Граф алгоритма и параллельные вычисления. Внутренний параллелизм программ.

Лекция 3

3.1 Внутренний параллелизм

- Программа содержит параллелизм, если некоторые ее части (операторы) могут быть выполнены параллельно так, что результат параллельного выполнения будет всегда совпадать с результатом последовательного выполнения

3.1.1 Пример: квадратное уравнение $ax^2+bx+c=0$

```
D=SQRT(b^2-4*a*c) // D>O
X1=(-b+D)/(2*a)
X2=(-b-D)/(2*a)
```

D=SQRT(b^2-4*a*c) // D>O	
X1=(-b+D)/(2*a)	X2=(-b-D)/(2*a)

3.1.2 Пример: умножение матрицы на вектор $c=A_{2x2}*b$

```
DO  i = 1, 2
   c(i)=0
   DO j=1,2
      c(i)=c(i)+A(i,j)*b(j)
   End DO
End DO
```

```
ParDO  i = 1, 2
   c(i)=0
   DO j=1,2
      c(i)=c(i)+A(i,j)*b(j)
   End DO
End ParDO
```

c(1)=0 DO j=1,2 c(1)=c(1)+A(1,j)*b(j) End DO	c(2)=0 DO j=1,2 c(2)=c(2)+A(2,j)*b(j) End DO

c(1)=0 ParDO j=1,2 c(1)=c(1)+A(1,j)*b(j) End ParDO	c(2)=0 ParDO j=1,2 c(2)=c(2)+A(2,j)*b(j) End ParDO

c(1)=0		c(2)=0	
c(1)=c(1)+A(1,1)*b(1)	c(1)=c(1)+A(1,2)*b(2)	c(2)=c(2)+A(2,1)*b(1)	c(2)=c(2)+A(2,2)*b(2)

3.1.3 Пример: цикл, который нельзя распараллелить

```
DO  i = 1, n-3
      U(i)=f(x)
      U(i+1)=U(i)
      U(i+2)=U(i+1)
      U(i+3)=U(i+2)
EndDO
```

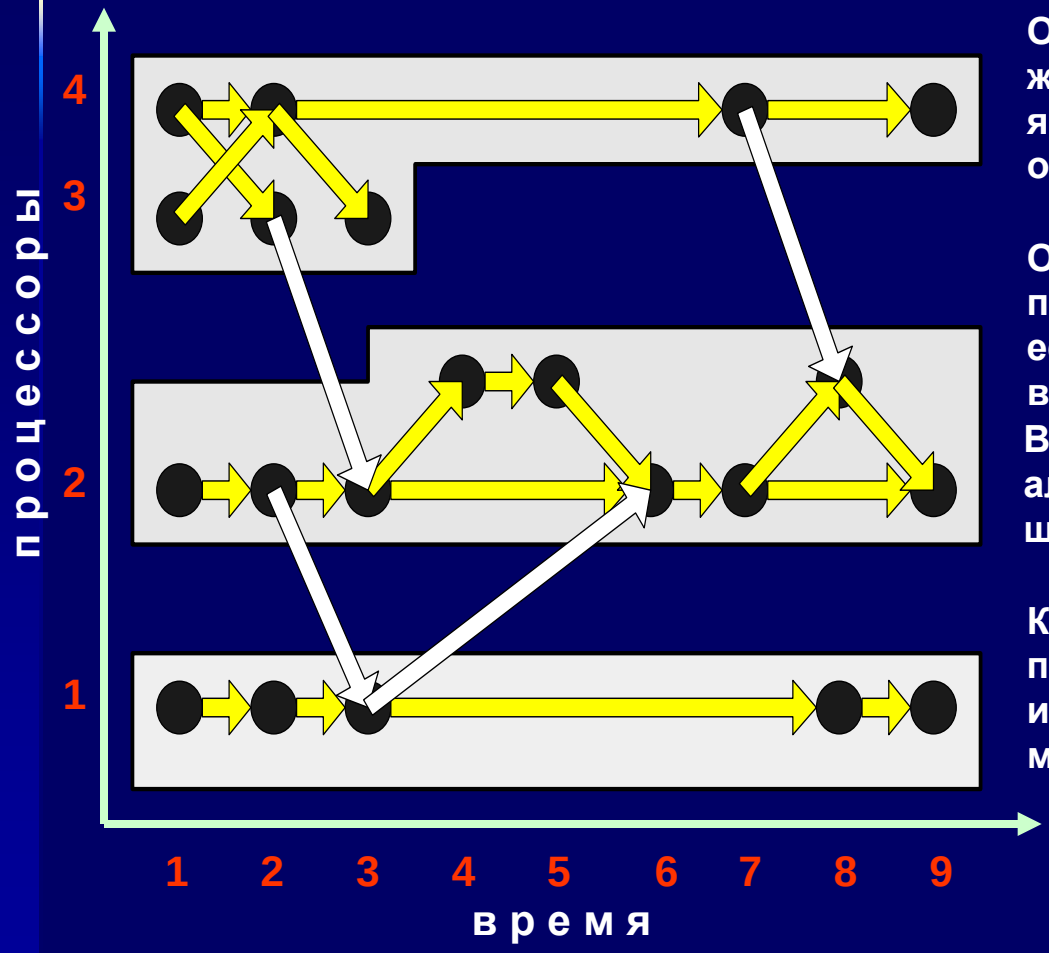
3.1.4 Скрытый параллелизм

```
DO  i = 1, n
      DO j = 1, n
       U( i + j ) = U( 2n – i – j + 1)
      End DO
End DO
```

3.1.4 Выявление скрытого параллелизма: как это сделать?

```
DO  i = 1, n
      ParDO  j = 1, n – i
         U( i + j ) = U( 2n – i – j + 1)
      End ParDO
      ParDO j = n – i + 1, n
         U( i + j ) = U( 2n – i – j + 1)
      End ParDO
End DO
```

3.2 Параллельная форма алгоритма



Операции, выполняемые в один и тот же момент времени, называются ярусом параллельной формы, число операций – шириной яруса.

Общее количеств ярусов параллельной формы называется ее
высотой.
Высота определяет время реализации алгоритма, среднее значение ширины ярусов – ускорение.

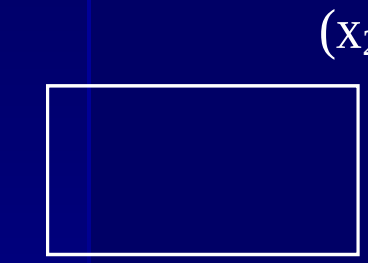
Каждый алгоритм имеет много параллельных форм. Наиболее интересны параллельные формы минимальной высоты.

3.2 Параллельная форма алгоритма

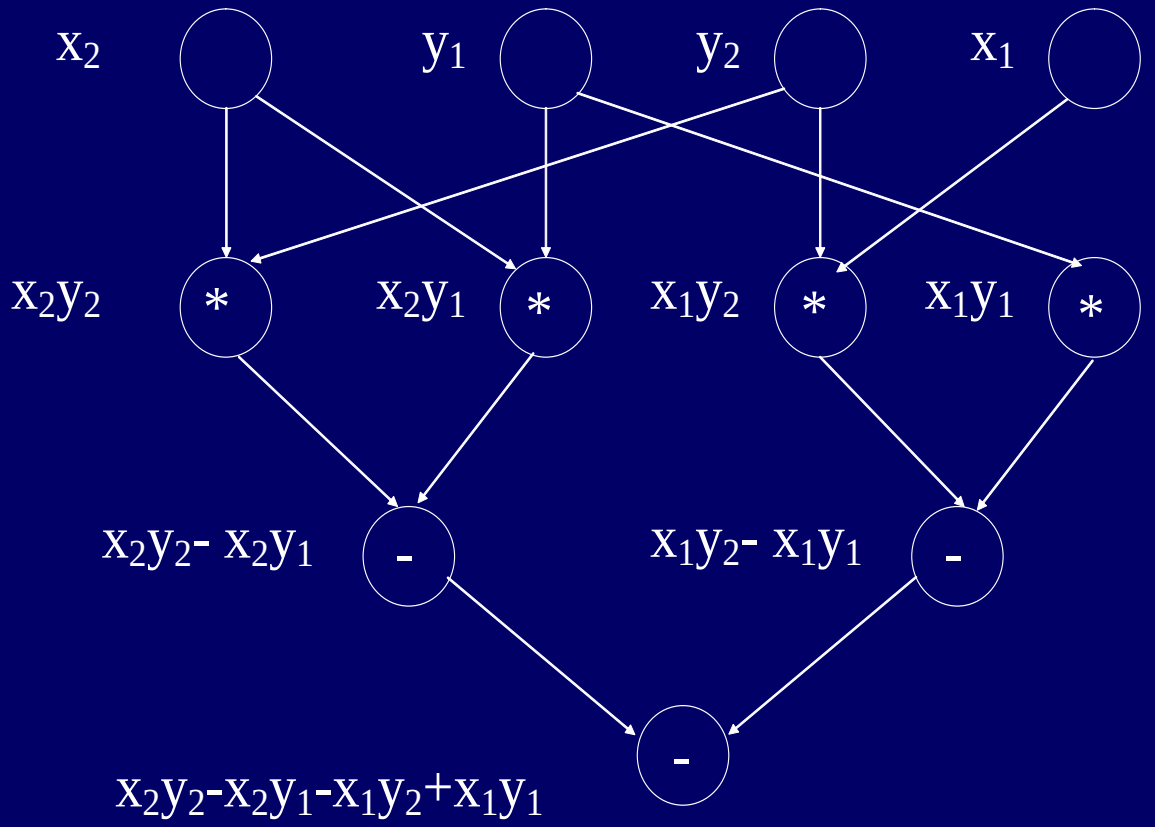
- **Параллельная форма алгоритма может быть эквивалентно задана и исследована с помощью ориентированного графа**

3.2 Пример

Граф алгоритма вычисления площади прямоугольника, заданного координатами двух противолежащих углов



$$S = (x_2-x_1)(y_2-y_1) = x_2y_2-x_2y_1-x_1y_2+x_1y_1$$

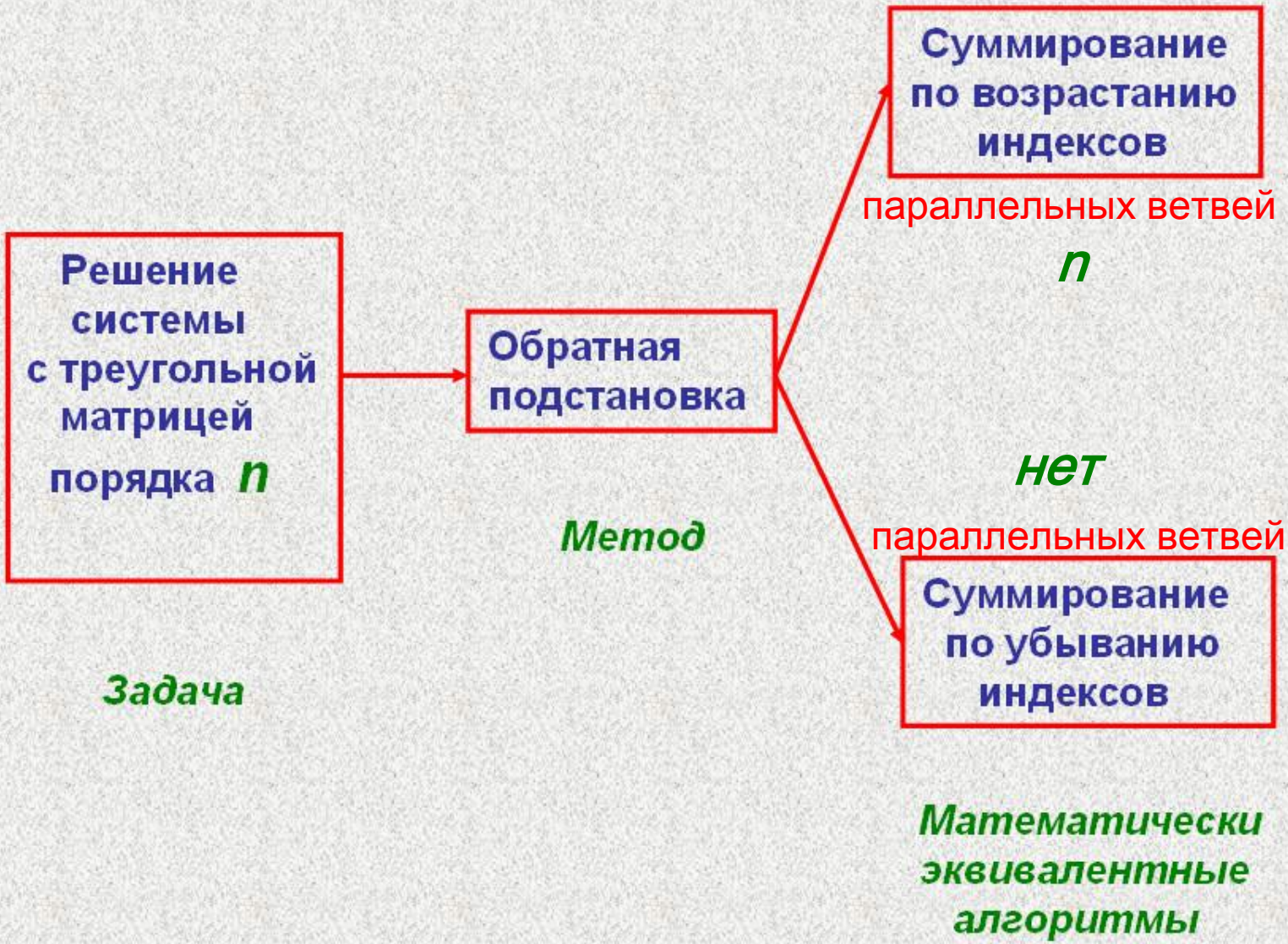


3.3 Эквивалентные преобразования

1. Изменение вида формульных выражений на основе законов ассоциативности, коммутативности, дистрибутивности, а также за счет приведения и создания подобных членов.
2. Выбор порядка выполнения операций или, другими словами, расстановка скобок в формульных выражениях в соответствии с указанными законами.

При одних и тех же входных данных
все эквивалентные преобразования
приводят к одним и тем же результатам,
ЕСЛИ ВСЕ ОПЕРАЦИИ ВЫПОЛНЯЮТСЯ ТОЧНО.


Задача
Решение системы линейных алгебраических уравнений с квадратной матрицей порядка n
Математически эквивалентные преобразования формульных выражений
Метод Гаусса
число операций
n^3
n^log27
число операций
Метод Штрассена
e^n
число операций
Формулы Крамера
Разные вычислительные свойства



Разные вычислительные свойства

3.4 Задача – метод – алгоритм – программа

ЗАДАЧА. Описывается модель изучаемого явления в форме некоторой совокупности математических соотношений.

МЕТОД. Заданы в общем виде множество выполняемых операций и схема связей между ними.

АЛГОРИТМ. Понятие введено для последовательных вычислений. Определены множество операций и порядок их выполнения. Изменение порядка может изменить вычислительные свойства алгоритма. В параллельных вычислениях строгое понятие алгоритма не введено.

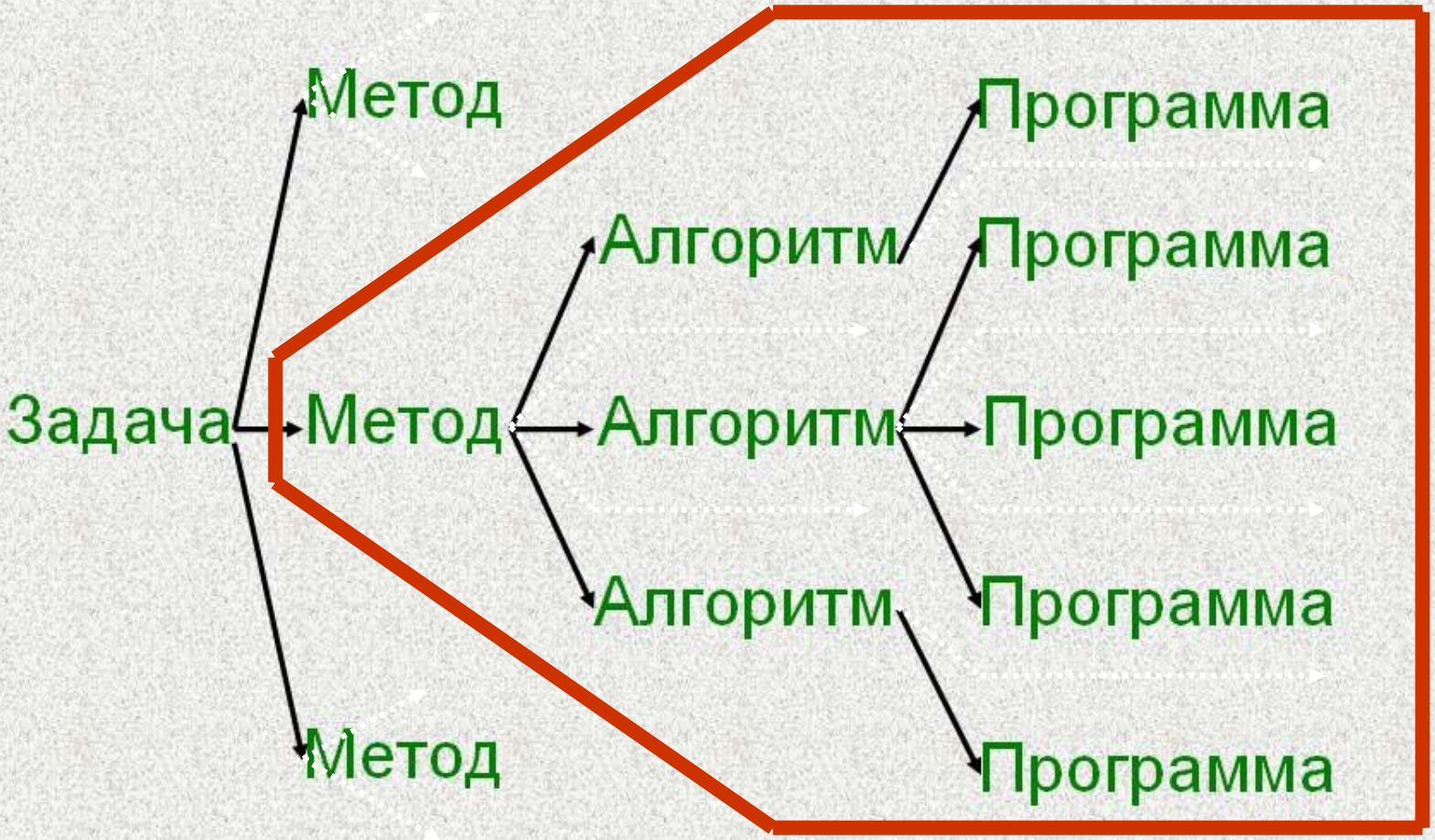
ПРОГРАММА. За основу написания программы всегда берется текст некоторого алгоритма. Однако в интересах эффективности реализации порядок выполнения операций и даже сами операции чаще всего меняются. Тем самым реализуется алгоритм, отличный от исходного и имеющий в общем случае другие вычислительные свойства.

3.4 Задача – метод – алгоритм – программа

Каждый численный метод порождает множество математически эквивалентных алгоритмов и программ за счет математически эквивалентных формульных преобразований. Все они дают одни и те же результаты при одних тех же входных данных, если операции выполняются точно. Однако на данном множестве имеется огромный разброс вычислительных свойств, например, таких как:

- **общее число выполняемых операций;**
- **влияние ошибок округления;**
- **число независимых ветвей вычислений;**
- **размер требуемой памяти;**
- **сложность коммуникационных связей;**
- **и многих других.**

Наличие одних хороших свойств не гарантирует, что хорошими будут какие-то другие.



А что же нужно исследовать и распараллеливать?

Математически эквивалентные алгоритмы

Математически эквивалентные программы

Разные вычислительные свойства

3.5 ГРАФ АЛГОРИТМА

Существуют ли на множестве математически эквивалентных алгоритмов и программ представительные подмножества, на которых гарантированно сохраняются какие-либо вычислительные свойства, например, влияние ошибок округления?

Да, существуют !

3.5 ГРАФ АЛГОРИТМА

- **Чтобы найти инварианты алгоритмов, не зависящие от форм записи, необходимо максимально освободиться от всех связей, не влияющих на конечный результат. Такие связи появляются из-за конкретных особенностей используемых языков описания алгоритмов. В первую очередь, эти особенности таковы:**

1. **Порядок выполнения операций; точнее, излишняя свобода или ограничения в его выборе;**
2. **Использование памяти; точнее пересчет содержимого ячеек памяти в программах;**
3. **Все особенности в языках программирования, связанные с оформлением описания программ.**

3.5 ГРАФ АЛГОРИТМА

- **Вершины графа символизируют исполняемые операции алгоритма. Из вершины *A* дуга идет в вершину *B* в том и только в том случае, когда операция, соответствующая вершине *A* порождает результат, используемый в качестве аргумента операцией, соответствующей вершине *B*.**
- **Построенный граф представляет информационное ядро алгоритма и называется «Граф алгоритма»**

3.6 Теорема об ошибках округления

Пусть фиксирован способ округления. Предположим, что алгоритмы выполняют одни и те же множества операций. Для того чтобы при одинаковых входных данных влияние ошибок округления в алгоритмах было одним и тем же, необходимо и достаточно, чтобы графы алгоритмов были изоморфны.

3.7 Графы зависимостей

- Граф алгоритма играет исключительно важную роль независимо от того, какая используется форма описания. Однако при анализе программ приходится принимать во внимание не только содержательные операции алгоритма, но и операции с памятью. В этом случае по такому же принципу строятся более общие графы, называемые *графы зависимостей*.
- Во всех графах зависимостей две вершины-операции связываются дугой тогда и только тогда, когда обе операции используют содержимое одной и той же ячейки памяти.
- В любом таком графе будет много «лишних» дуг, если содержимое одних и тех же ячеек памяти используется или пересчитывается многократно.

3.8 Минимальныме графы зависимостей

- **Число дуг в графах зависимостей очень избыточно. При их уменьшении исследование графов упрощается. Но при значительном уменьшении числа дуг могут исчезнуть какие-то важные структурные свойства.**

- **Каковы оптимальные графы зависимостей? Такие графы найдены и названы *минимальными графами зависимостей*. Сохраняя все важнейшие свойства графов зависимостей, они имеют минимальное число дуг (В. В. Воеводин, 1986) .**

3.9 Линейный класс программ

Будем считать, что программа принадлежит к линейному классу, если она состоит только из операторов присваивания и операторов цикла, а пересчет переменных и описание границ циклов осуществляется с помощью линейных индексных выражений. Все коэффициенты при индексах являются целыми постоянными числами. Неопределенные внешние переменные могут входить в индексные выражения аддитивно.

Линейный класс играет такую же роль в изучении структуры программ как линейные функции в математическом анализе, линейные неравенства в задачах оптимизации, линейная алгебра в вычислительной математике и т. п.

Предложены различные расширения линейного класса, в том числе на программы, имеющие обращение к функциям и подпрограммам, а также ветвления, произвольную организацию циклов и др.

Фундаментальная теорема

Для любой программы из линейного класса любой минимальный граф зависимостей представляется конечной системой линейных функций, заданных на линейных многогранниках. Число функций не зависит от значений внешних переменных (В. В. Воеводин, 1995).

Основным инструментом исследования структуры минимальных графов зависимостей являются развертки (В. В. Воеводин, 1987).

Рассмотрим вещественный функционал $f(x)$, определенный на вершинах графа. Пусть дуга идет из вершины u в вершину v. Будем говорить, что функционал $f(x)$ возрастает (не убывает) вдоль дуги графа, если $f(v)>f(u)$ ($f(v)\geq f(u)$). Функционал $f(x)$ называется строгой (обобщенной) разверткой, если он строго возрастает (не убывает) вдоль всех дуг графа.

Пусть алгоритм реализуется на какой-то реальной или гипотетической, параллельной или последовательной вычислительной машине. Всегда значение функционала $f(x)$ можно трактовать как момент времени, в который на данной машине выполняется операция, соответствующая вершине x

Три простых утверждения

или как установить независимость операций

Пусть на вершинах графа зависимостей задана строгая развертка $f(x)$. Никакая пара вершин, лежащих на поверхности уровня $f(x)=\text{const}$, не может быть связана путем графа.

Пусть на вершинах графа зависимостей заданы две обобщенные развертки $f_1(x)$ и $f_2(x)$. Построим развертку $f(x) = \alpha f_1(x) + \beta f_2(x)$, где $\alpha,\beta>0$. Никакая пара вершин, лежащих на поверхности уровня $f(x)=\text{const}$ и не лежащих на поверхности уровня $f_1(x)$ или $f_2(x)$, не может быть связана путем графа.

Пусть на поверхности уровня какой-нибудь развертки находятся какие-то вершины. Если никакая пара из этих вершин не связана путем графа, то операции, соответствующие этим вершинам, можно выполнять параллельно.

Теорема

Для любой программы из линейного класса и любого минимального графа зависимостей существуют нетривиальные развертки, представляемые конечной системой линейных функционалов, заданных на линейных многогранниках. Число линейных функционалов не зависит от значений внешних переменных (В. В. Воеводин, 1991).

Построение графов для большого числа конкретных алгоритмов выявило удивительную закономерность: большое разнообразие алгоритмов не приводит к такому же разнообразию информационных структур. Многие графы формально различных алгоритмов оказались изоморфными в главном, отличаясь друг от друга содержанием вершин и дуг. Например,

На всем множестве алгоритмов линейной алгебры различных по существу типов графов оказалось всего лишь порядка десятка.

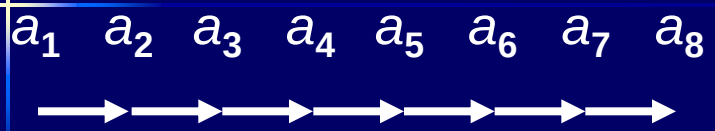
Графы различаются, главным образом, математическим содержанием вершин-операций.

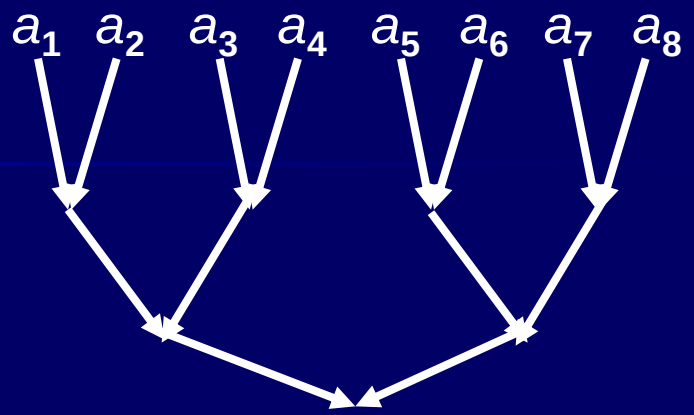
Гипотеза:

Типовых информационных структур алгоритмов в конкретных областях немного.

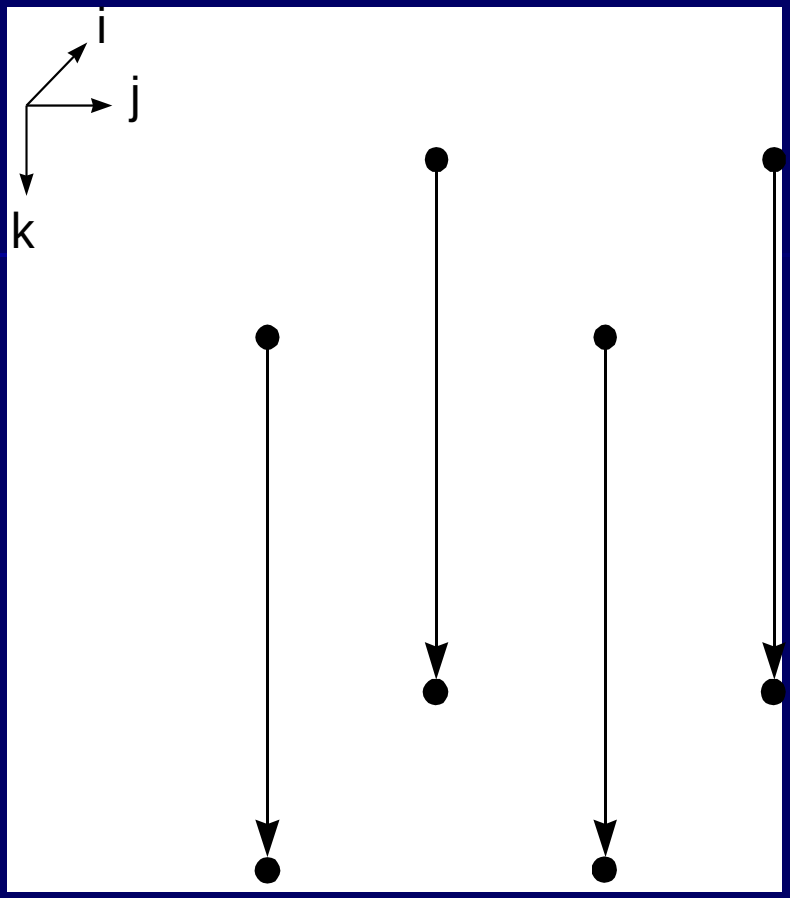
Практика подтверждает гипотезу !

ПРИМЕРЫ ТИПОВЫХ СТРУКТУР

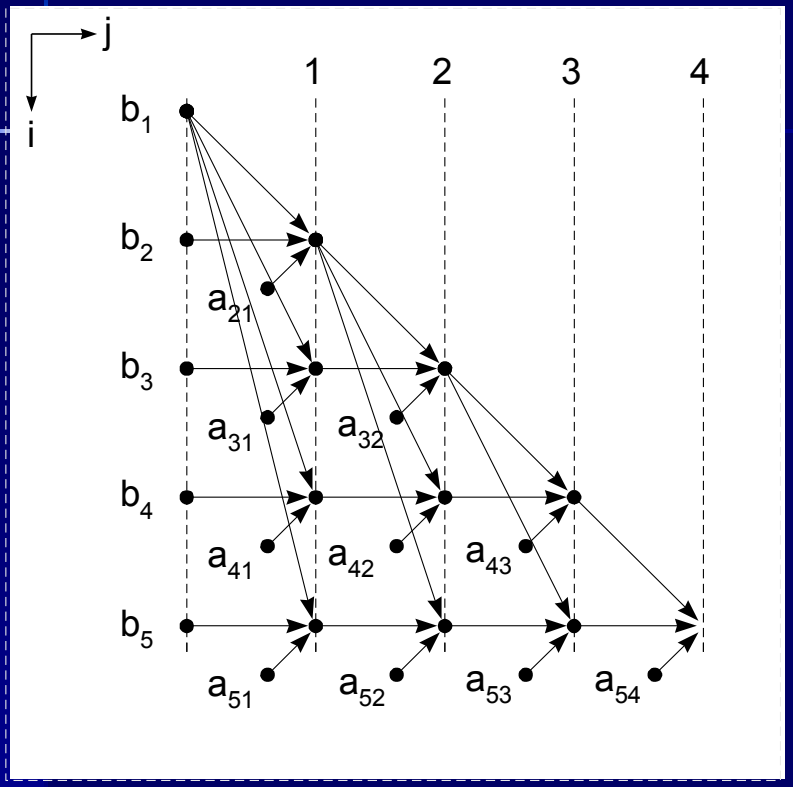


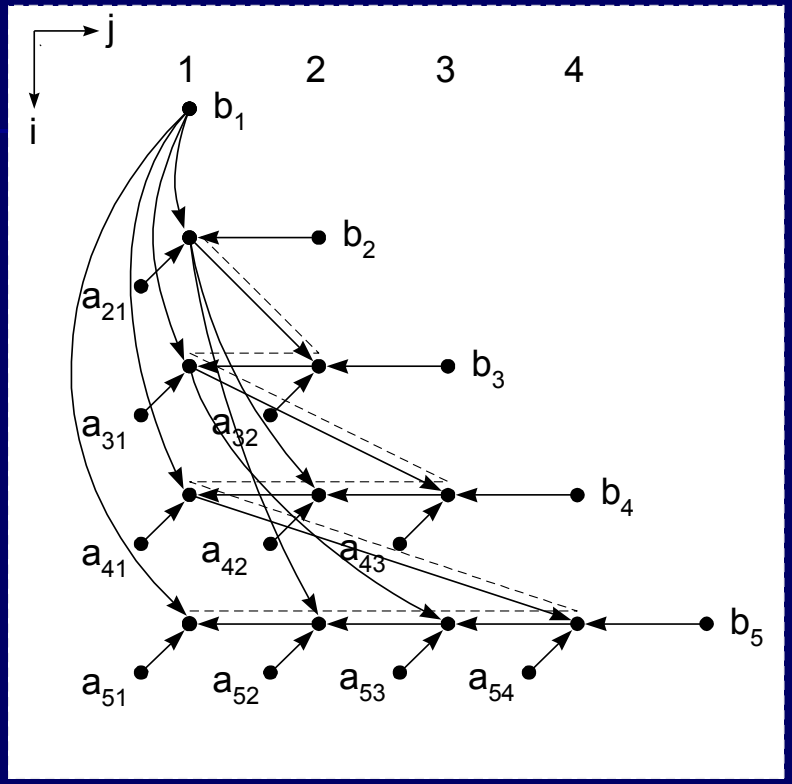


Последовательное и попарное суммирование

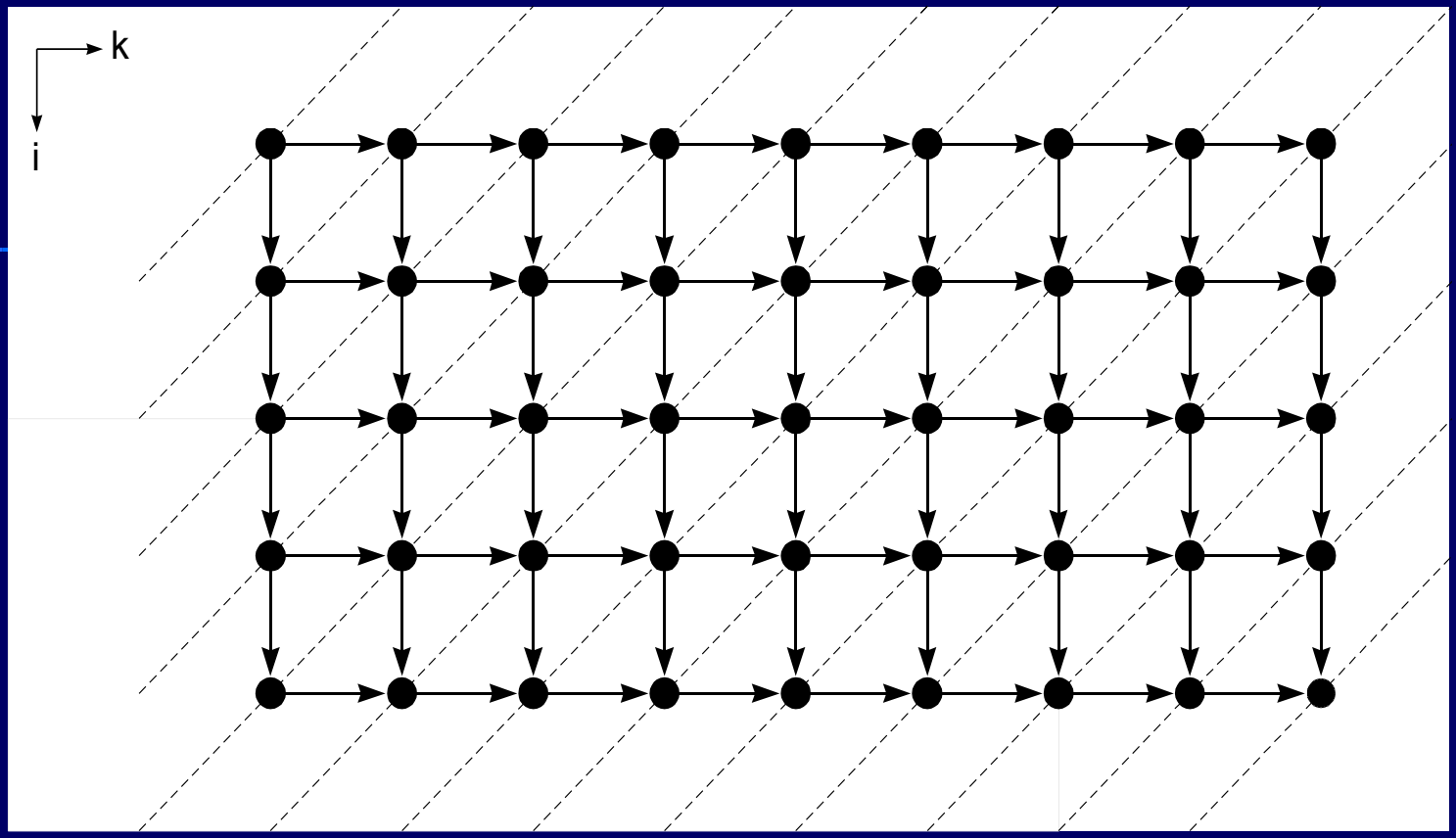


Перемножение матриц, *n*=2

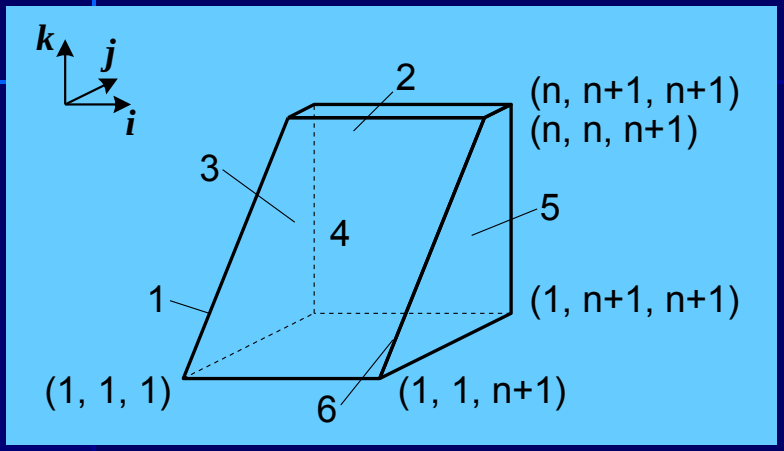


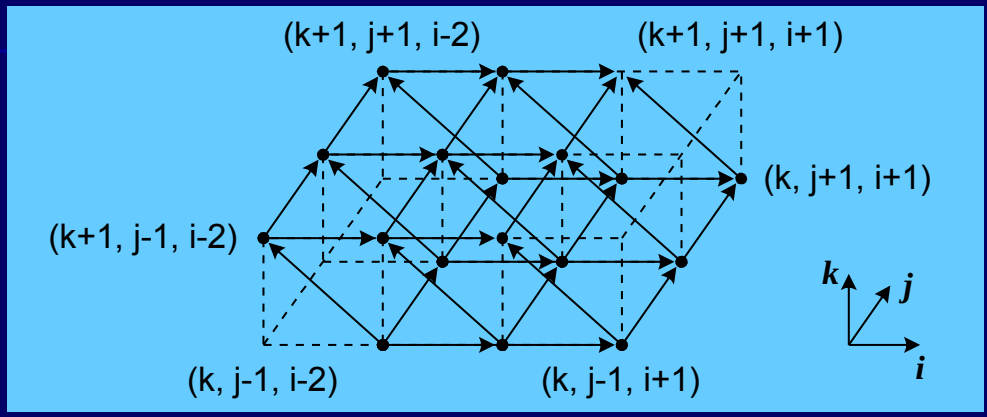


Решение треугольных систем

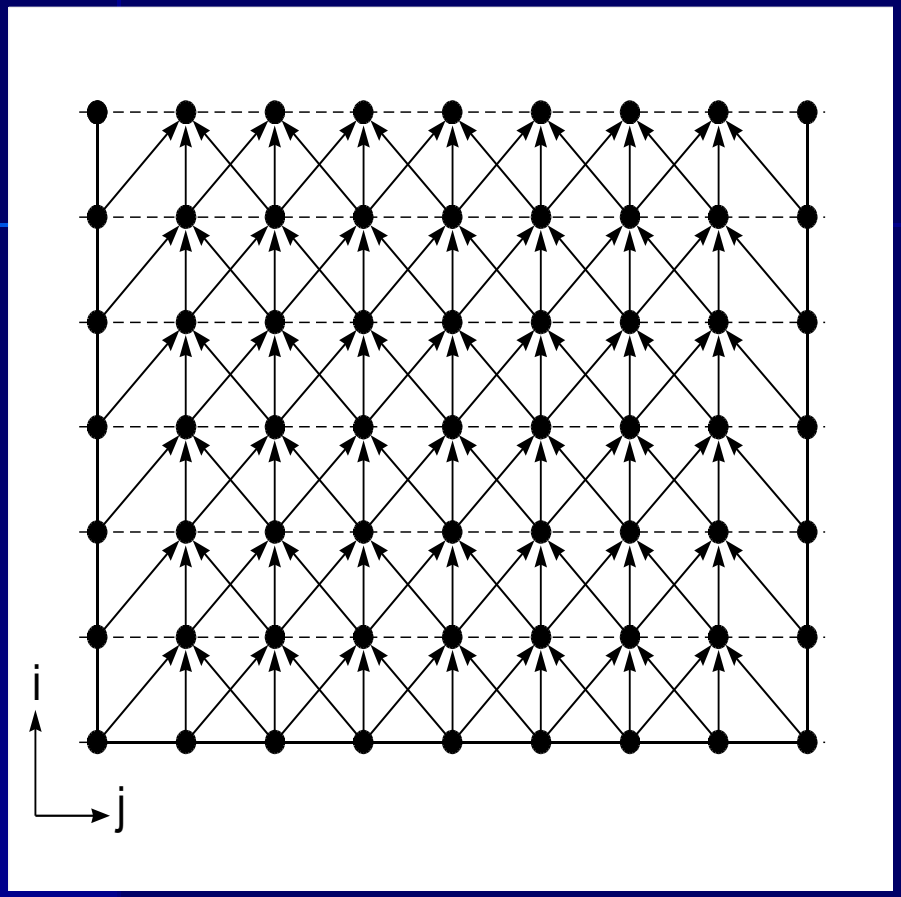


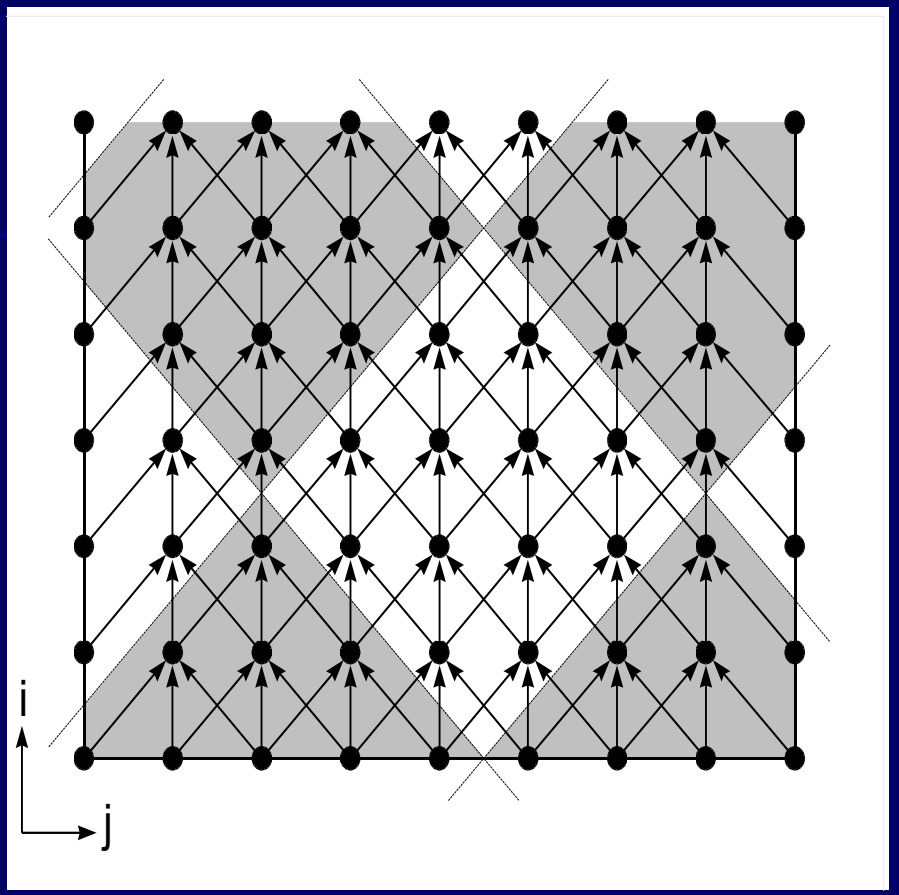
Решение блочно двухдиагональных систем





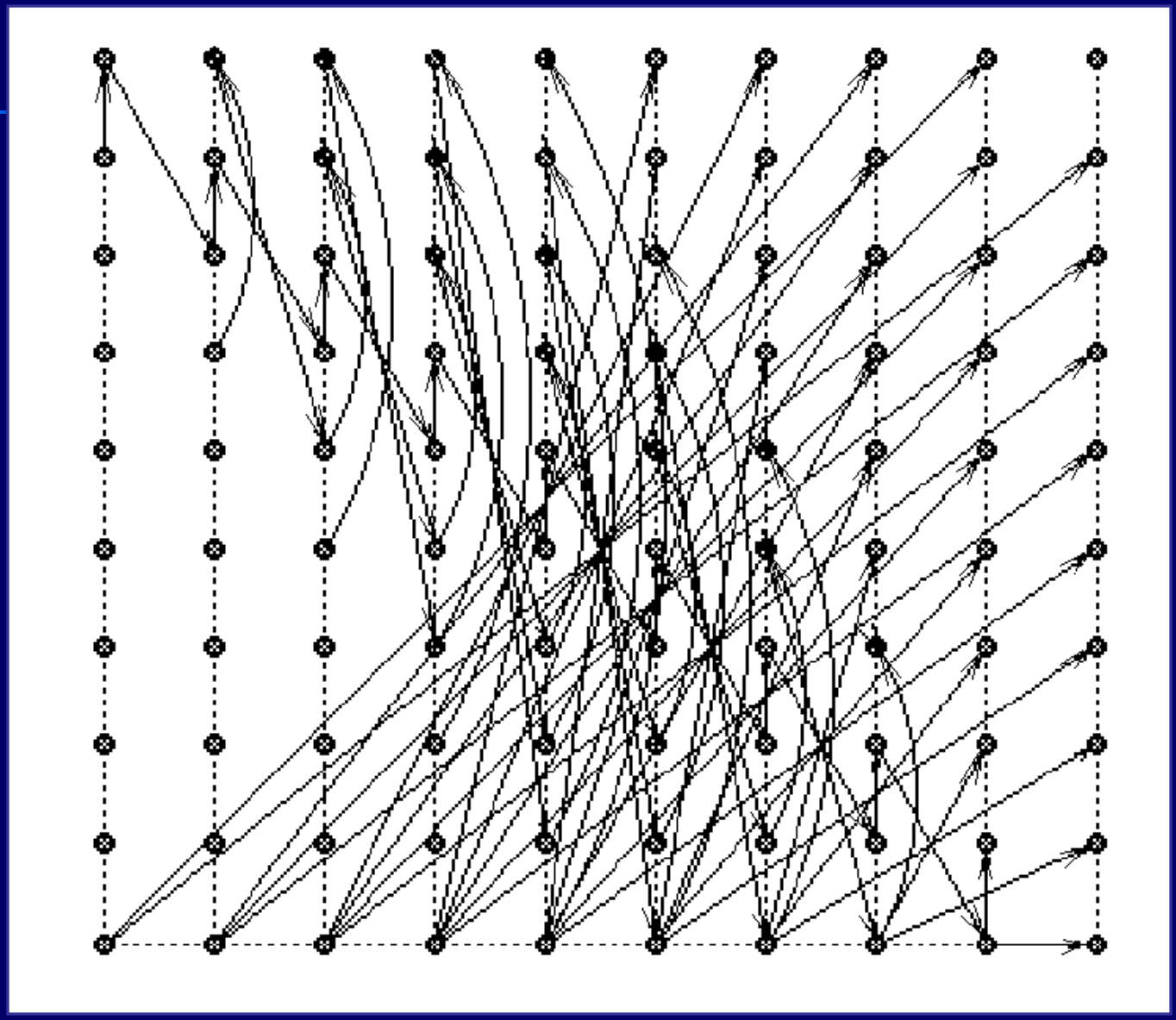
Решение системы методом Жордана





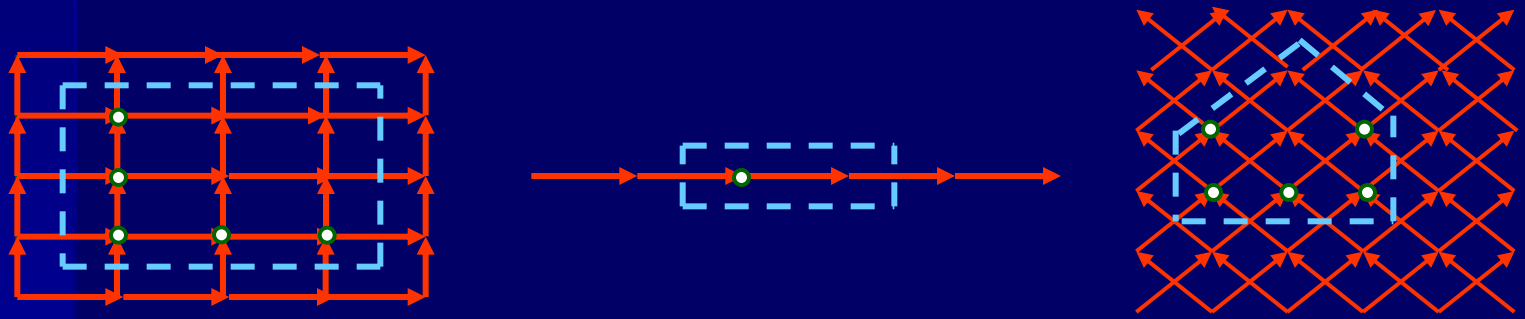
Явная схема и макропараллелизм

Граф зависимостей “простого” примера



ГРАФЫ АЛГОРИТМОВ – ЭТО ОЧЕНЬ ПРОСТО

Пусть граф алгоритма размещен в некотором пространстве *X* и его подграф принадлежит области *D*. Будем называть соответствующий данному подграфу фрагмент алгоритма *локально вычисляемым*, если при задании начальных условий на вершинах вблизи границы области *D* весь фрагмент может быть реализован без выхода из области *D*. В зависимости от вида области *D* фрагмент может быть отрезком, прямоугольником, конусом и т.п.



Можно ли расщепить алгоритм на локально вычисляемые фрагменты и реализовать совокупность фрагментов параллельно?

РЕКУРРЕНТНЫЕ АЛГОРИТМЫ

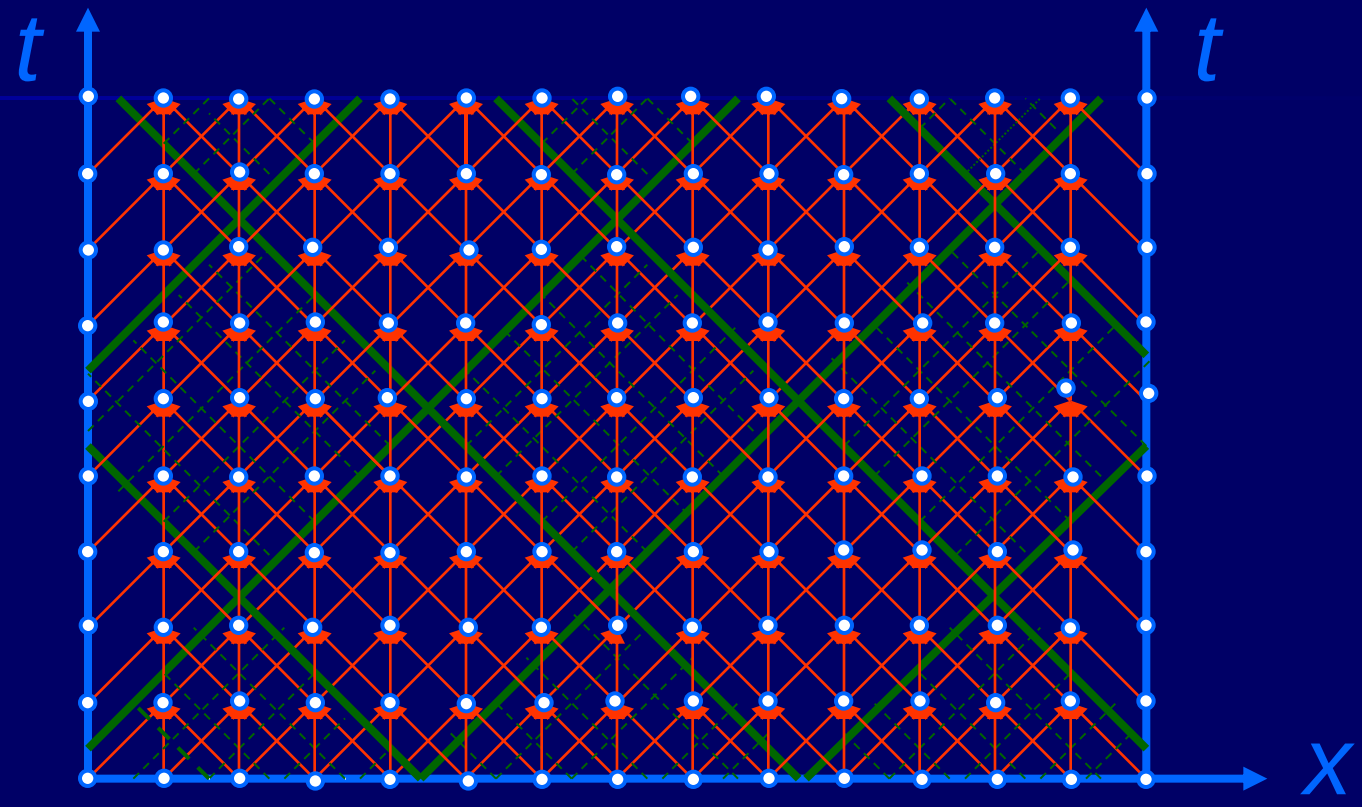
$$U_f = F_f\,(U_{f-f_1}, \ldots, U_{f-f_r}),\quad f \text{ из } D$$

Пример: явная схема уравнения теплопроводности

$$u_j^i = u_j^{i-1} + \frac{\tau}{h^2}\left(u_{j-1}^{i-1} - 2u_j^{i-1} + u_{j+1}^{i-1}\right)$$

Все рекуррентные алгоритмы расщепляются на локально вычисляемые фрагменты и совокупность фрагментов реализуется параллельно.

РЕГУЛЯРНЫЕ ГРАФЫ

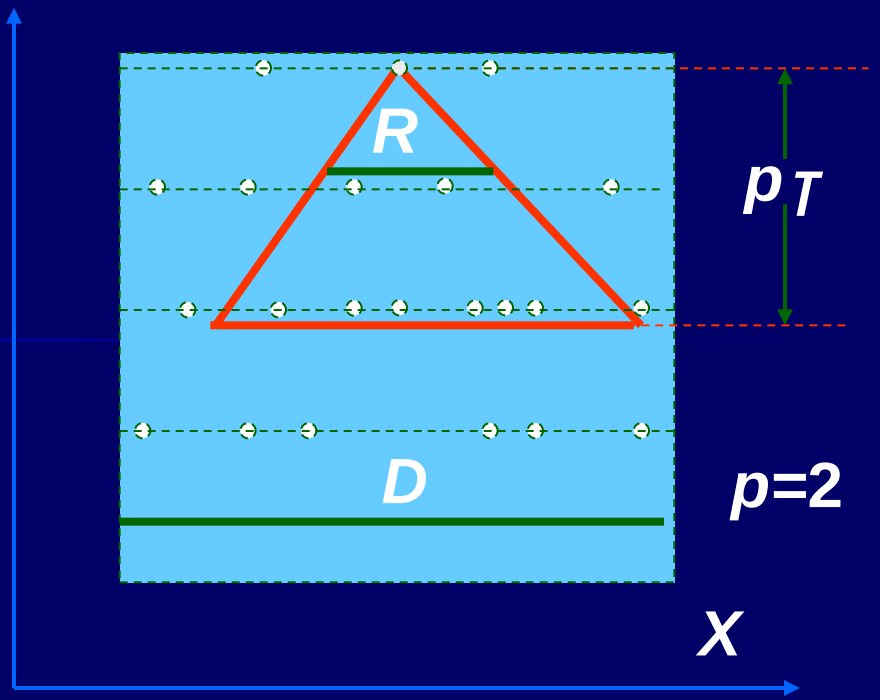


Граф алгоритма явной схемы уравнения теплопроводности

Обмен информацией через пограничные полосы

Сетка обязана быть регулярной

ОПОРНЫЙ КОНУС



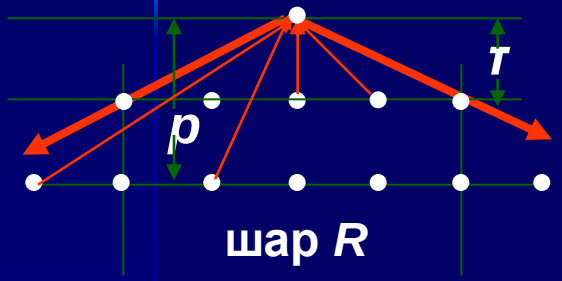
Пусть в пространстве X задана область D и шар R малого размера по сравнению с D. Рассмотрим пространство $W=X \oplus t$. Выберем число T и построим в пространстве W конус K с вершиной в точке (0,0), сечением которого в гиперплоскости $t = -T$ является шар R с центром в точке $(0,-T)$. Множество точек конуса с условием $-pT \le t \le 0$ назовем опорным конусом высоты p.

ЛОКАЛЬНЫЕ АЛГОРИТМЫ

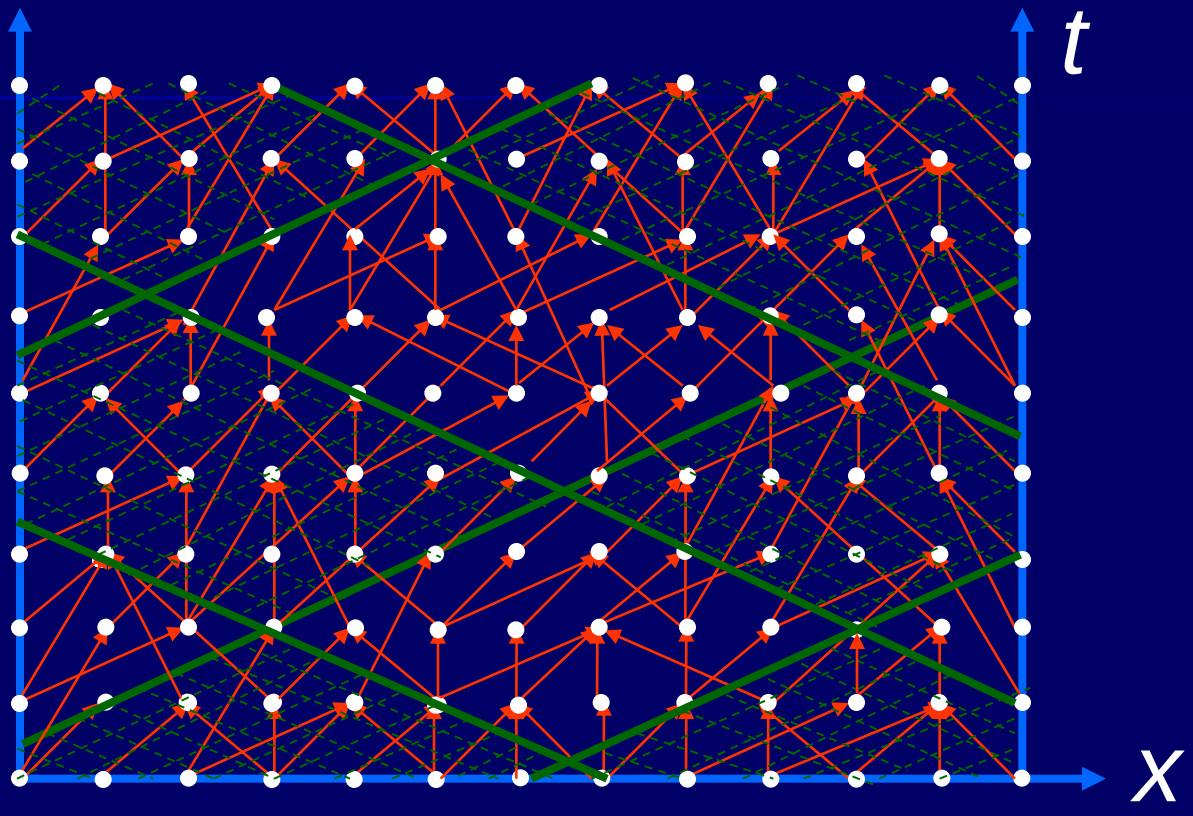
Пусть в области D пространства X любым способом строится последовательность сеток $\sum_t$. Допустим, что в D задан шар R, а в пространстве W опорный конус K высоты p. Предположим, что точка x_t^i сетки $\sum_t$ является вершиной опорного конуса и значение функции F_t на ней вычисляется как какая-то функция точек, попавших в конус K из других сеток. Любой алгоритм вычисления функций F_t таким способом будем называть **локальным.**

Все локальные алгоритмы расщепляется на локально вычисляемые фрагменты и совокупность фрагментов реализуется параллельно.

ЛОКАЛЬНЫЕ ГРАФЫ



опорный конус



Расщепление на локально вычисляемые фрагменты

Обмен информацией через пограничные полосы

Сетка не обязана быть регулярной

Как показывает опыт, большинство реальных алгоритмов или являются локальными, или сводятся к локальным, или расщепляются на фрагменты, значительная часть которых оказывается локальными. В частности, локальными являются все рекуррентные алгоритмы.

НЕ ИЗВЕСТНЫ ДРУГИЕ,
МАССОВО ИСПОЛЬЗУЕМЫЕ СТРУКТУРЫ,
КРОМЕ ЛОКАЛЬНЫХ,
КОТОРЫЕ ЛЕГКО МАСШТАБИРУЮТСЯ
ПОД ТРЕБОВАНИЯ
МНОГОПРОЦЕССОРНЫХ СИСТЕМ
С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ.