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

(часть 2)

Вл.В.Воеводин

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

voevodin@parallel.ru

ВМК МГУ, 2020

Множество графовых моделей программ

(опорные точки)

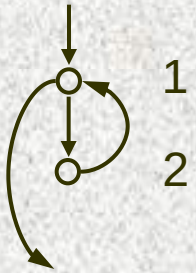
Операционные модели

Информационные модели

Компактные модели

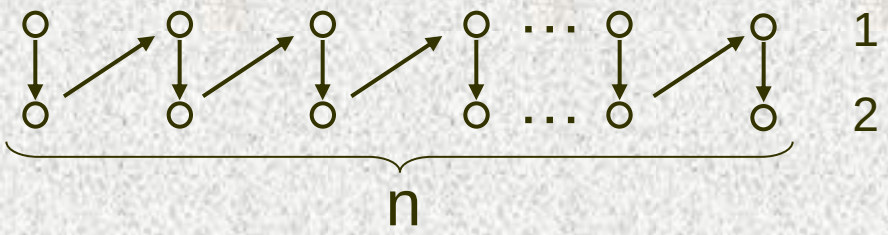
Истории

Граф управления

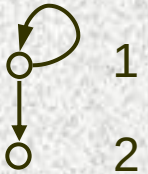


…

Операционная история

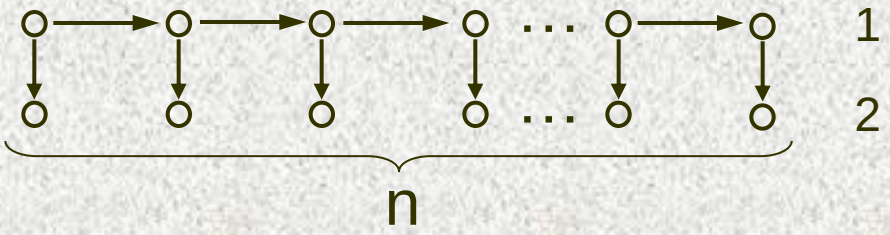


Информационный граф



…

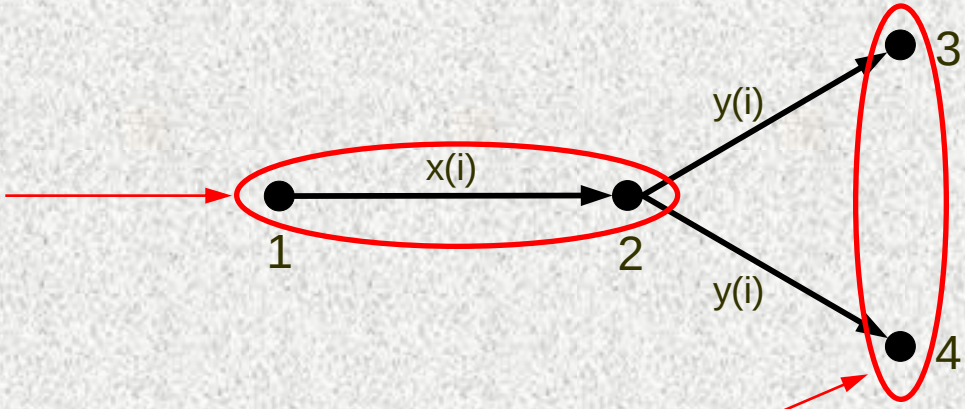
Информационная история



Какое отношение выбрать для описания свойств программ?

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?

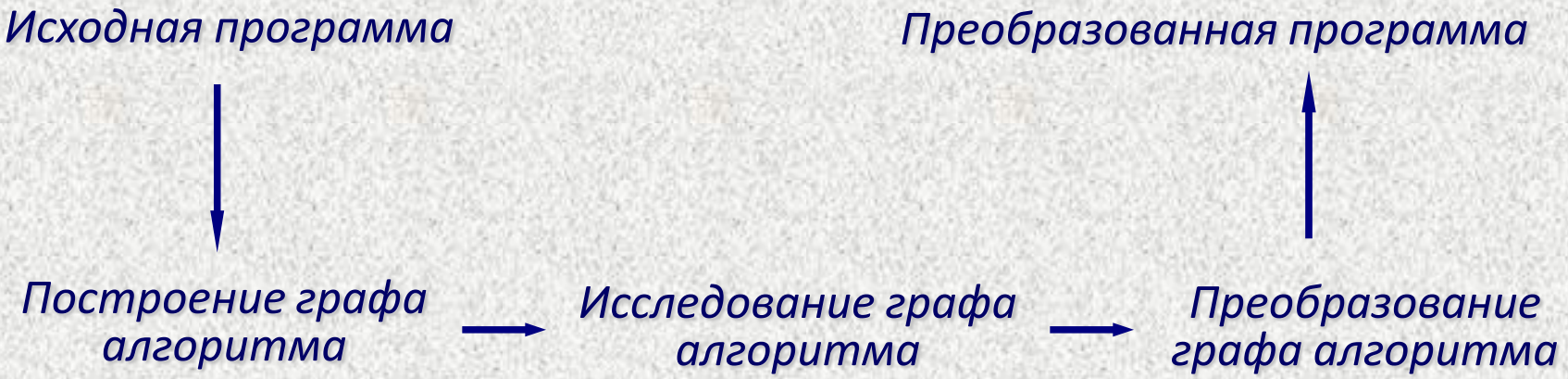
Аргументы для выбора степени компактности модели:

- *компактность описания,* *(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.*

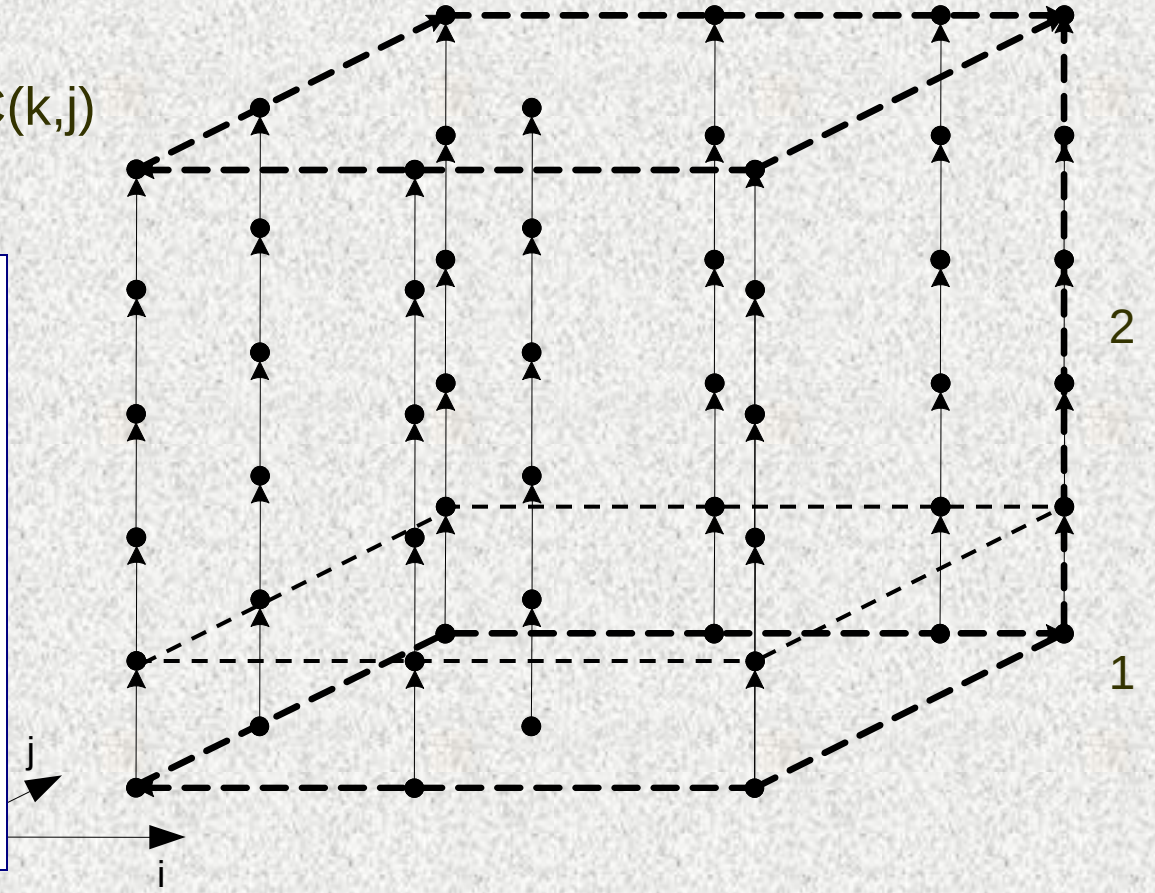
Общая схема анализа и преобразования структуры программ



Программы и их графы алгоритма
(умножение матриц)

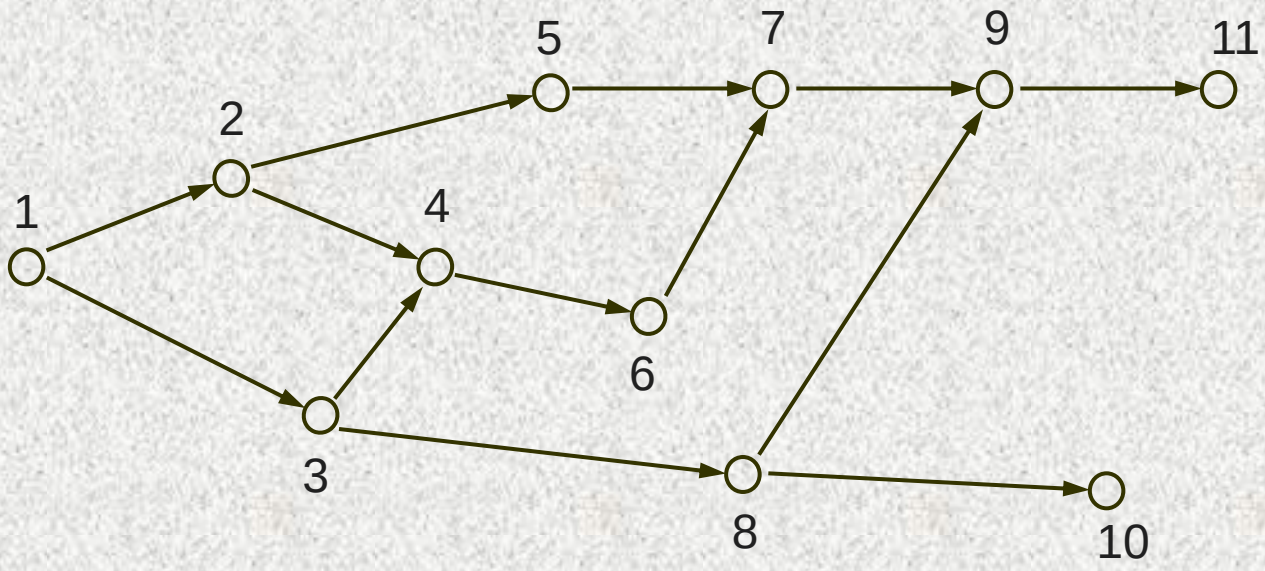
```
    Do i = 1, n
       Do j = 1, n
1      A(i,j) = 0
         Do k = 1, n
2          A(i,j) = A(i,j) + B(i,k)*C(k,j)
```

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



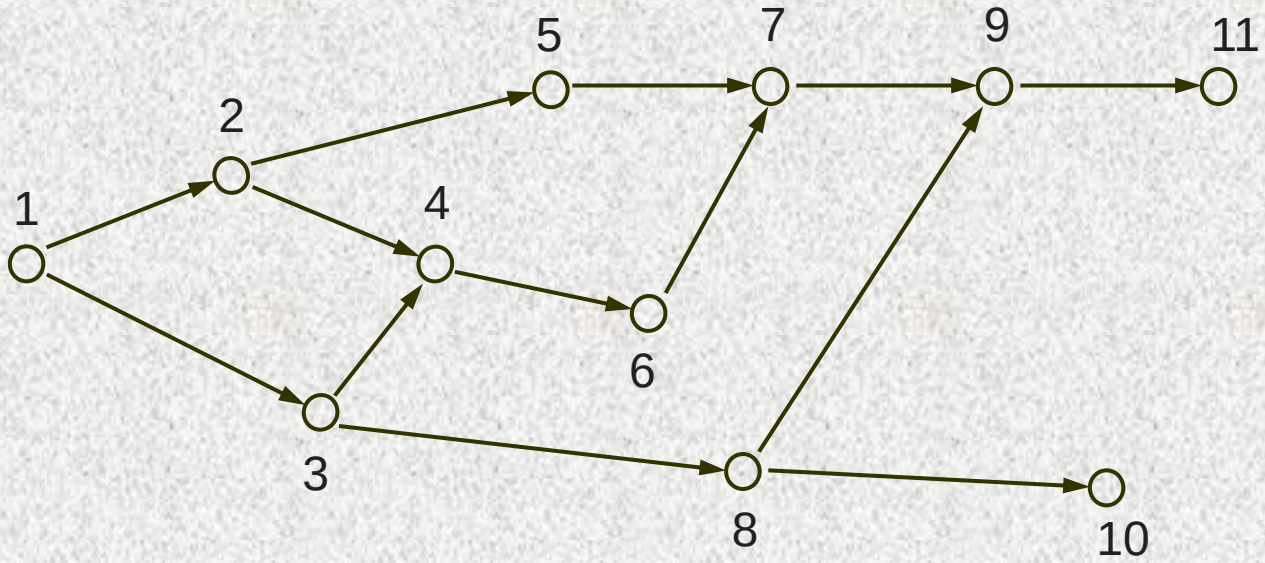
Как описать ресурс параллелизма программ и алгоритмов?

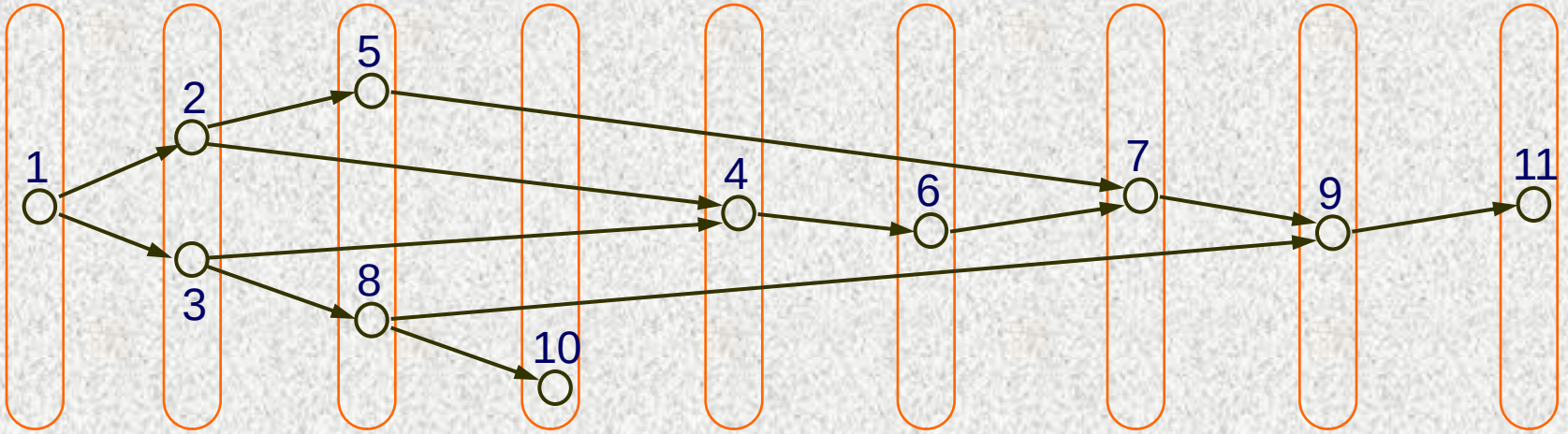
Ярусно-параллельная форма графа алгоритма



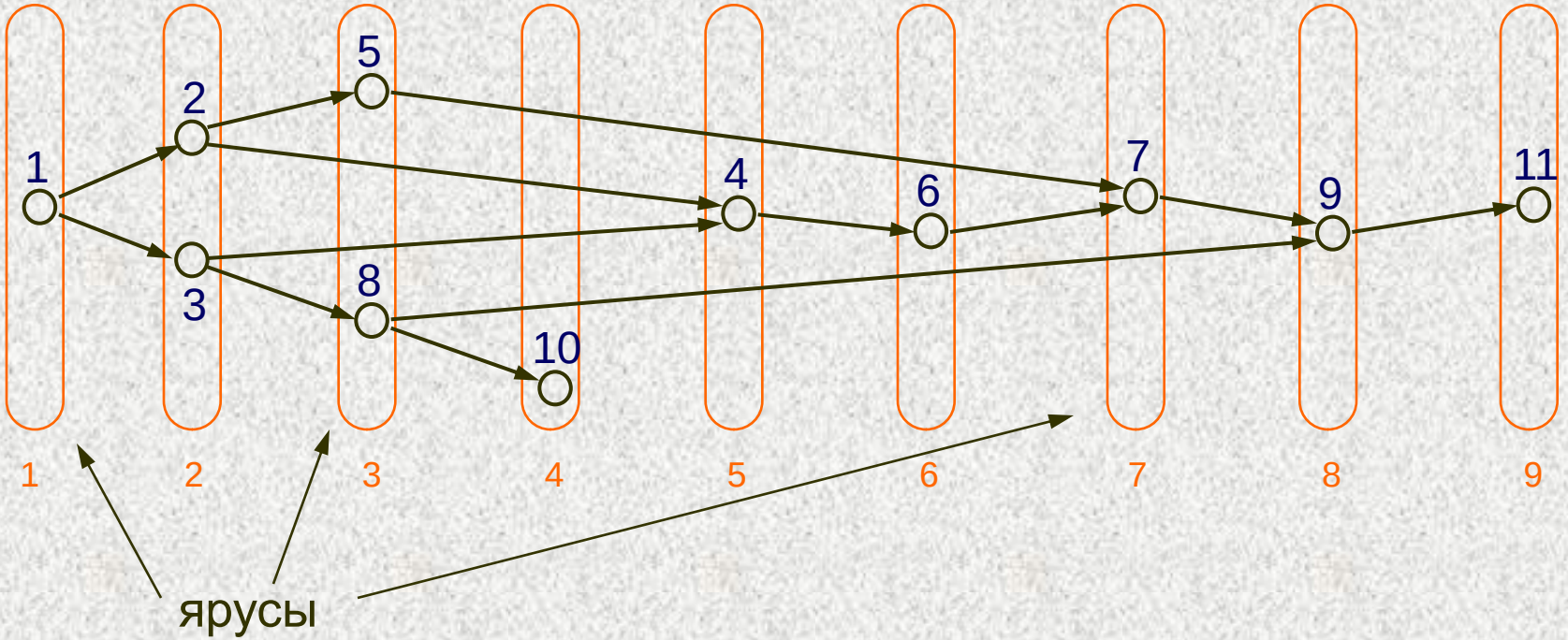
Как определить и сделать понятным ресурс параллелизма в графе алгоритма (в программе, в алгоритме) ?

Ярусно-параллельная форма графа алгоритма





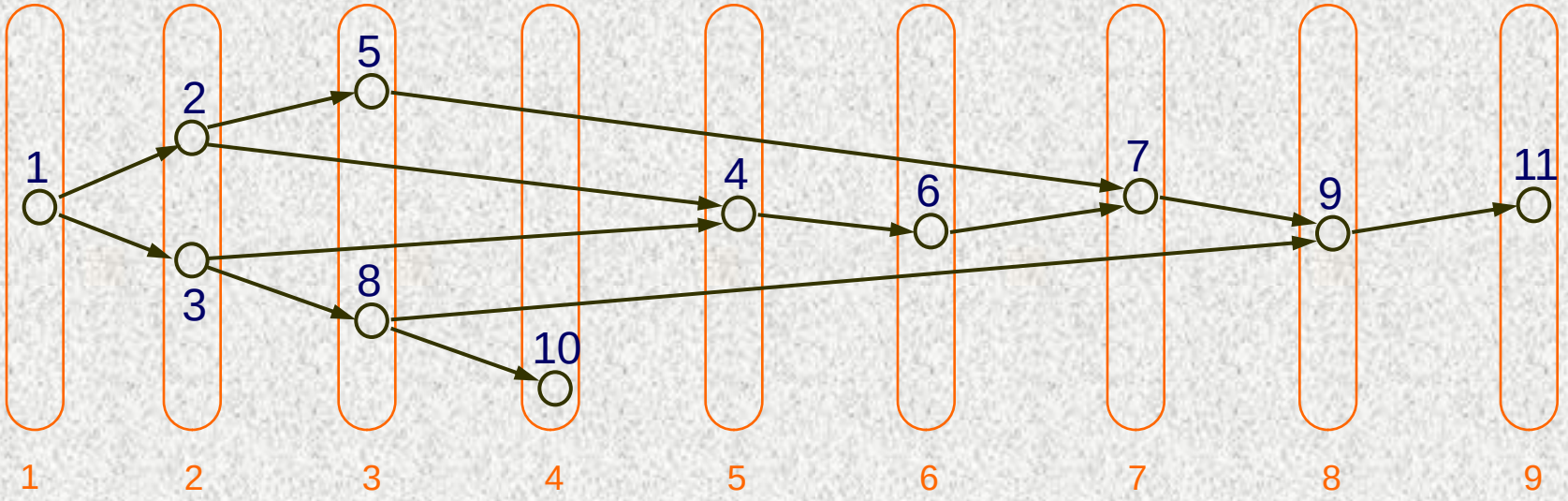
Ярусно-параллельная форма графа алгоритма



- 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



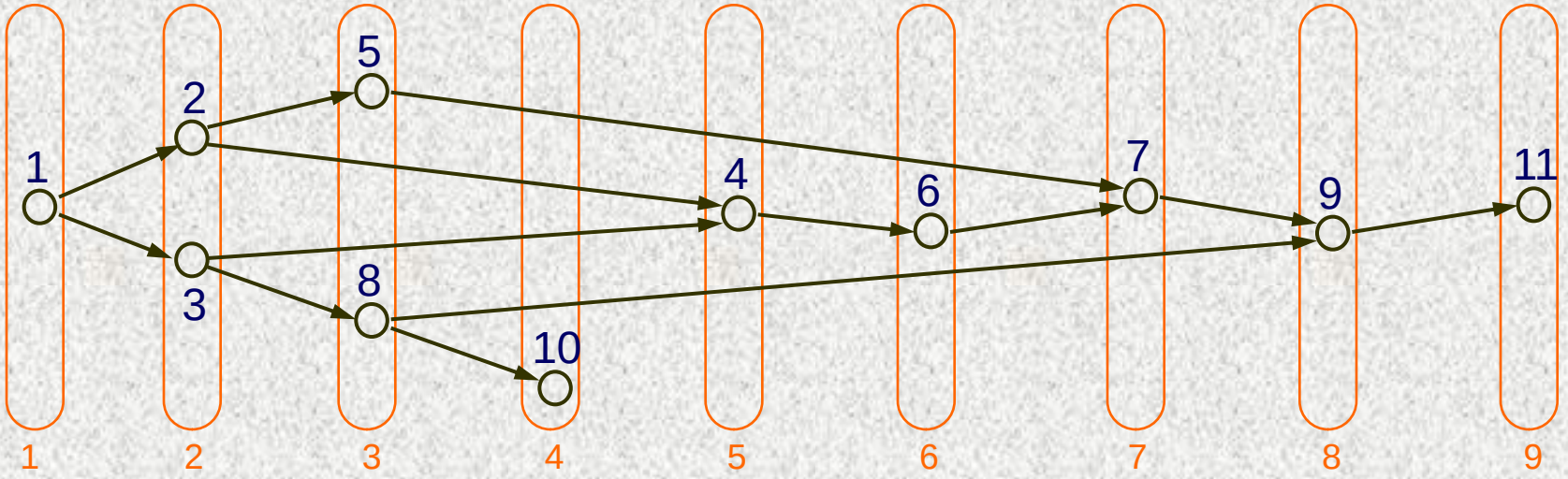
Высота ЯПФ – это число ярусов,

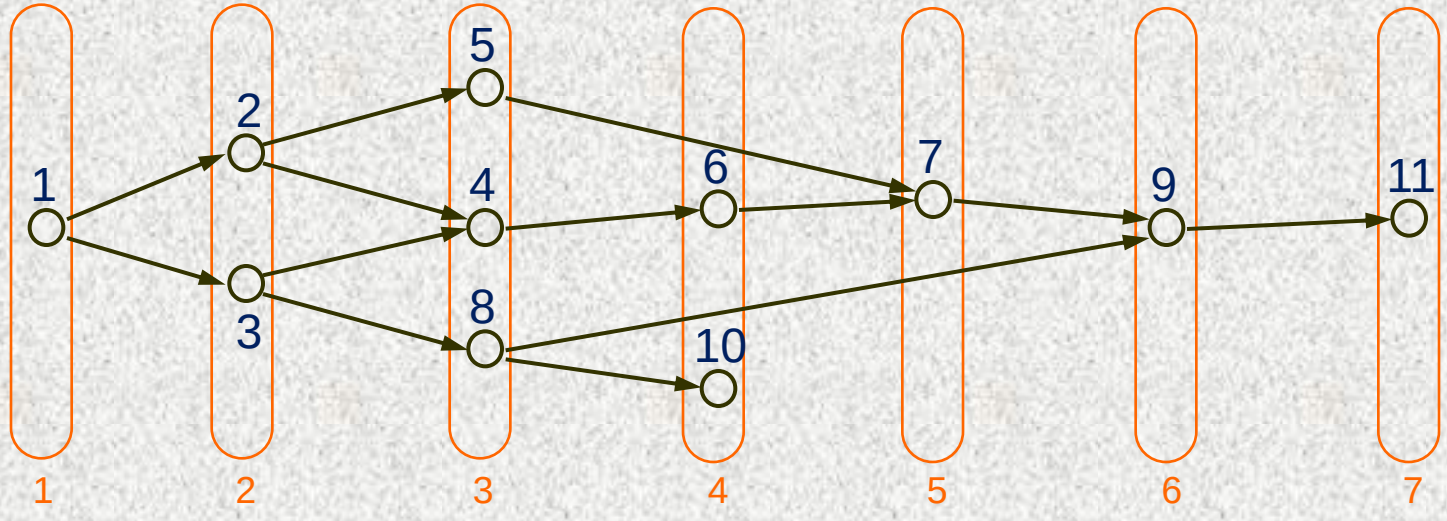
Ширина яруса – число вершин, расположенных на ярусе,

Ширина ЯПФ – это максимальная ширина ярусов в ЯПФ.

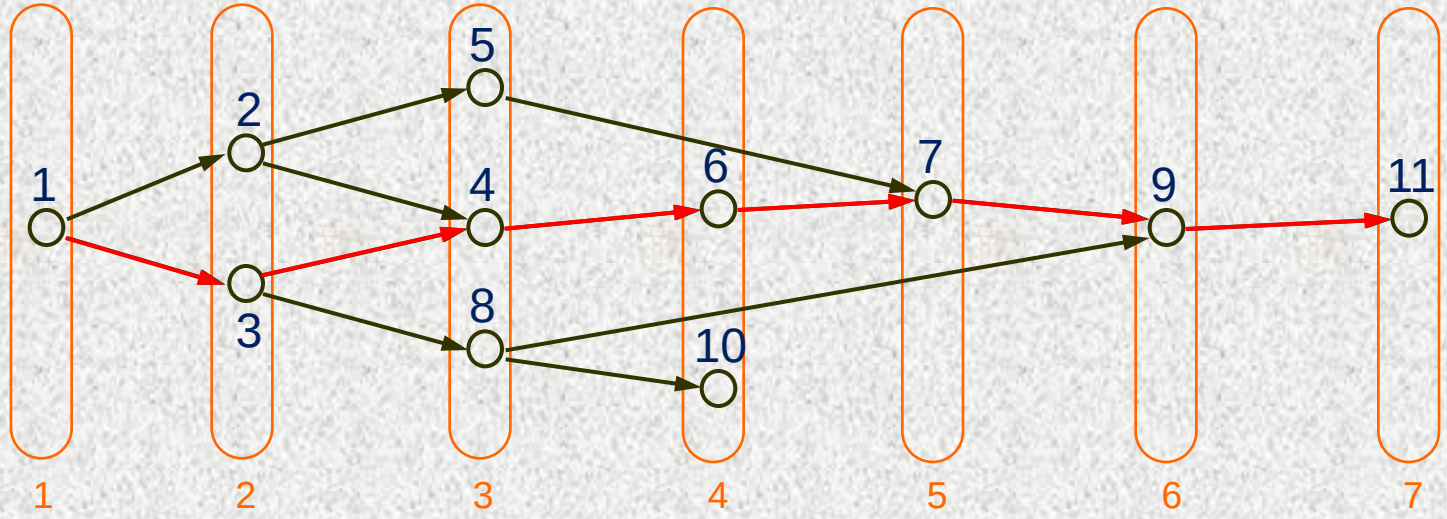
Высота ярусно-параллельной формы - это сложность параллельной реализации алгоритма/программы.

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.

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

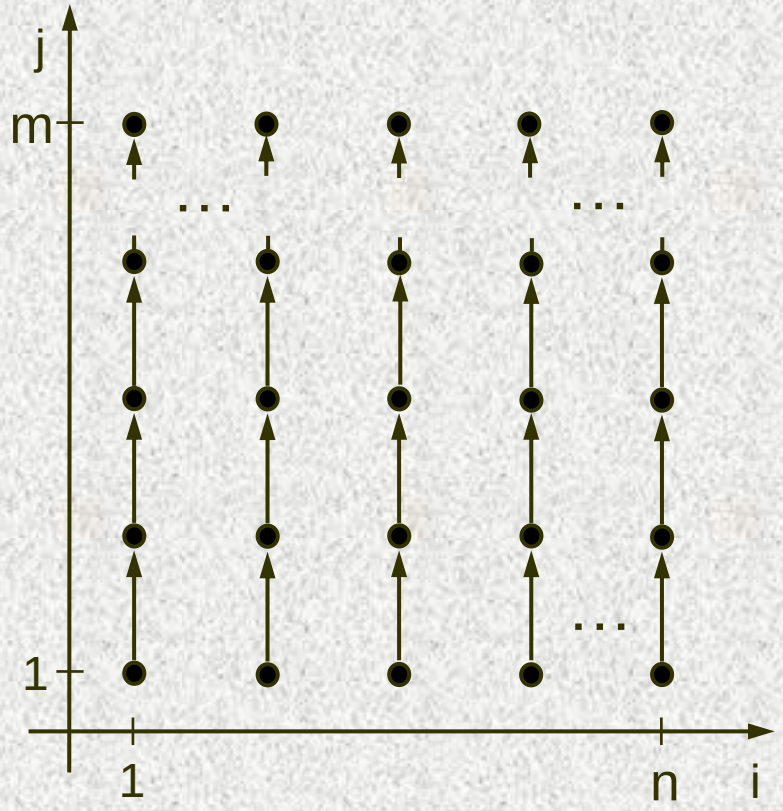
(каков у фрагмента ресурс параллелизма?)

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

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

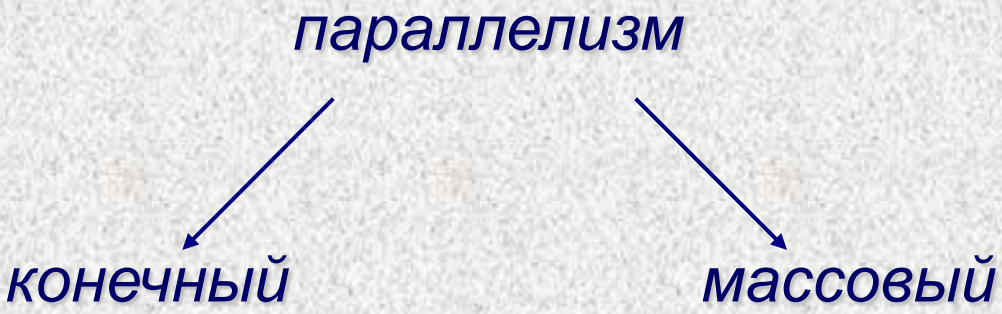
$$\alpha = \frac{\text{Число последовательных операций}}{\text{Общее число операций}} = \frac{m}{n*m} = \frac{1}{n}$$

$$S \approx n$$



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

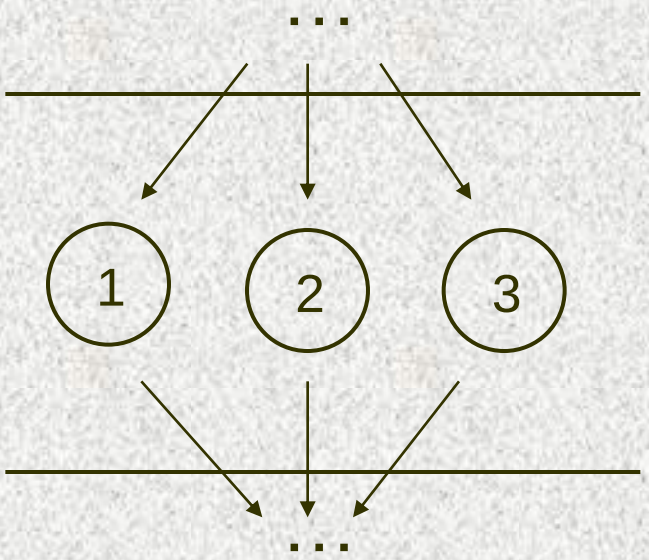
Конечный параллелизм.

```
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

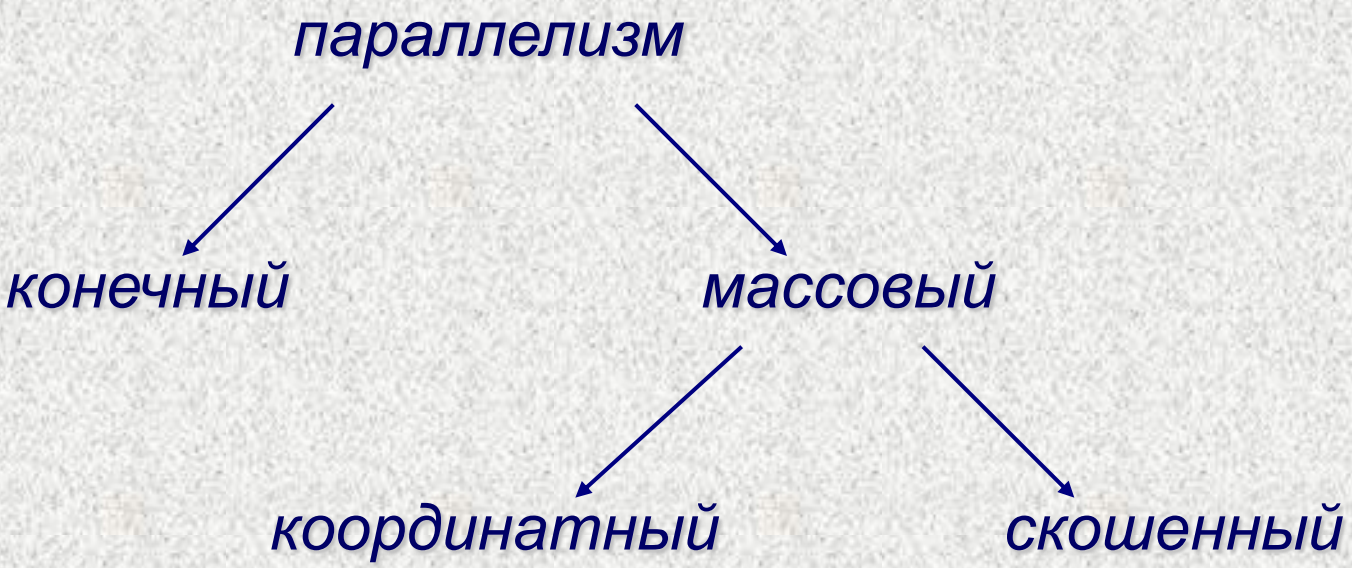


Виды параллелизма в алгоритмах и программах

Массовый параллелизм.

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

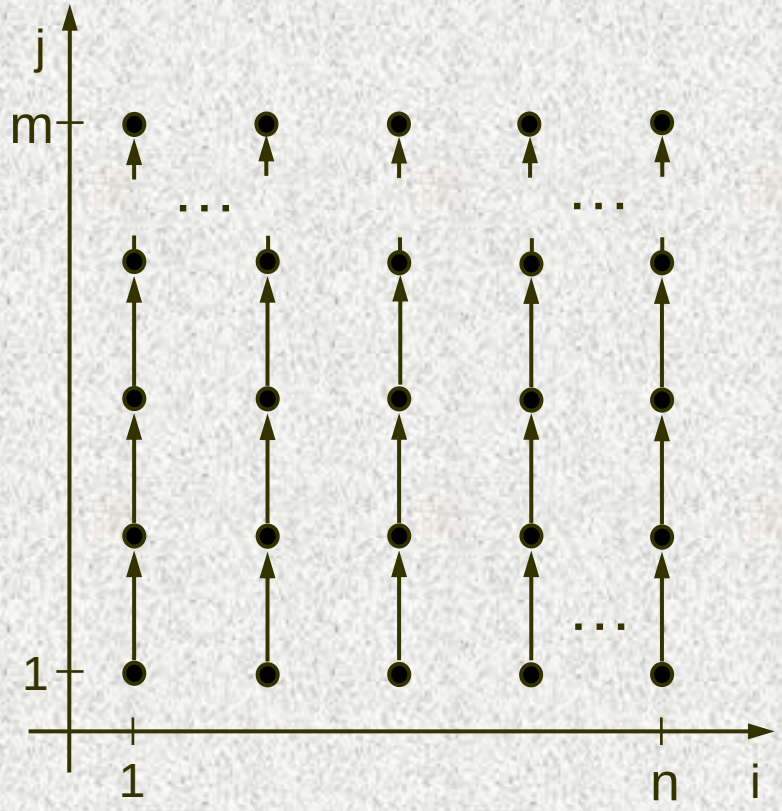
Виды параллелизма в алгоритмах и программах



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 *= {*$f_1 = i_1$*,*

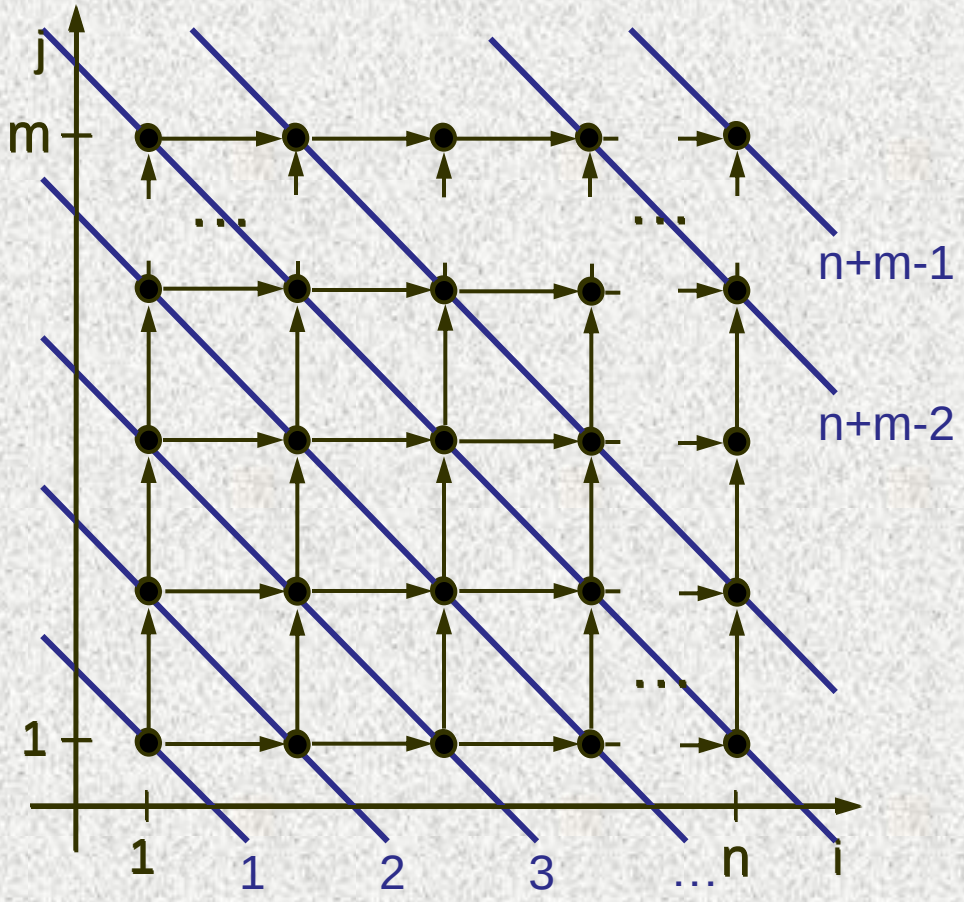
i_1 *— это параметр анализируемого цикла,*

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

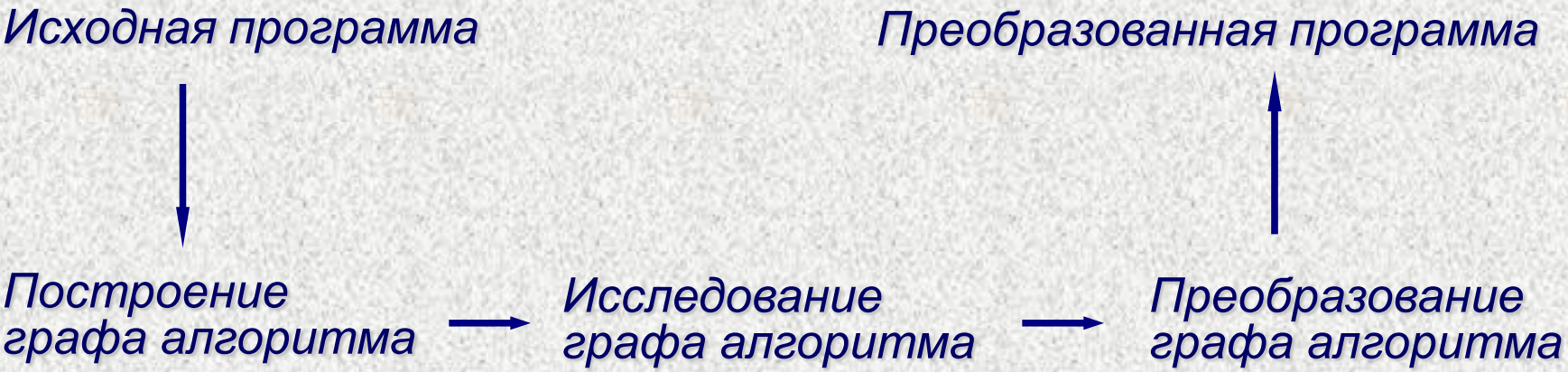
Виды параллелизма в алгоритмах и программах

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



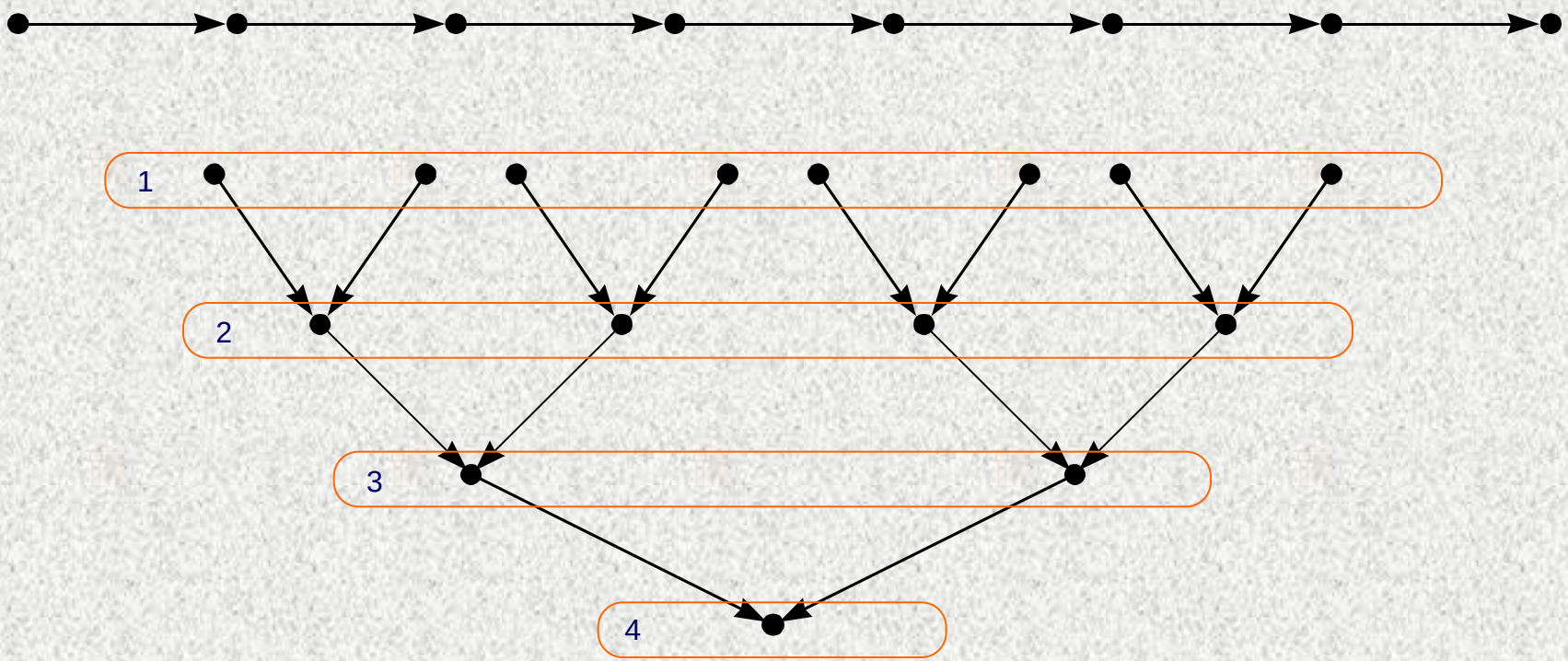
Эквивалентные преобразования программ

(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…

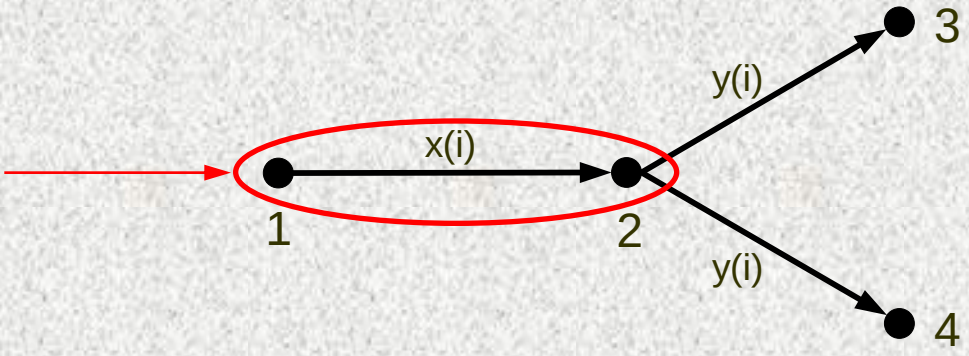
Эквивалентные преобразования программ
(суммирование элементов массива)



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.

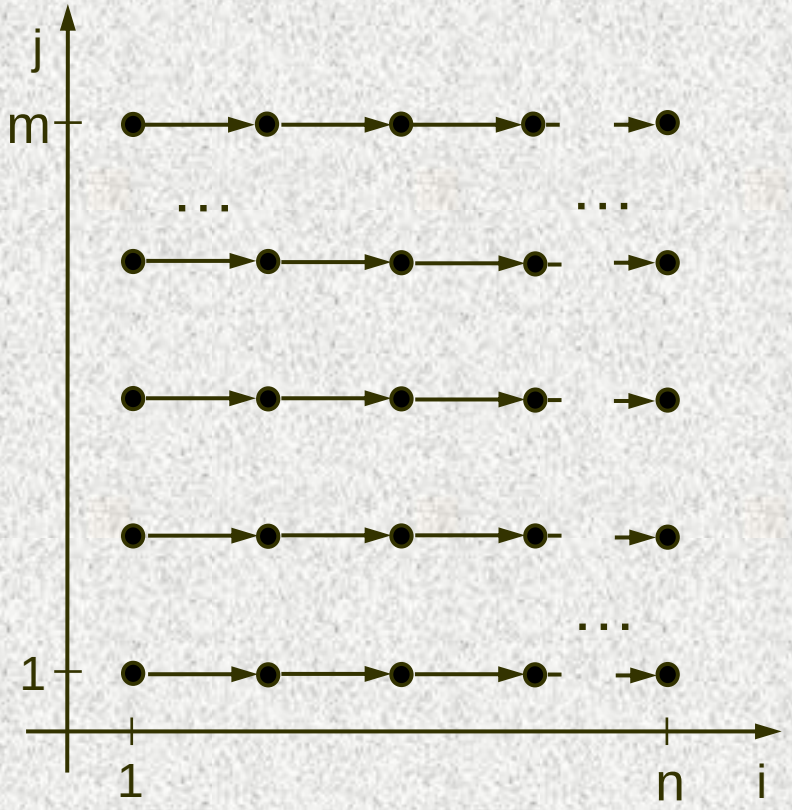
Информационная независимость определяет ресурс параллелизма программы.

Эквивалентные преобразования программ на практике…

Элементарные преобразования циклов

Перестановка циклов.

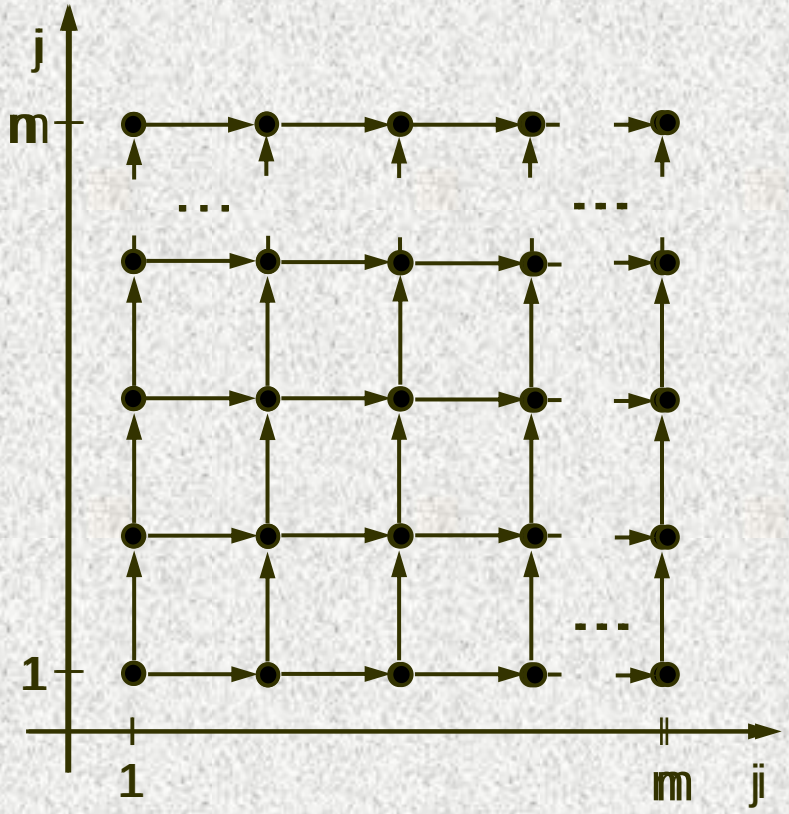
```
for( i = 0; i < n; ++i)
#pragma omp parallel for
    for( j = 0; j < m; ++j)
        A[i][j] = A[i-1][j] + C[i][j]*x;
```



Элементарные преобразования циклов

Перестановка циклов.

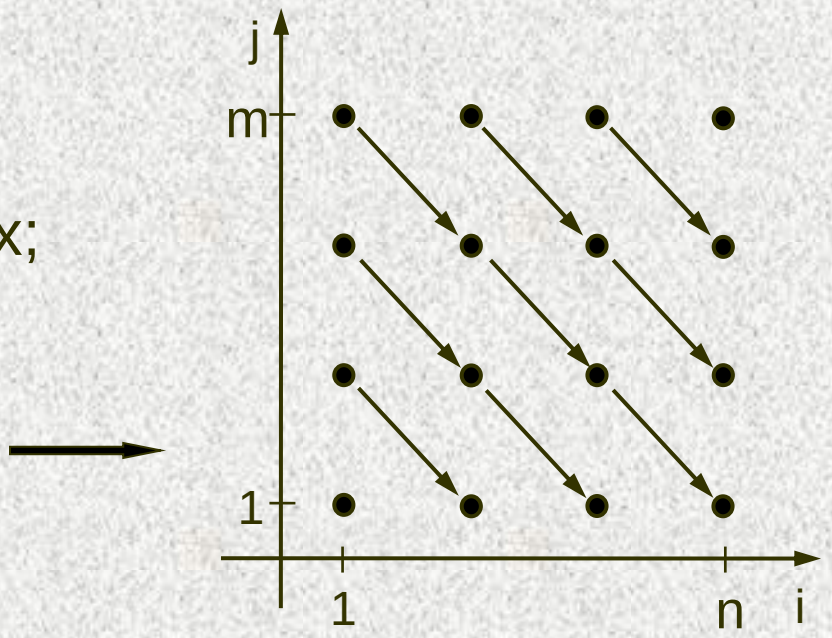
```
#pragma omp parallel for
for( i = 0; i < n; ++i)
    for( j = 0; j < m; ++j)
        A[i][j] = A[i–1][j] + C[i][j]*x;
```



Элементарные преобразования циклов

Всегда ли перестановка циклов является эквивалентным преобразованием?

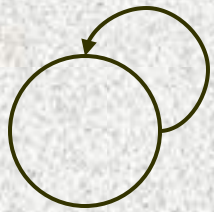
```
for( i = 0; i < n; ++i)
    for( j = 0; j < m; ++j)
        A[i][j] = A[i+c1][j+c2] + C[i][j]*x;

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.

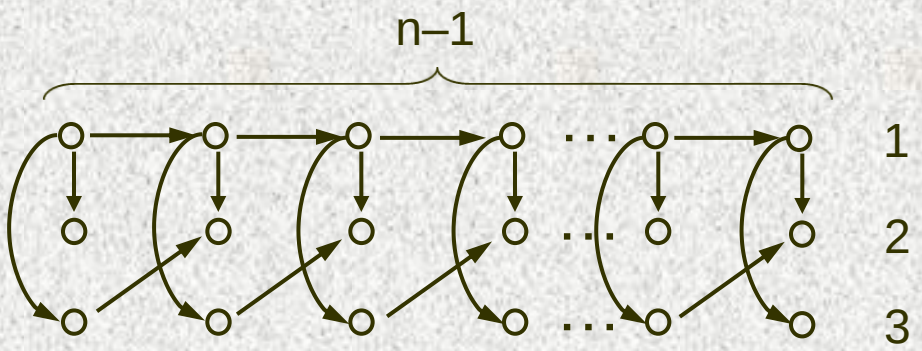
```
   for( i = 1; i < n; ++i) {
1      A[i] = A[i–1]*p + q;
2      C[i] = (A[i] + B[i–1])*s;
3      B[i] = (A[i] – B[i])*t;
   }
```



Элементарные преобразования циклов

Распределение циклов.

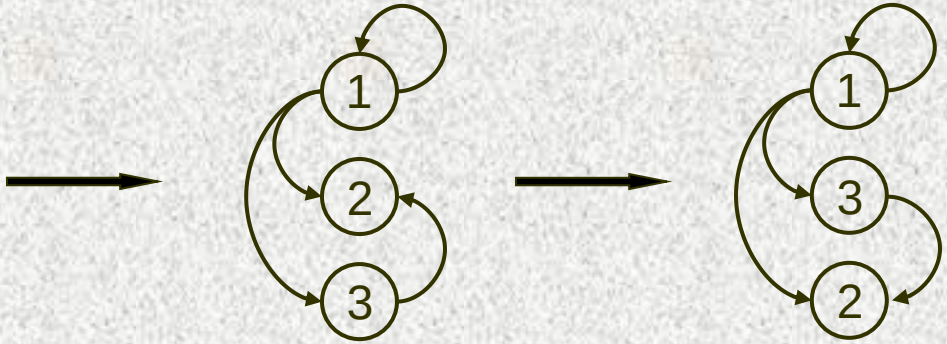
```
   for( i = 1; i < n; ++i) {
1      A[i] = A[i–1]*p + q;
2      C[i] = (A[i] + B[i–1])*s;
3      B[i] = (A[i] – B[i])*t;
   }
```



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.

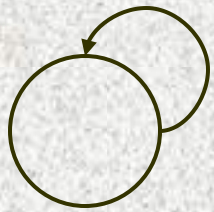
```
for( i = 1; i < n; ++i) {
    A[i] = A[i–1]*p + q;
    C[i] = (A[i] + B[i–1])*s;
    B[i] = (A[i] – B[i])*t;
}
```

```
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.

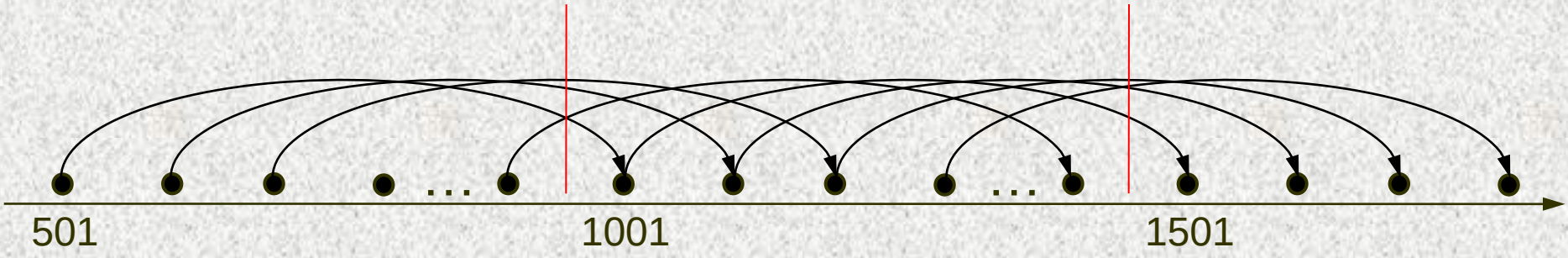
```
for( i = 501; i <= 2000; ++i)
    A[i] = A[i] + A[i–500];
```



Элементарные преобразования циклов

Расщепление циклов.

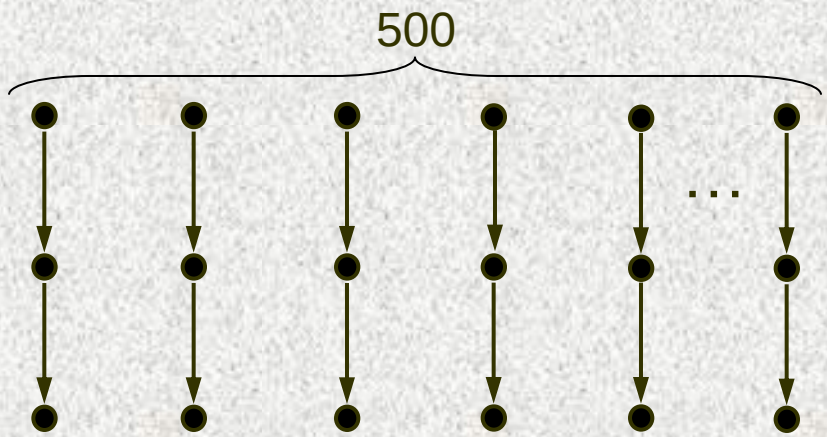
```
for( i = 501; i <= 2000; ++i)
    A[i] = A[i] + A[i–500];
```



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];
```

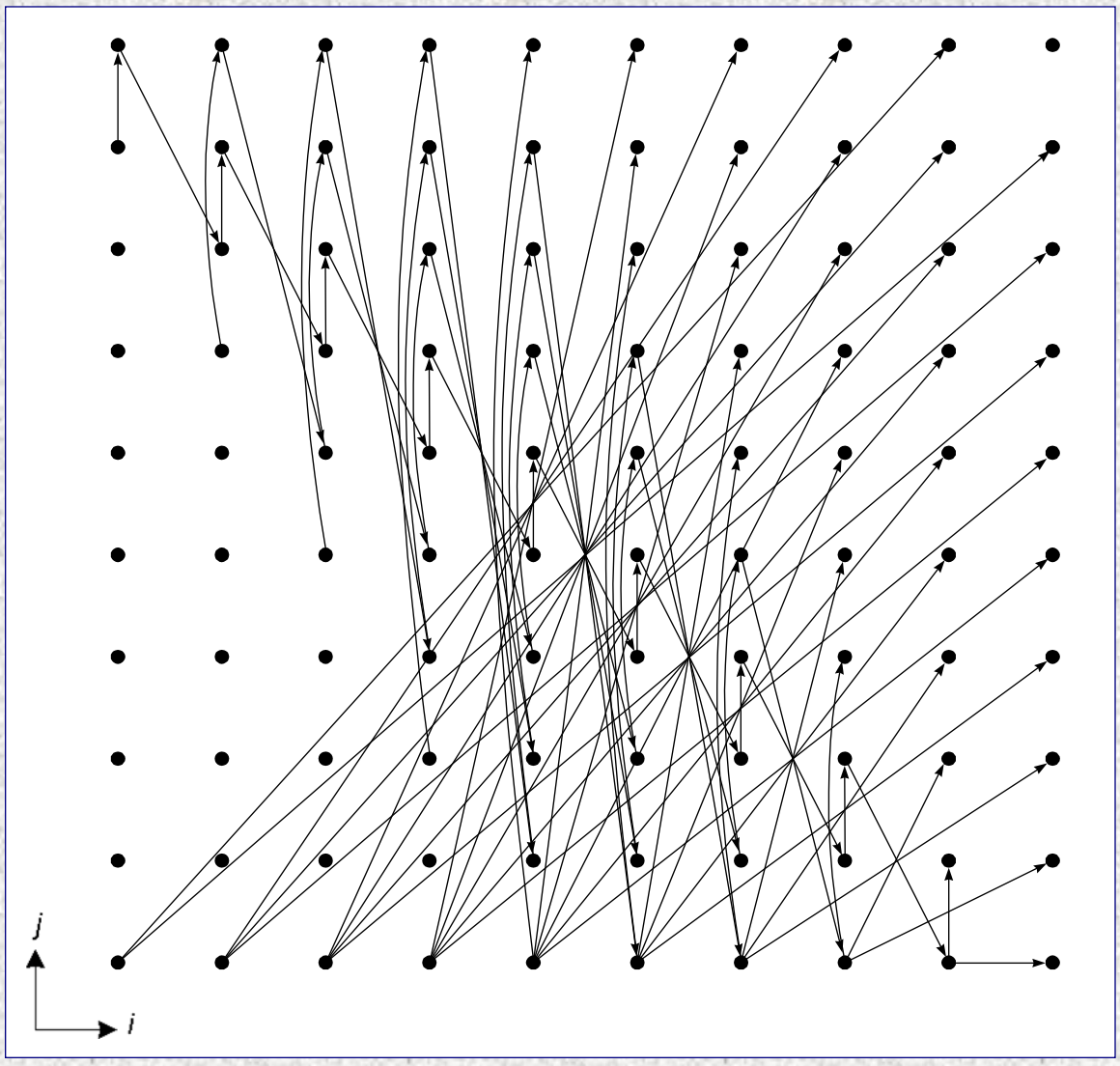
```
if( m == 3 )
    for( i = 0; i < n; ++i)
        R[i] = P[i] + D[i] * F[i];
else
     for( i = 0; i < n; ++i)
        R[i] = Q[i] – D[i] * F[i];
```

Означает ли малый размер программы простоту ее информационной структуры?

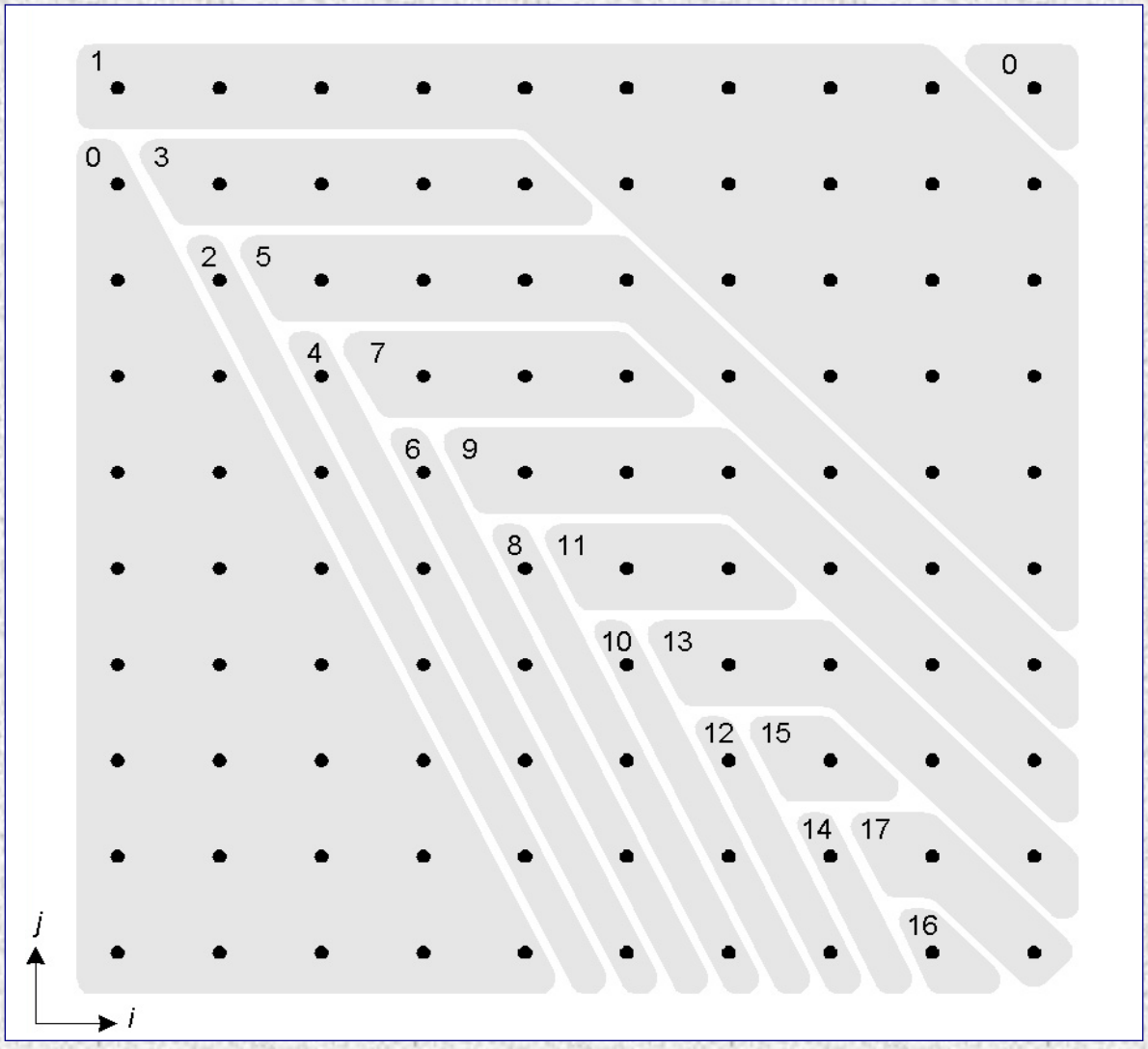
Простой пример…
(последовательный вариант)

```
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…



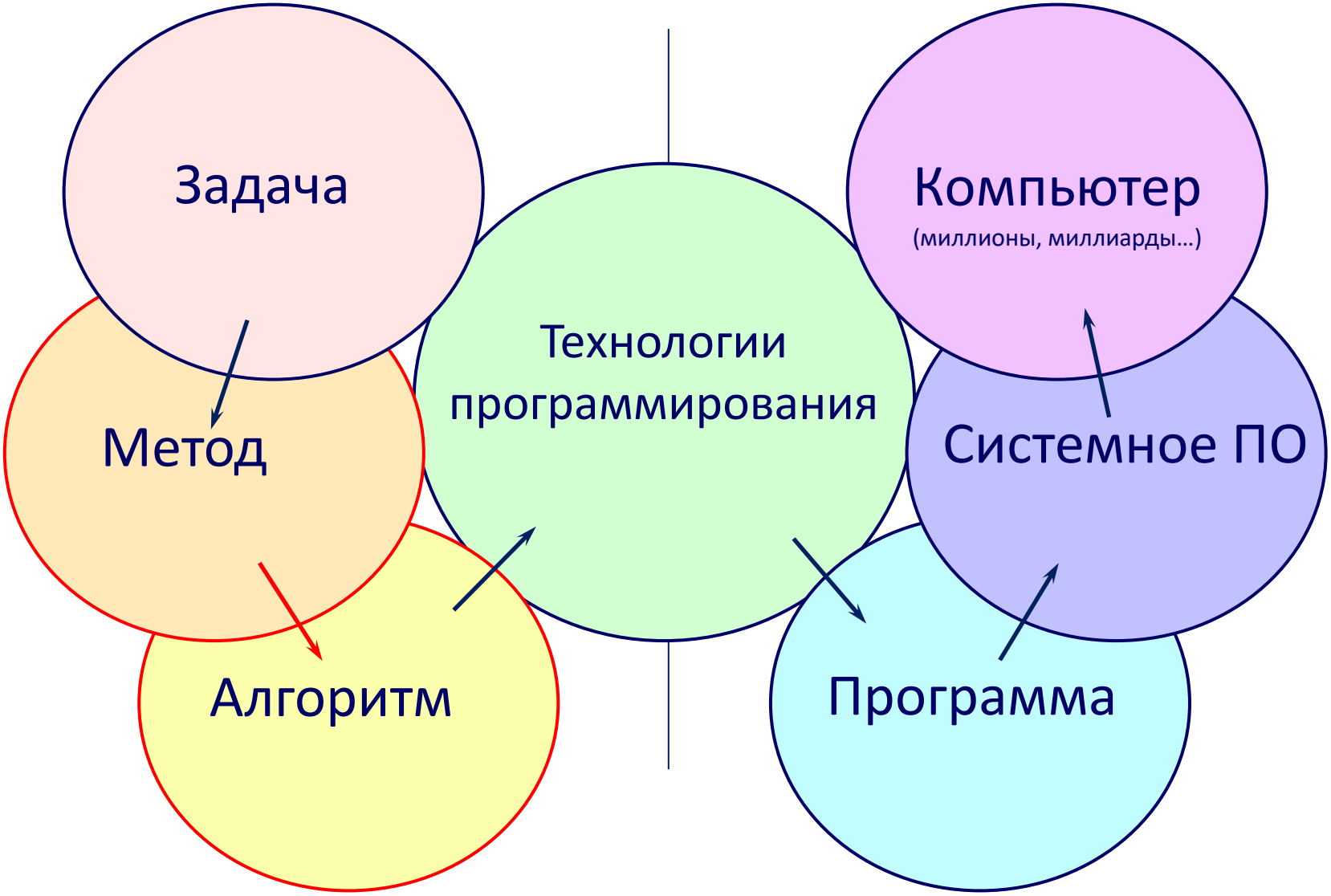
Совсем не простой пример…
(параллельный вариант)

```
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)

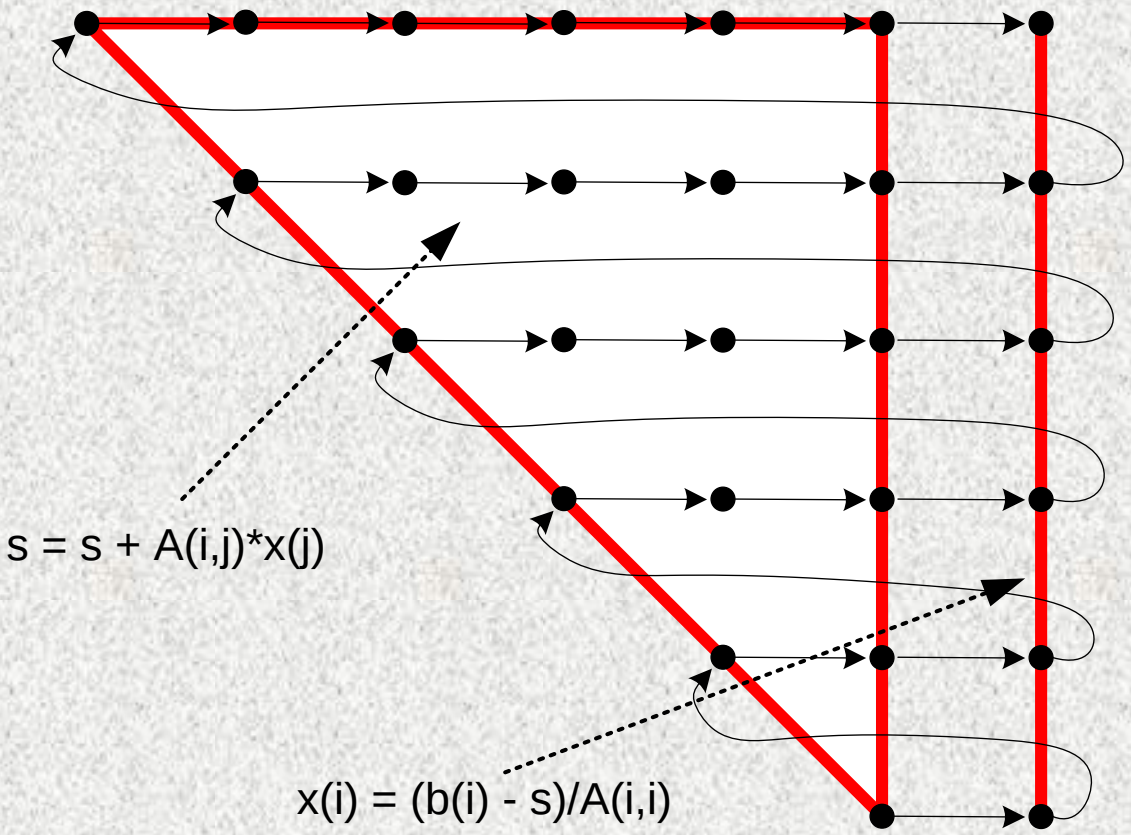


Схема программы:

```
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)

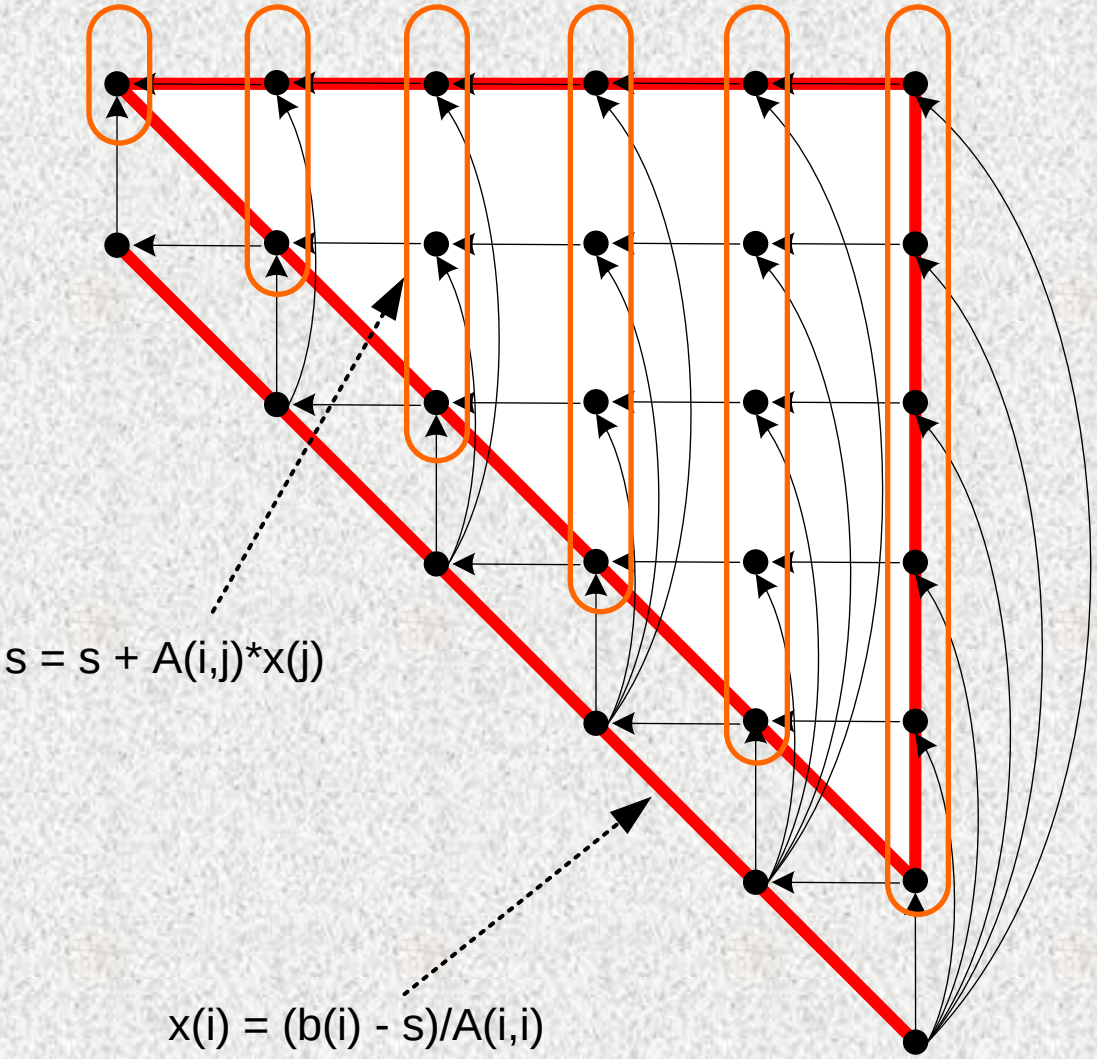


Схема программы:

```
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!

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

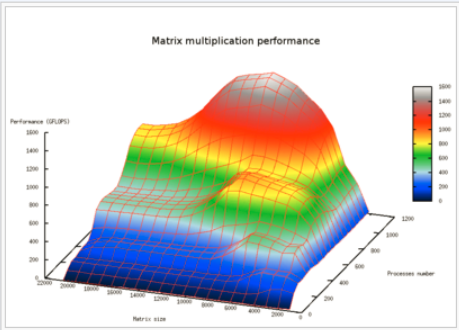
1.1 Общее описание алгоритма

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:

Предварительные объявления о зачете…

- *Список вопросов к зачету опубликован на https://parallel.ru/vmk20*

- *Консультация по курсу состоится 12 декабря, суббота, в 14 часов. Ссылку вышлем дополнительно.*

- *Итоги по конкурсу изображений, выступлениям в формате стенд-ап и выполнению практических заданий будут подведены к 12 декабря.*

- *Электронное тестирование пройдет 15-го декабря в 14:35, пароли будут высланы старостам групп.*

- *Устные зачеты (экзамены) – 24 и 28 декабря с 10:00, расписание по группам и ссылки будут высланы позже. Необходима включенная камера на стороне студентов и преподавателей.*

Стенд-ап (стендап, stand up) по теме курса “Суперкомпьютеры и параллельная обработка данных”

Заявлены три выступления в формате стенд-ап, 10 минут на каждое выступление:

1. Черкасов Владислав, 416 группа. Как суперкомпьютер справляется с приземлёнными задачами?

2. Раева Анастасия, 412 группа. Суперкомпьютеры и их применение.

3. Гринев Тимофей, 416 группа. Введение в теорию суперлюдей (пародия на теорию суперкомпьютеров)

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

(часть 2)

Вл.В.Воеводин

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

voevodin@parallel.ru

ВМК МГУ, 2020