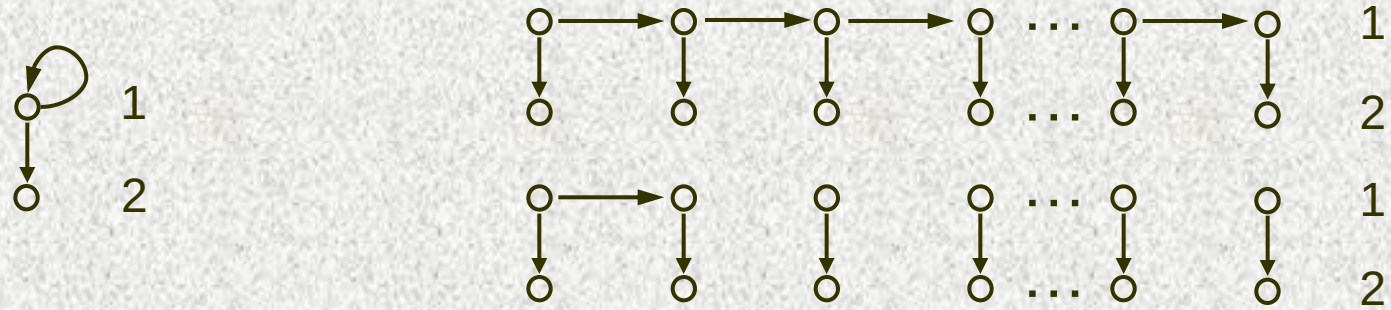
Аргументы для выбора степени компактности модели:

- *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?

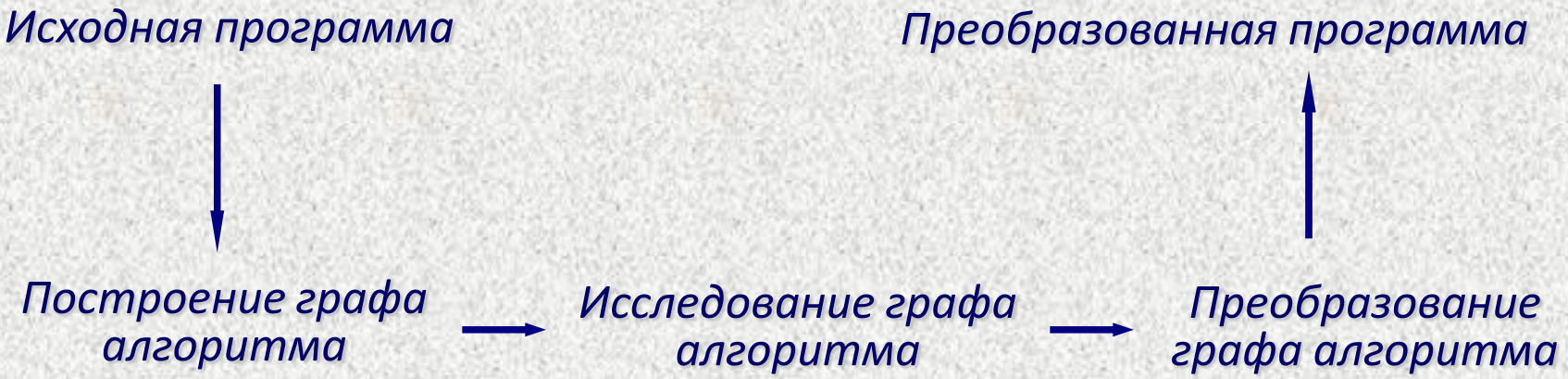
Аргументы для выбора степени компактности модели:

- *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



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

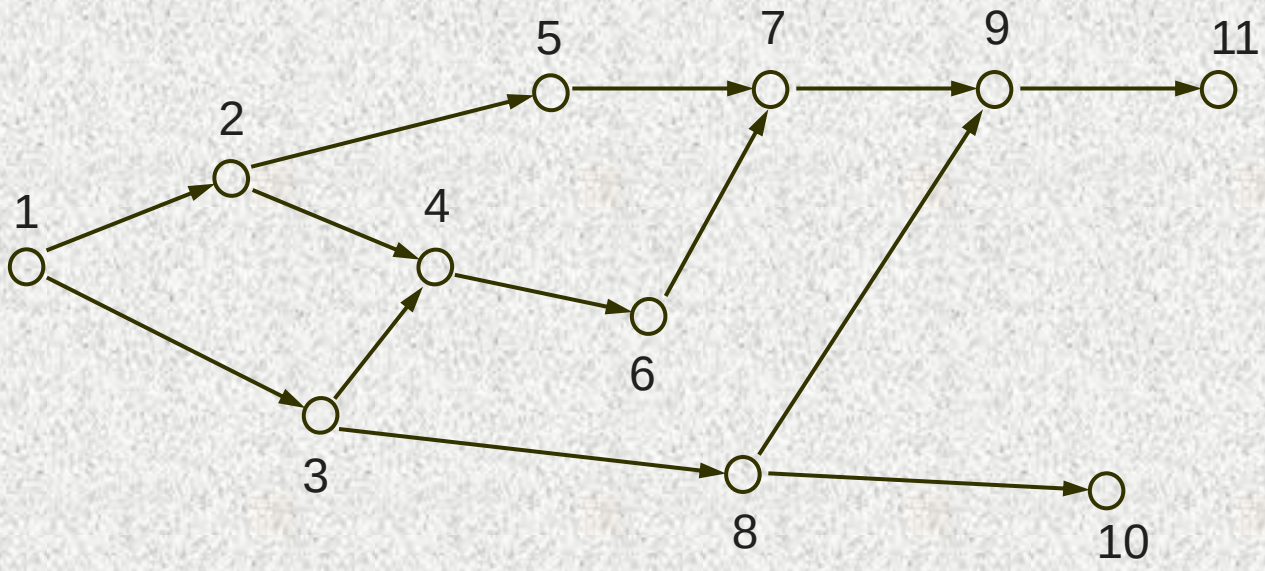
входные данные программы

срабатывания операторов

информационное отношение

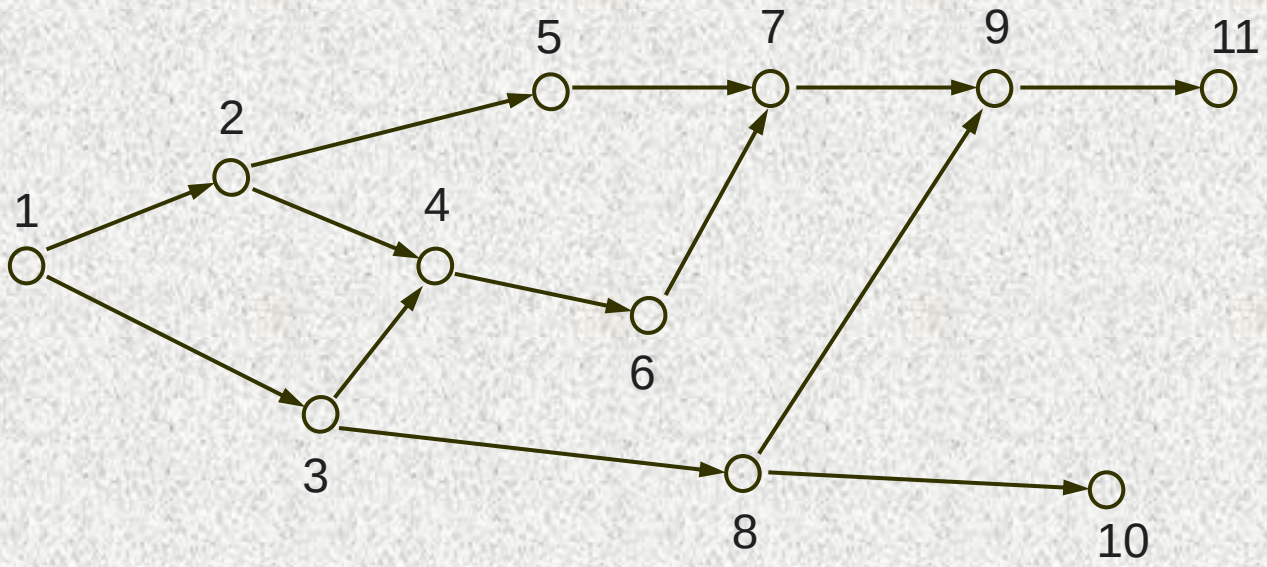
Критический путь графа алгоритма, ярусно-параллельная форма графа алгоритма

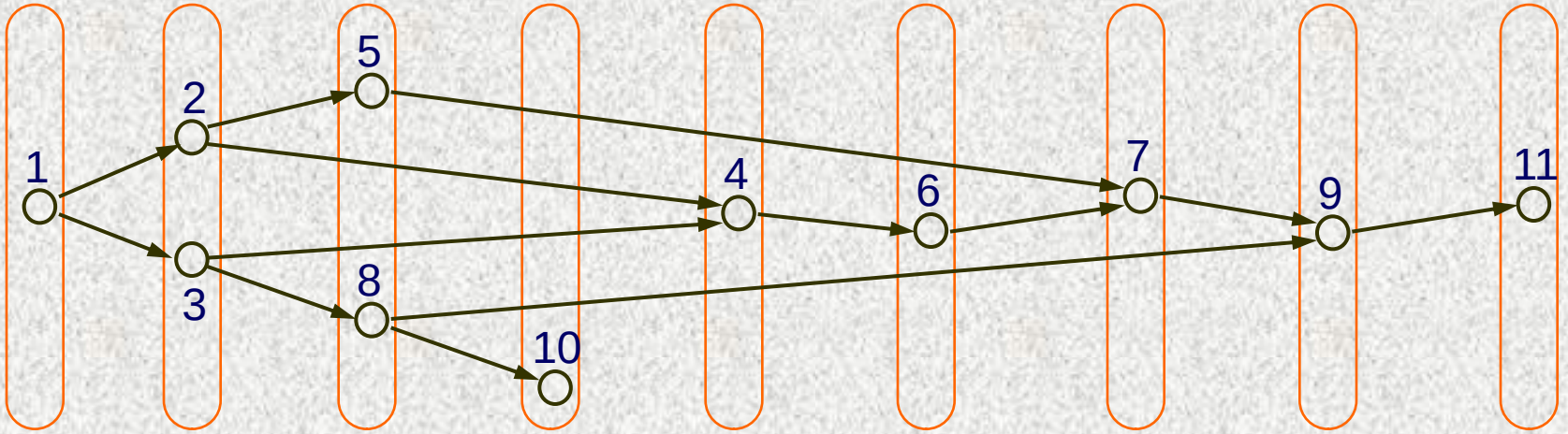
Ярусно-параллельная форма графа алгоритма



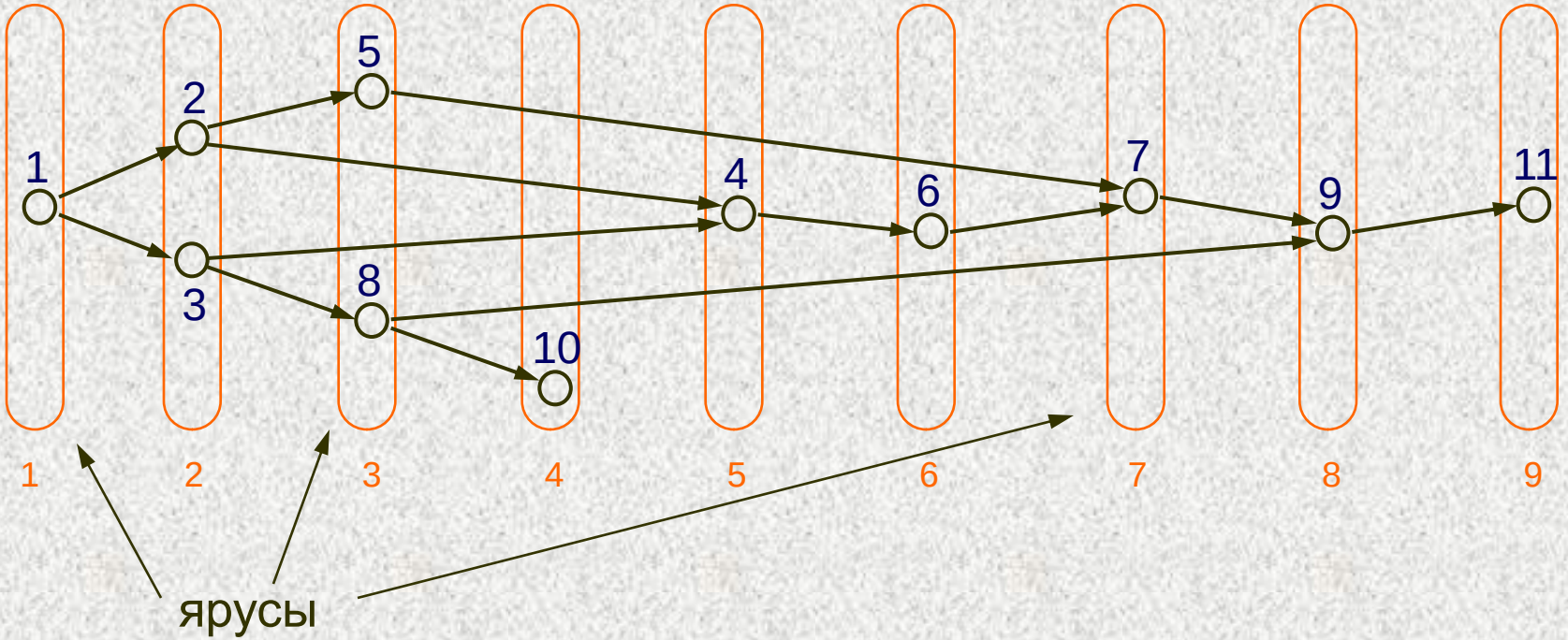
Как определить и сделать понятным ресурс параллелизма в графе алгоритма (в программе, в алгоритме) ?

Ярусно-параллельная форма графа алгоритма





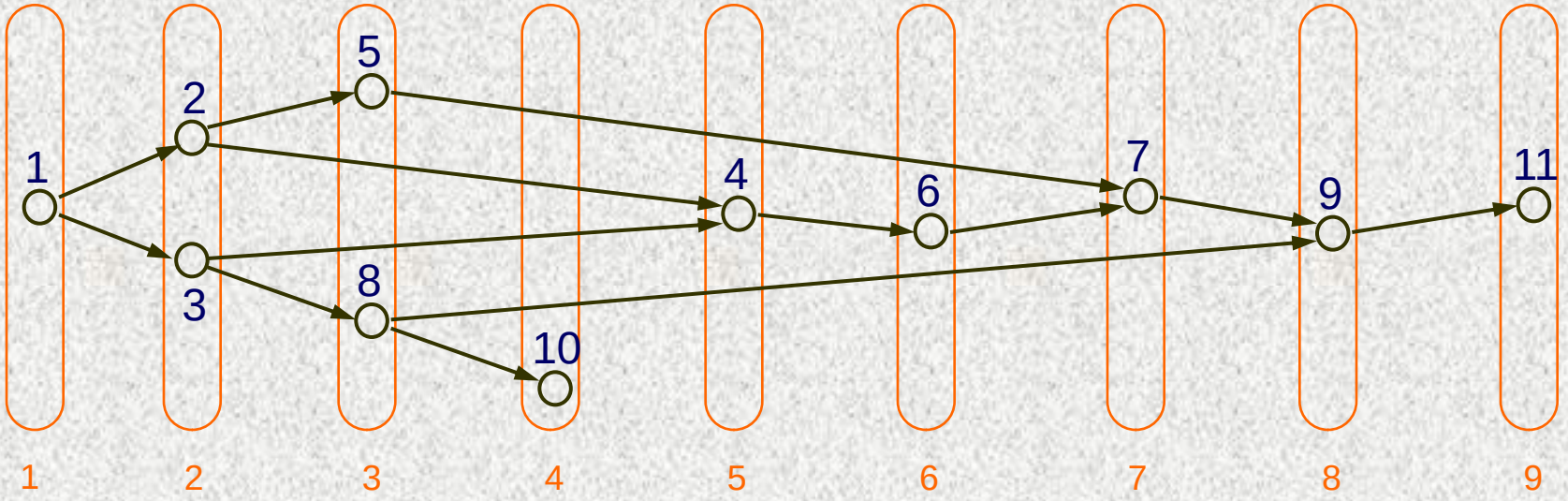
Ярусно-параллельная форма графа алгоритма



- 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



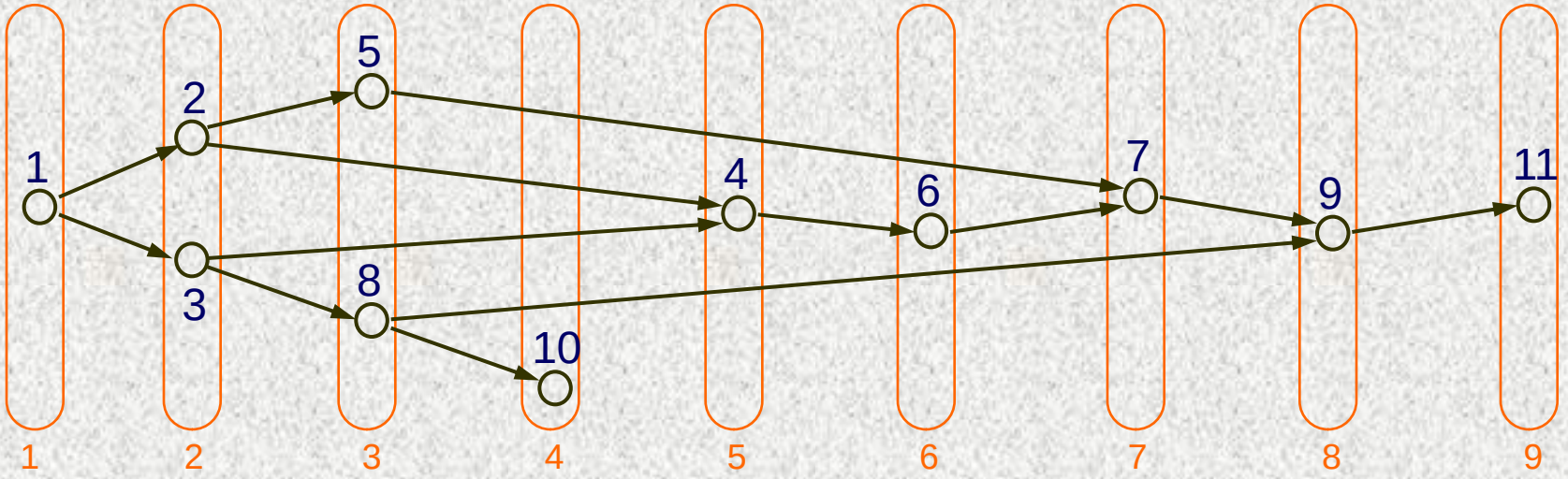
Высота ЯПФ – это число ярусов,

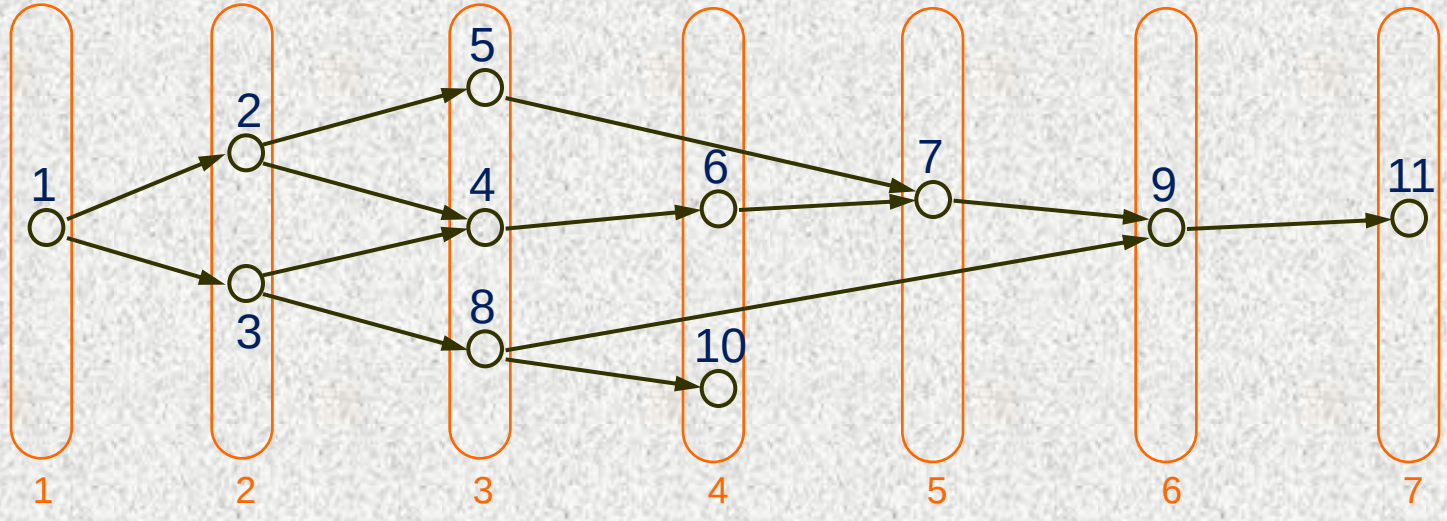
Ширина яруса – число вершин, расположенных на ярусе,

Ширина ЯПФ – это максимальная ширина ярусов в ЯПФ.

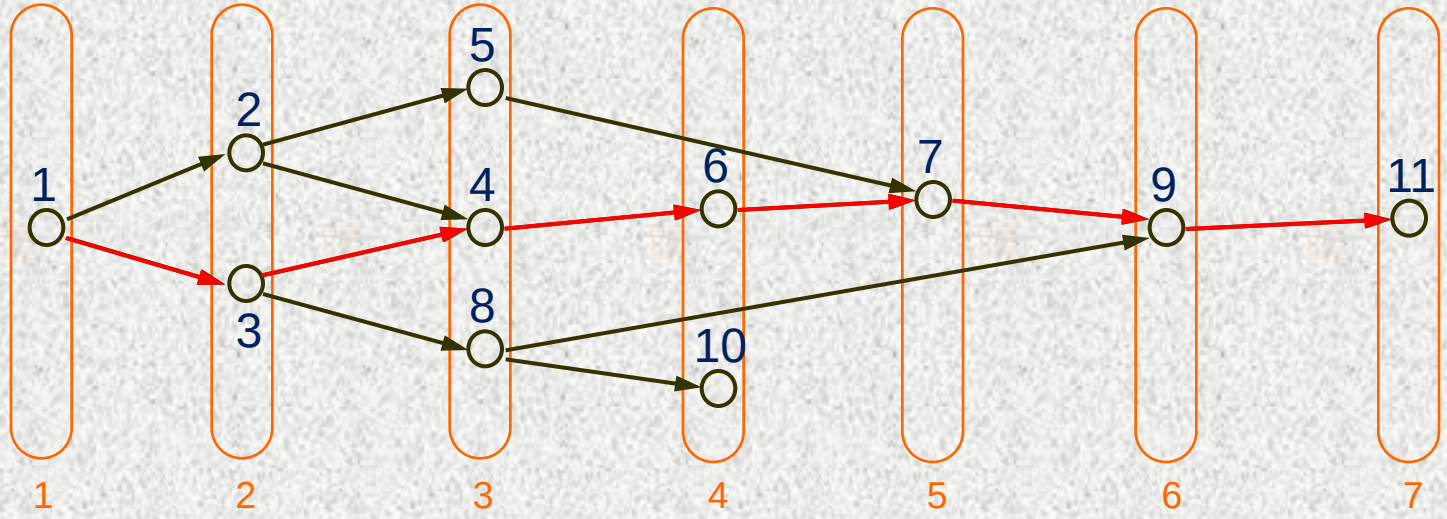
Высота ярусно-параллельной формы - это сложность параллельной реализации алгоритма/программы.

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.

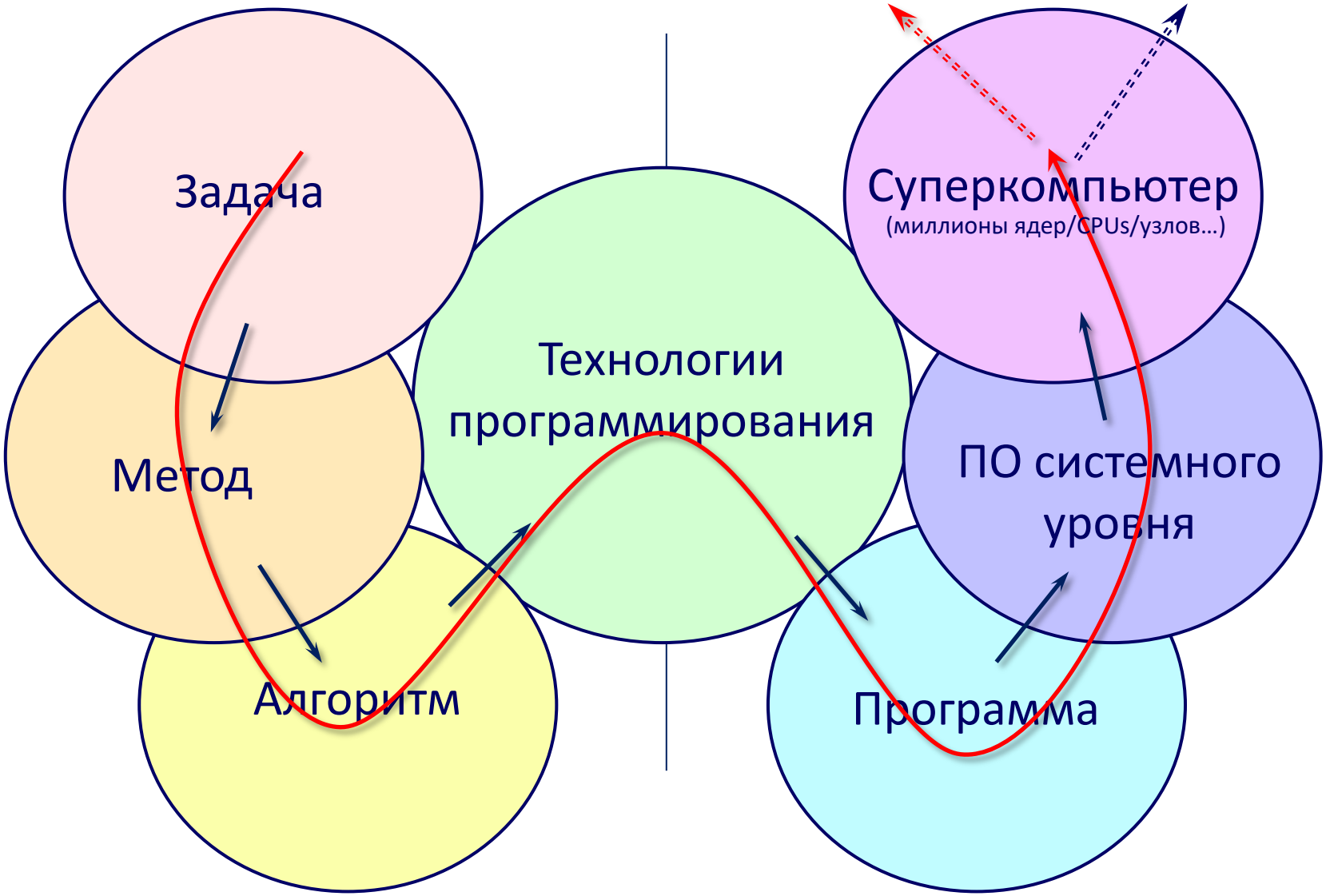
Этапы решения задач на параллельных вычислительных системах

Решение задач и вычислительные системы

(Реальность)
Реальная
производительность

Огромный
разрыв !

(Ожидания)
Пиковая
производительность



Алгоритмическая сторона

Компьютерная сторона

Особенность современных компьютеров – это высокая степень параллельности.

Решение задачи на параллельной вычислительной системе будет эффективным только в том случае, если на всем пути от Задачи до Компьютера нет ни одного “узкого места”!

Центральная проблема:
отображение программ и алгоритмов на архитектуру параллельных вычислительных систем (co-design, кодизайн)

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

СУПЕРКОМПЬЮТЕРЫ И ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ
(консультация к гос. экзамену)

А.С.Антонов

Вед. н.с. НИВЦ МГУ, к.ф.-м.н.

asa@parallel.ru

ВМК МГУ, 2026