Прямые методы для параллельных компьютеров

Рассмотрим систему линейных уравнений

$$Ax = b$$

с невырожденной матрицей A размера $n \times n$.

Будем полагать матрицу A заполненной и рассмотрим компьютеры **параллельной архитектуры**

LU – разложение

Рассмотрим параллельную систему с локальной памятью
Пусть $p = n$, i-я строка матрицы A находится в i-м процессоре
Тогда один из возможных вариантов LU разложения

Шаг 1: $\mathbf{a_1} \rightarrow P_i$, $i = 2, \ldots, n$

В P_i $(i=2,\ldots,n)$ вычисляются l_{i1} и новые элементы a_{ij} $(j=2,\ldots,n)$

Шаг 2: $\mathbf{a_2} \rightarrow Pi$, $i = 3, \ldots, n$

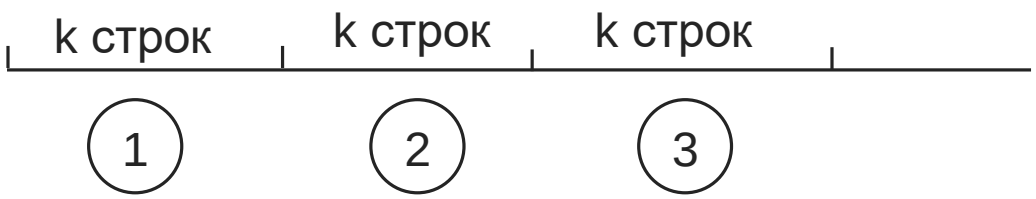
В P_i $(i=3,\ldots,n)$ вычисляются l_{i2} и новые элементы a_{ij} $(j=3,\ldots,n)$

Недостатки: 1) Значительный объем обмена данных
2) Уменьшение на один активный процессор на каждом шагу

Альтернатива – хранение по столбцам
i-й столбец $\rightarrow$ в i-й процессор
Тогда
l_{i1} вычисляется в P1 $\rightarrow$ рассылка $\rightarrow$ параллельная модификация

Слоистая схема хранения

Возникающие сложности – это проблема балансировки нагрузки
Реально $p \le n$. В этом случае проблема балансировки смягчается
Пусть $n = k\,p$ и храним данные по строкам

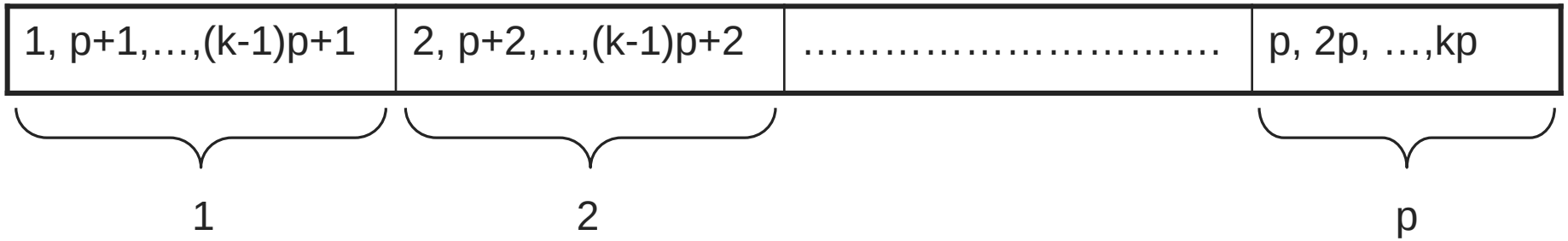


блочная схема хранения

f(k) = общее время вычислений / время обменов и простоев

Другая схема $n = k\,p$

циклическая слоистая схема



Принцип слоистого хранения оправдан главным образом для систем с локальной памятью. Но его можно применять и в системах с глобальной памятью как средство распределения заданий между процессорами.
Однако в системах с глобальной памятью динамическая балансировка может осуществляться и с помощью банка заданий.

Обсудим LU разложение с помощью слоистой схемы

Обычный способ
Вычисление l_{i1} → модификация соответствующих строк → начало второго шага (рассылка второй строки остальным процессорам)

ЗАДЕРЖКА

Выход – начать рассылку второй строки, как только она сосчитана

Это стратегия опережающей рассылки

Если можно совместить рассылку и обмен, то пока будет идти рассылка, процессоры будут заниматься модификацией.
Насколько выгодна такая стратегия для конкретной машины будет зависеть от топологии межпроцессорных связей.

Стратегию опережающей рассылки можно использовать и в системах с разделяемой памятью.
Здесь при прямолинейной реализации потребовалось бы *синхронизация* после каждого шага (не начать следующий шаг, пока не кончен предыдущий).
Стратегия опережающей рассылки теперь носит характер
стратегии опережающих вычислений

Она меняет порядок действий, маркируя *k+1* строку, как только ее модифицируют, - «ГОТОВО»
Теперь другие процессора, завершив свою работу, на ***k*-м** шаге, могут немедленно приступить к ***k+1*** шагу, если *k+1* строка «готова»
Это – **конвейеризация**

Проблемы частичного выбора главного элемента

Дополнительные аспекты проблемы хранения информации

1. *A* – циклическая столбцовая схема → поиск ГЭ в одном процессоре

Проста, но чревата опасностью простоя остальных процессоров во время поиска.
Эту опасность можно уменьшить, применяя
Стратегию Опережающих Вычислений и Рассылки

Ведущая строка → передать другим → параллельная работа по перестановке
определена (явно или неявно)

2. *A* – циклическая строчная схема → поиск ГЭ во всех процессорах →

→ответ в одном процессоре → разослать остальным → перестановка
(2 процессора)

Надо принять решение, выполнять ли перестановку физически или переиндексировать – либо простой процессоров при работе двух, либо искажение циклической схемы хранения

ijk-формы и параллельно-векторные компьютеры

Приводимые обозначения

- *kij, r* – форма *kij* с использованием циклической слоистой строчной схемы
- *kij, c* - форма *kij* с использованием циклической слоистой столбцовой схемы
- ПВД – полные векторные длины, т.е. можно работать с векторами максимальной длины, допустимыми на данном шаге *LU*-разложения
- ЧВД – частичные векторные длины – соответственно данные распределены между процессорами и, следовательно, длины векторов / число процессоров

kij, r:	Минимальные задержки	ПВД
kij, c:	Задержки на вычисление множителей и рассылку	ЧВД
kji, r:	Минимальные задержки	ЧВД
kji, c:	Минимальные задержки	ПВД при наличии задержек
ikj, r:	Сильный дисбаланс нагрузки	ЧВД
ikj, c:	Задержки на вычисление множителей и рассылку	ЧВД
jki, r:	Большие расходы на обмены	ЧВД
jki, c:	Задержки на вычисление множителей и рассылку	ПВД при наличии задержек
ijk, r:	Большие расходы на обмены	
ijk, c:	Большие расходы на обмены	
jik, r:	Большие расходы на обмены	
jik, c:	Большие расходы на обмены	

Пример мелкозернистого алгоритма

В принципе LU-разложение можно организовать следующим образом (здесь каждый шаг соответствует параллельным вычислениям)

Шаг.1 Вычислить первый столбец множителей l_{i1}, $i = 2, \ldots, n$.

Шаг.2 Вычислить модифицированные элементы

$a_{ij}^{1} = a_{ij} - l_{i1}a_{1j}$, $i = 2, \ldots, n$, $j = 2, \ldots, n$.

Шаг 3. Вычислить второй столбец множителей l_{i2}, $i = 3, \ldots, n$.

Шаг 4. Вычислить модифицированные элементы

$a_{ij}^{2} = a_{ij}^{1} - l_{i2}a_{2j}^{1}$, $i = 3, \ldots, n$, $j = 3, \ldots, n$.

……

Шаг 2n-3. Вычислить последний множитель $l_{n,n-1}$.

Шаг 2n-2. Модифицировать элемент (n, n)

Поскольку для модификации любого элемента требуется две операции (**+** и **x**), то процесс завершится за **3(n - 1)** временных шагов.

В этой схеме предполагается, что в системе имеется по крайней мере $(n - 1)^2$ процессоров, которые необходимы на шаге 2.

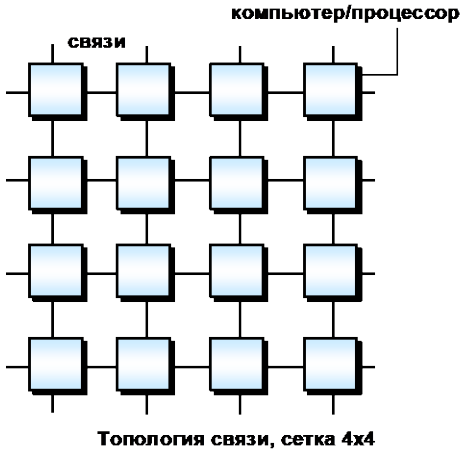
Пример мелкозернистого алгоритма (продолжение)

В приведенном примере обойден важный вопрос обменов
Рассмотрим одну из схем, в которой пересылка данных между любыми двумя процессорами происходит за один шаг.
Пусть процессора пронумерованы, как элементы матрицы, т.е. процессор P_{ij} производит пересчет элемента a_{ij}
Тогда схема предыдущего слайда модифицируется следующим образом:

Шаг 1. Вычислить l_{i1} в процессоре P_{i1}, $i = 2, \ldots, n$.
Шаг 1а. Переслать l_{i1} в процессоры P_{ij}, $i = 2, \ldots, n$, $j = 2, \ldots, n$
Шаг 2. Вычислить a_{ij}^1 в процессоре P_{ij}, $i = 2, \ldots, n$, $j = 2, \ldots, n$.
Шаг 2а. Переслать a_{2j}^1 в P_{ij}, $i = 3, \ldots, n$, $j = 2, \ldots, n$.
……

Если считать, что на каждом шаге все необходимые пересылки могут быть сделаны за единицу времени, то общее количество временных шагов увеличивается до **O(5n)**

Алгоритм потока данных



Пусть n^2 процессоров расположены в виде квадратной решетки и каждый соединен с четырьмя соседями.

Предполагаем:

(1) время пересылки = времени вычислений

(2) пересылка и вычисления могут проходить одновременно

Возможна следующая схема:

Шаг 1. Переслать a_{1j} из P_{1j} в P_{2j}, $j = 1, \ldots, n$.

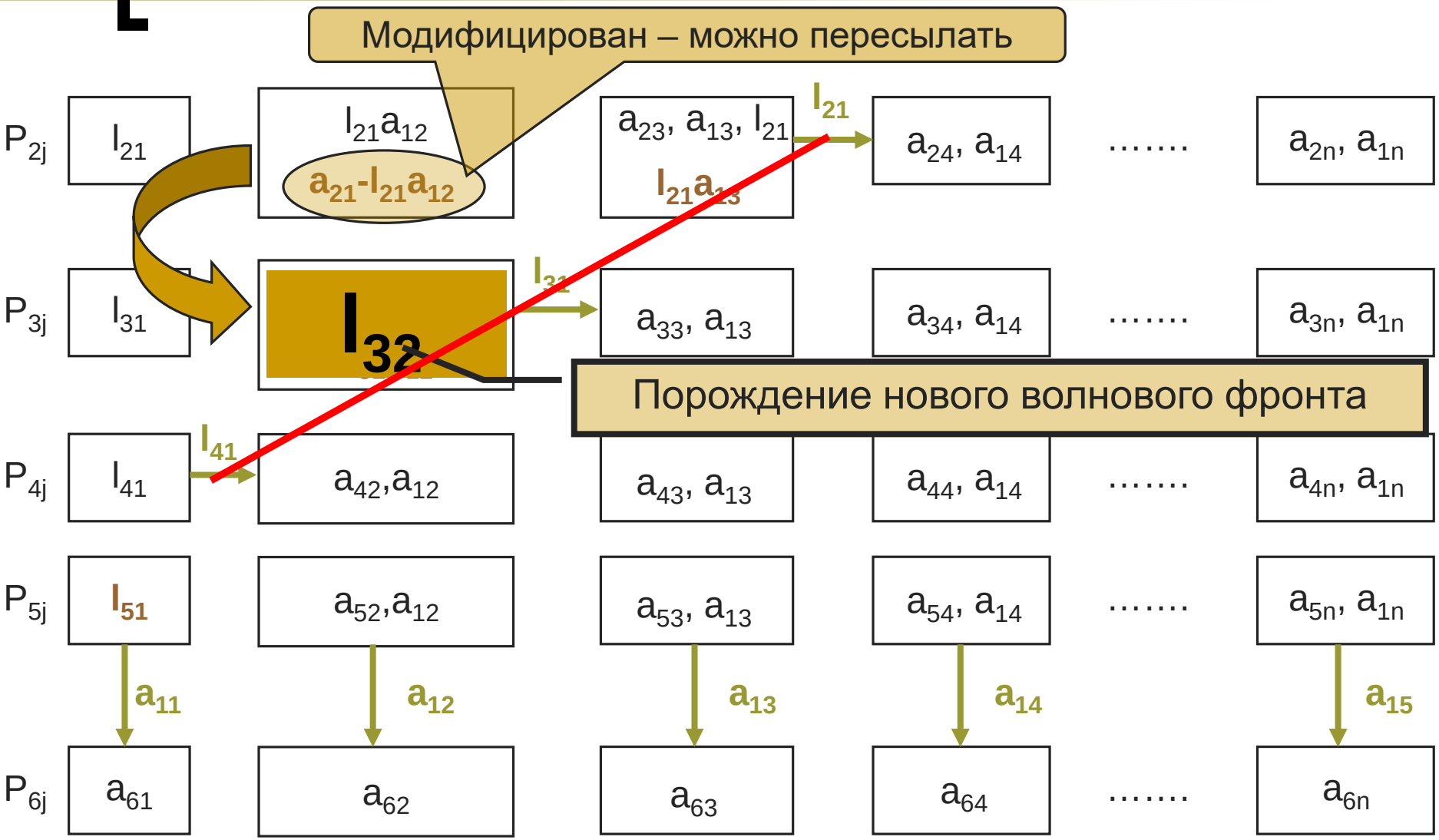
Шаг 2. Переслать a_{1j} из P_{2j} в P_{3j}, $j = 1, \ldots, n$. Вычислить l_{21}.

Шаг 3. Переслать a_{1j} из P_{3j} в P_{4j}, $j = 1, \ldots, n$.
Переслать l_{21} в P_{22}. Вычислить l_{31}

Шаг 4. Переслать a_{1j} из P_{4j} в P_{5j}, $j = 1, \ldots, n$. Переслать l_{21} в P_{23}.
Переслать l_{31} в P_{32}. Вычислить l_{41}. Вычислить $l_{21}a_{12}$ в P_{22}.

Шаг 5. Переслать a_{1j} из P_{5j} в P_{6j}, $j = 1, \ldots, n$. Переслать l_{21} в P_{24}.
Переслать l_{31} в P_{33}. Переслать l_{41} в P_{42}. Вычислить l_{51}
Вычислить $a_{22}^1 = a_{22} - l_{21}a_{12}$ в P_{22}
Вычислить $l_{21}a_{13}$ в P_{23}. Вычислить $l_{31}a_{12}$ в P_{32}

Алгоритм потока данных (продолжение)



Количество временных шагов для завершения факторизации

В конце пятого временного шага вычислен модифицированный элемент a_{22} множитель l_{21} находится в P_{24}, а в P_{23} вычислено произведение $l_{21}a_{13}$.
На каждом из последующих временных шагов завершается модификация очередного элемента, поэтому обработка второй строки будет закончена после 5+n-2 = n+3 временных шагов (ВШ).
Второй этап разложения начинается через четыре временных шага после первого. Т.о. обработка третьей строки завершится через n+7 ВШ.
Продолжая действовать т.о. можно показать, что процесс факторизации потребует n+3+4(n-2) = **5n – 5** ВШ

$$S_p = \frac{O\left(2/3\, n^3\right)}{O(5n)} = O\left(n^2\right)$$

Ленточные матрицы

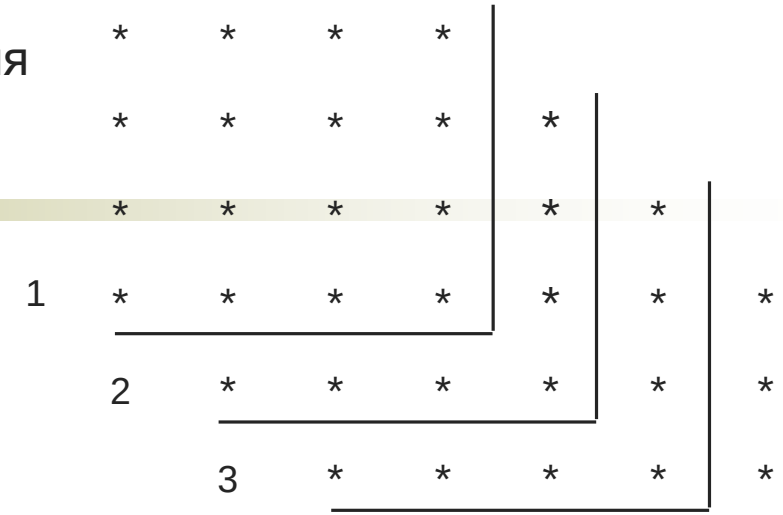
```
Для k = 1 до n – 1
    Для i = k + 1
        до min(k+β, n)
        l_ik = a_ik/a_kk
        Для j = k + 1
            до min(k+β, n)
            a_ij = a_ij – l_ik a_kj
```

```
Для k = 1 до n – 1
    Для s = k + 1
        до min(k+β, n)
        l_sk = a_sk/a_kk
        Для j = k + 1
            до min(k+β, n)
            Для i = k + 1
                до min(k+β, n)
                a_ij = a_ij – l_ik a_kj
```

LU-разложение ленточной матрицы для строчно и столбцово ориентированных алгоритмов

При достаточно больших β псевдокод для полнозаполненных матриц может быть основой как для векторных, так и для параллельных компьютеров. При уменьшении β такой подход совершенно не подходит

Первые шаги *LU*-разложения для ленточной матрицы при $\beta = 3$



Строчная (столбцовая) схема хранения ленточных матриц

$\mathbf{a_1}$	$\mathbf{a_2}$	....	$\mathbf{a_{\beta+1}}$	....	$\mathbf{a_{n-\beta}}$	....	$\mathbf{a_n}$
$\beta+1$	$\beta+2$		$2\beta+1$		$2\beta+1$		$\beta+1$

При параллельной реализации способ решения подобен векторным способам. β может рассматриваться как степень параллелизма.
Для полного использования процессоров должно соблюдаться условие $\beta \geq p$
Однако при выполнении этого условия дисбаланс нагрузки более вероятен

Перестановки

Перестановки, возникающие при выборе главного элемента в ленточной матрице при сохранении *LU*-разложения увеличивают β
Последствия для векторных и параллельных компьютеров более серьезные, чем для последовательных.
Пример. На первом шаге переставлены первая и пятая строка при $\beta=4$

* * * * * * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * *

В результате во 2,3,4,5 строках возникает заполнение. Т.о. ширина ленты над главной диагональю фактически удваивается; так будет и в дальнейшем, если потребуются дополнительные «наихудшие» перестановки

a_1	a_2	....	$a_{\beta+1}$	....	$a_{n-2\beta}$	....	a_n
$2\beta+1$	$2\beta+2$		$3\beta+1$		$3\beta+1$		$\beta+1$

Последовательный вариант решения ленточных матриц

$$Ax = F$$

$$A_i x_{i-1} + B_i x_i + C_i x_{i+1} = F_i$$

Трехдиагональная матрица

Идея метода прогонки

$$x_i = \alpha_{i+1} x_{i+1} + \beta_{i+1}$$

- предположение

$$i = n-1, n-2, \ldots, 1$$

выразим x_{i-1} и x_i через x_{i+1}

$$(A_i \alpha_i \alpha_{i+1} + B_i \alpha_{i+1} + C_i)\, x_{i+1} + A_i \alpha_i \beta_{i+1} + A_i \beta_i + B_i \beta_{i+1} - F_i = 0$$

$$\begin{cases} A_i \alpha_i \alpha_{i+1} + B_i \alpha_{i+1} + C_i = 0 \\ A_i \alpha_i \beta_{i+1} + A_i \beta_i + B_i \beta_{i+1} - F_i = 0 \end{cases}$$

Идея метода прогонки (1)

$$\begin{cases} \alpha_{i+1} = \dfrac{-C_i}{A_i\alpha_i + B_i} \\ \beta_{i+1} = \dfrac{F_i - A_i\beta_i}{A_i\alpha_i + B_i} \end{cases}$$

Из первого уравнения получим:

$$\begin{cases} \alpha_2 = -C_1/B_1 \\ \beta_2 = F_1/B_1 \end{cases}$$

$$x_i = \alpha_{i+1}x_{i+1} + \beta_{i+1}, i = n-1 \ldots 1$$

$$x_n = \frac{F_n - A_n\beta_n}{B_n + A_n\alpha_n}$$

Блочные методы

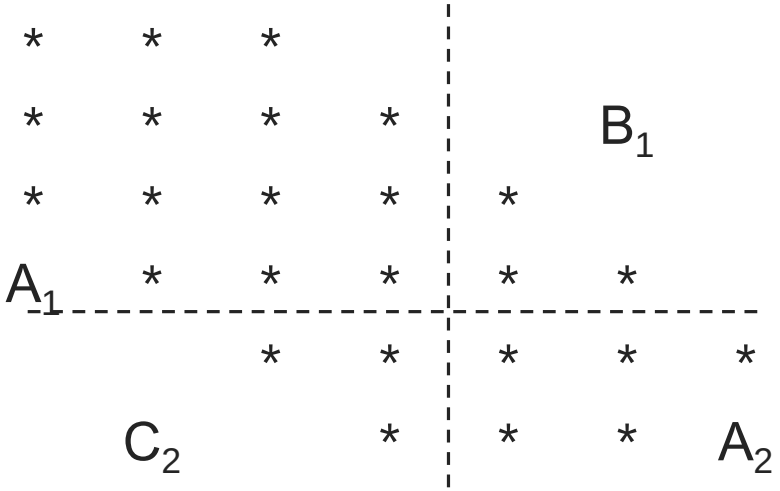
Рассмотренные методы хороши, когда β большая.
Если β небольшое, то можно рассмотреть «блочные методы»

$$\begin{bmatrix} A_1 & B_1 & & & \\ C_2 & A_2 & B_2 & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & B_{p-1} \\ & & & C_p & A_p \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_p \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ b_p \end{bmatrix}$$

Здесь $q = n/p$ и A_i – матрица q x q, x_i, b_i – вектора длины q

Блочные методы (продолжение)

Пусть A_i – невырожденная и достаточно устойчивая к LU-разложению матрица



Решим параллельно *q* частных наборов систем уравнений

$$A_i W_i = B_i; \quad A_i V_i = C_i; \quad A_i d_i = b_i$$

Блочные методы (продолжение)

$$\begin{bmatrix} A_1^{-1} & & & 0 \\ & A_2^{-1} & & \\ & & \ddots & \\ 0 & & & A_p^{-1} \end{bmatrix} \begin{bmatrix} A_1 & B_1 & & \\ C_2 & A_2 & B_2 & \\ & & \ddots & \\ & & C_p & A_p \end{bmatrix} = \begin{bmatrix} A_1^{-1}A_1 & A_1^{-1}B_1 & & \\ A_2^{-1}A_2 & A_2^{-1}A_2 & A_2^{-1}B_2 & \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} I & W_1 & & \\ V_2 & I & W_2 & \\ \cdot & \cdot & \cdot & \cdot \\ & & V_p & I \end{bmatrix}$$

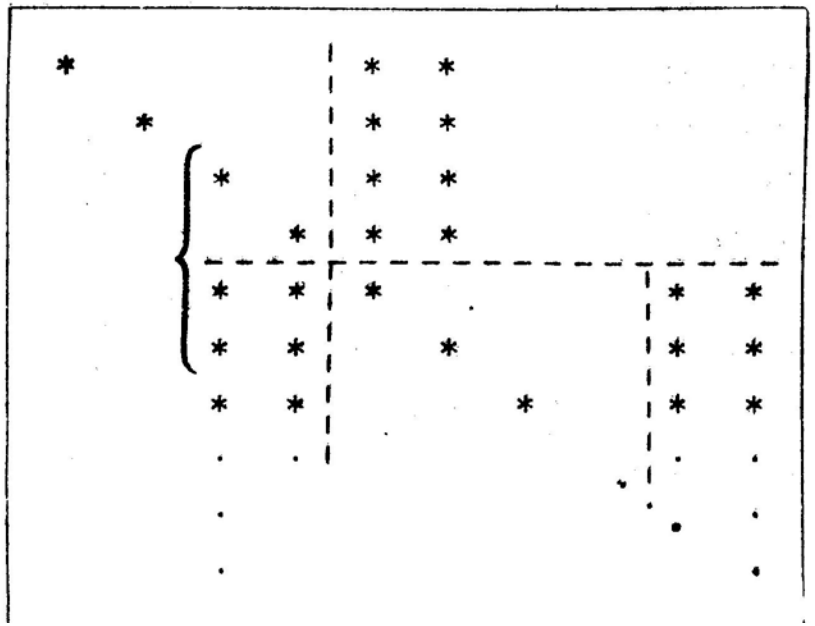
$$\begin{bmatrix} A_1^{-1} & & & 0 \\ & A_2^{-1} & & \\ & & \ddots & \\ 0 & & & A_p^{-1} \end{bmatrix} \begin{bmatrix} b_1 \\ \cdot \\ \cdot \\ b_p \end{bmatrix} = \begin{bmatrix} A_1^{-1}b_1 \\ \cdot \\ \cdot \\ A_p^{-1}b_p \end{bmatrix} = \begin{bmatrix} d_1 \\ \cdot \\ \cdot \\ d_p \end{bmatrix}$$

$$\begin{bmatrix} I & W_1 & & \\ V_2 & I & W_2 & \\ \cdot & \cdot & \cdot & \cdot \\ & & V_p & I \end{bmatrix} \begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ x_p \end{bmatrix} = \begin{bmatrix} d_1 \\ \cdot \\ \cdot \\ d_p \end{bmatrix}$$

Блочные методы (продолжение)

j-й столбец матрицы W_i/V_i заполнен, если не равны нулю соответствующие столбцы в матрицах B_i/C_i

Структура редуцированной системы



Предполагаем, что
W_{i1}, W_{i3}, V_{i1}, V_{i3}
имеют размерность β x β,
а, соответственно, W_{i2}, V_{i2}
имеют размерность $(q - 2\beta)$ x β
Аналогичным образом представлены
x_i и d_i

$$W_i = \begin{bmatrix} W_{i1} & 0 \\ W_{i2} & 0 \\ W_{i3} & 0 \end{bmatrix}, \quad V_i = \begin{bmatrix} 0 & V_{i1} \\ 0 & V_{i2} \\ 0 & V_{i3} \end{bmatrix}, \quad x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \end{bmatrix}, \quad d_i = \begin{bmatrix} d_{i1} \\ d_{i2} \\ d_{i3} \end{bmatrix}$$

Блочные методы (продолжение)

Пользуясь этими представлениями, можно записать первое блочное уравнение

$\mathbf{x_1} + W_1\mathbf{x_2} = \mathbf{d_1}$

$$\begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \end{bmatrix} + \begin{bmatrix} W_{11} \\ W_{12} \\ W_{13} \end{bmatrix} x_{21} = \begin{bmatrix} d_{11} \\ d_{12} \\ d_{13} \end{bmatrix}$$

Не зависят от $\mathbf{x_{i2}}$

Аналогично, второе блочное уравнение $V_2\mathbf{x_1} + \mathbf{x_2} + W_2\mathbf{x_3} = \mathbf{d_2}$

$$\begin{bmatrix} V_{21} \\ V_{22} \\ V_{23} \end{bmatrix} x_{13} + \begin{bmatrix} x_{21} \\ x_{22} \\ x_{23} \end{bmatrix} + \begin{bmatrix} W_{21} \\ W_{22} \\ W_{23} \end{bmatrix} x_{31} = \begin{bmatrix} d_{21} \\ d_{22} \\ d_{23} \end{bmatrix}$$

Тоже самое делается для остальных блочных уравнений.

Блочные методы (продолжение)

$$x_{13} + W_{13} x_{21} = d_{13},$$

$$V_{21} x_{13} + x_{21} + W_{21} x_{31} = d_{21},$$

$$V_{23} x_{13} + x_{23} + W_{23} x_{31} = d_{23},$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

Редуцированная система

$$x_{12} = d_{12} - W_{12} x_{21}, \quad x_{22} = d_{22} - V_{22} x_{13} - W_{22} x_{31}$$

$$x_{i2} = d_{i2} - V_{i2} x_{i-1,3} - W_{i2} x_{i+1,1}$$

Блочный алгоритм Лори-Самеха

- Шаг 1. Выполнить *LU*-разложение для A_i и решить системы для V_i, W_i, $\mathbf{d_i}$
- Шаг 2. Решить редуцированную систему относительно векторов $\mathbf{x_{i1}}$, $\mathbf{x_{i3}}$
- Шаг 3. Вычислить векторы $\mathbf{x_{i2}}$, а затем вычислить $\mathbf{x_{11}}$, $\mathbf{x_{p3}}$

Шаги 1,3 имеют степень параллелизма p и решение i-го уравнения поручается i-му процессору.

Потенциально узкое место – решение редуцированной системы

Блочный алгоритм Лори-Самеха (продолжение)

Порядок редуцированной системы для различных p и β $(2\beta(p-1))$

p \ β	1	2	10	20	50
2	2	4	20	40	100
10	18	36	180	360	900
20	38	76	380	760	1900
50	98	196	980	1960	4900