Введение в методы параллельного программирования

Автор слайдов:
Гергель В.П., профессор, д.т.н.
Декан факультета ВМК ННГУ
http://www.intuit.ru/studies/courses/1021/284/info

Параллельные методы умножения матрицы на вектор

- Постановка задачи
- Способы распределения данных
- Последовательный алгоритм
- Алгоритм 1 – ленточная схема, разделение матрицы по строкам
- Алгоритм 2 – ленточная схема, разделение матрицы по столбцам
- Алгоритм 3 – блочная схема

Постановка задачи

Умножение матрицы на вектор

$$c = A \cdot b$$

или

$$\begin{pmatrix} c_0 \\ c_1 \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} a_{0,0}, & a_{0,1}, & \ldots, & a_{0,n-1} \\ & & \ldots & \\ a_{m-1,0}, & a_{m-1,1}, & \ldots, & a_{m-1,n-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_{n-1} \end{pmatrix}$$

- Задача умножения матрицы на вектор может быть сведена к выполнению ***m*** независимых операций умножения строк матрицы ***A*** на вектор ***b***

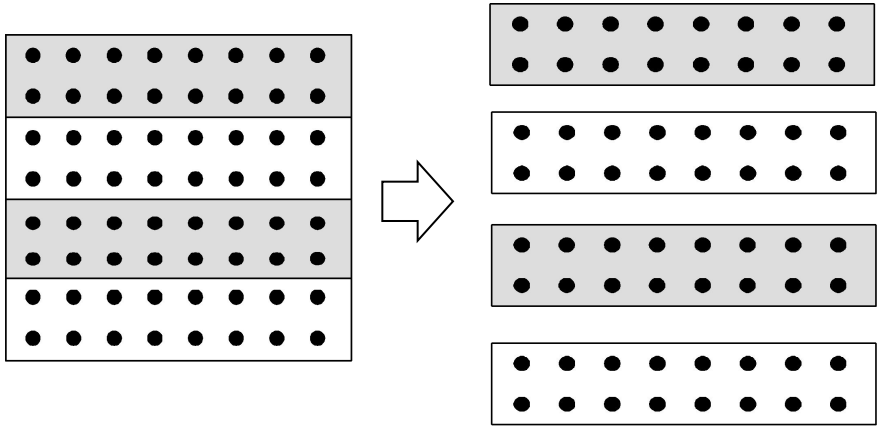
$$c_i = (a_i, b) = \sum_{j=1}^{n} a_{ij} b_j, \quad 0 \le i < m$$

В основу организации параллельных вычислений может быть положен ***принцип распараллеливания по данным***

Способы распределения данных: *ленточная схема*

Непрерывное (последовательное) распределение

горизонтальные полосы



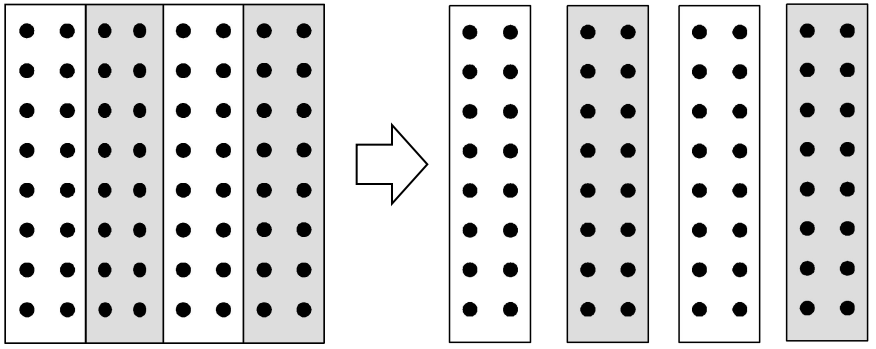
$$A = (A_0, A_1, ..., A_{p-1})^T,$$

$$A_i = (a_{i_0}, a_{i_1}, ..., a_{i_{k-1}}),$$

$$i_j = ik + j, 0 \le j < k, k = m / p$$

(a_i, $0 \le i < m$, – *строки матрицы* A)

вертикальные полосы



$$A = (A_0, A_1, ..., A_{p-1}),$$

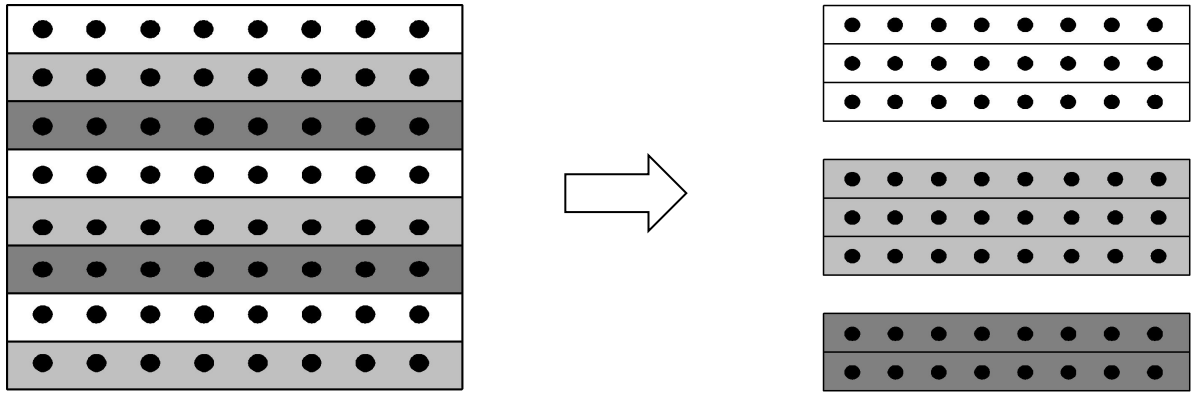
$$A_i = (\alpha_{i_0}, \alpha_{i_1}, ..., \alpha_{i_{k-1}}),$$

$$i_j = il + j, 0 \le j < l, l = n / p$$

(α_i, $0 \le i < m$, – *столбцы матрицы* A)

Способы распределения данных: *ленточная схема*

Чередующееся (цикличное) горизонтальное разбиение

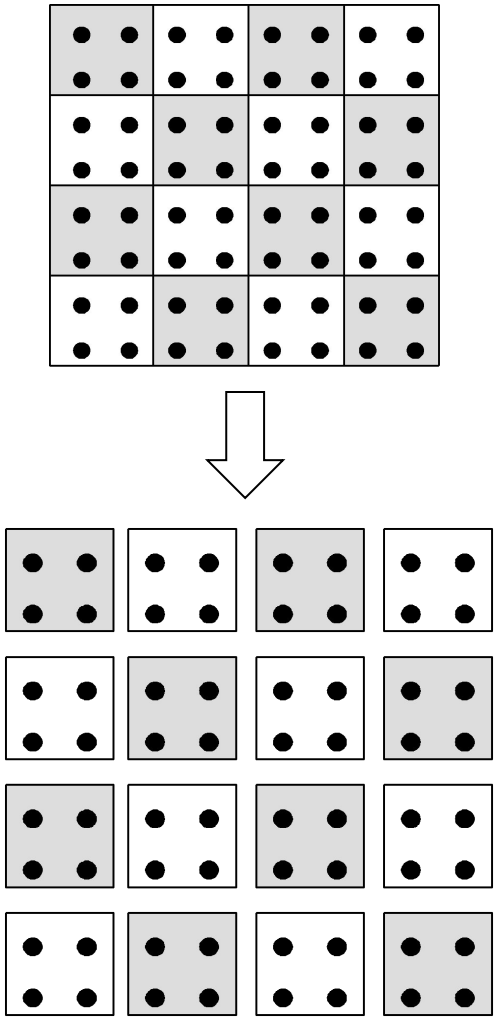


$$A = (A_0, A_2, ..., A_{p-1})^T,$$

$$A_i = (a_{i_0}, a_{i_1}, ..., a_{i_{k-1}}),$$

$$i_j = i + jp, 0 \le j < k, k = m / p$$

Способы распределения данных: *блочная схема*



$$A = \begin{pmatrix} A_{00} & A_{02} & \dots A_{0q-1} \\ & \dots & \\ A_{s-11} & A_{s-12} & \dots A_{s-1q-1} \end{pmatrix},$$

$$A_{ij} = \begin{pmatrix} a_{i_0 j_0} & a_{i_0 j_1} & \dots a_{i_0 j_{l-1}} \\ & \dots & \\ a_{i_{k-1} j_0} & a_{i_{k-1} j_1} & a_{i_{k-1} j_{l-1}} \end{pmatrix},$$

$$i_v = ik + v, 0 \le v < k, k = m / s$$

$$j_u = jl + u, 0 \le u \le l, l = n / q$$

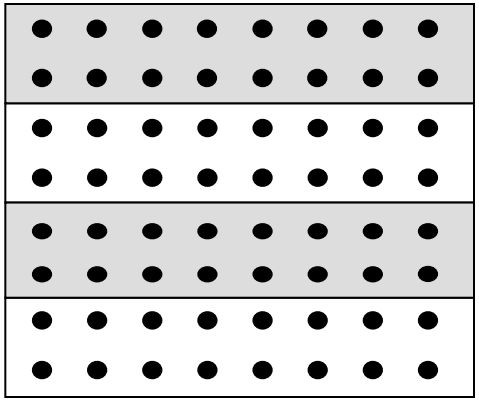
Последовательный алгоритм

```
// Последовательный алгоритм умножения матрицы на
вектор
for ( i = 0; i < m; i++ ) {
  c[i] = 0;
  for ( j = 0; j < n; j++ ) {
    c[i] += A[i][j]*b[j]
  }
}
```

- Для выполнения матрично-векторного умножения необходимо выполнить ***m*** операций вычисления скалярного произведения
- Трудоемкость вычислений имеет порядок *O(mn)*.

Алгоритм 1: ленточная схема (разбиение матрицы по строкам)…

- **Распределение данных** – ленточная схема (разбиение матрицы по строкам)

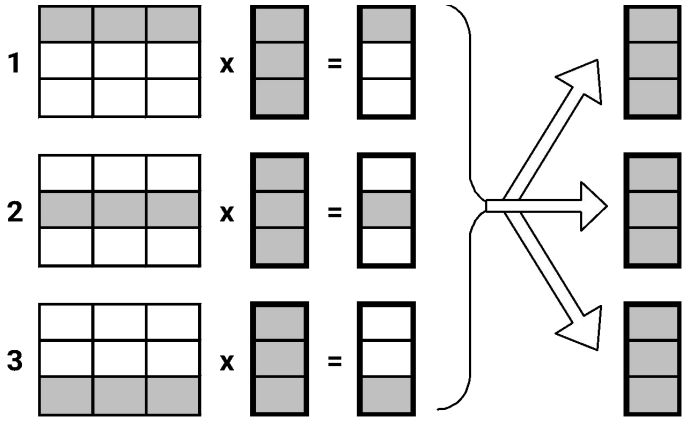


- **Базовая подзадача** - операция скалярного умножения одной строки матрицы на вектор

$$c_i = (a_i, b) = \sum_{j=1}^{n} a_{ij} b_j, \quad 0 \le i < m$$

Алгоритм 1: ленточная схема (разбиение матрицы по строкам)…

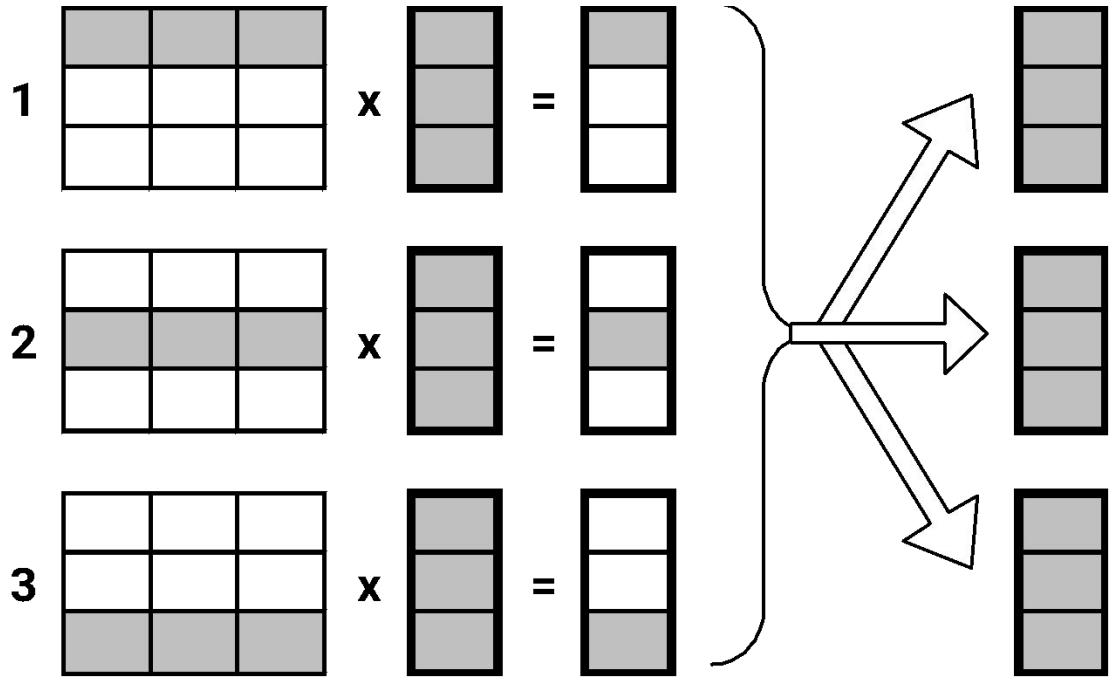
- **Выделение информационных зависимостей**
  - Базовая подзадача для выполнения вычисления должна содержать строку матрицы ***A*** и копию вектора ***b***. После завершения вычислений каждая базовая подзадача будет содержать один из элементов вектора результата ***c***



  - Для объединения результатов расчетов и получения полного вектора ***c*** на каждом из процессоров вычислительной системы необходимо выполнить операцию обобщенного сбора данных

Алгоритм 1: ленточная схема (разбиение матрицы по строкам)…

- Схема информационного взаимодействия



Алгоритм 1: ленточная схема (разбиение матрицы по строкам)…

- **Масштабирование и распределение подзадач по процессорам**
  - Если число процессоров ***p*** меньше числа базовых подзадач ***m (p<m)***, базовые подзадачи могут быть укрупнены с тем, чтобы каждый процессор выполнял несколько операций умножения строк матрицы ***A*** и вектора ***b***. В этом случае, по окончании вычислений каждая базовая подзадача будет содержать набор элементов результирующего вектора ***c***
  - Распределение подзадач между процессорами вычислительной системы может быть выполнено с учетом возможности эффективного выполнения операции обобщенного сбора данных

Алгоритм 1: ленточная схема (разбиение матрицы по строкам)…

- **Анализ эффективности**
  - Общая оценка показателей ускорения и эффективности

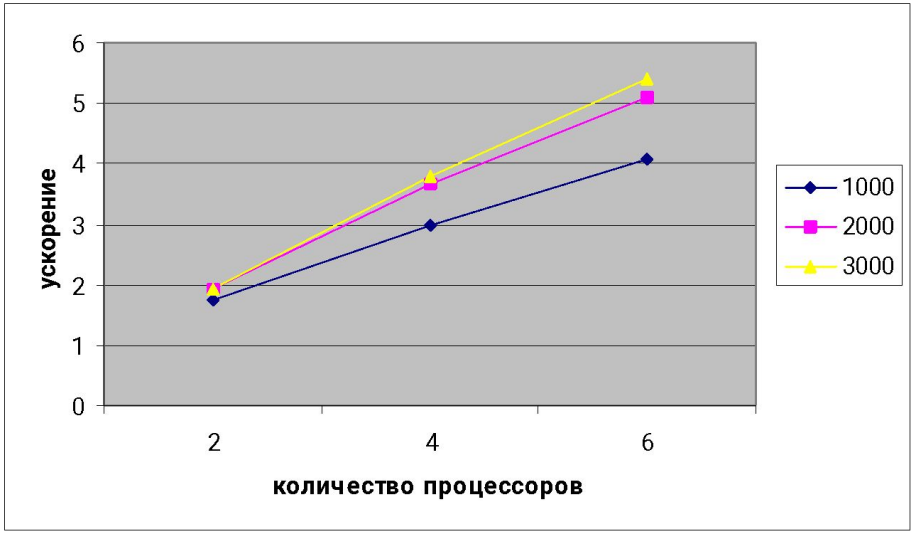
$$S_p = \frac{n^2}{n^2 / p} = p \qquad E_p = \frac{n^2}{p \cdot (n^2 / p)} = 1$$

Разработанный способ параллельных вычислений позволяет достичь идеальных показателей ускорения и эффективности

Алгоритм 1: ленточная схема (разбиение матрицы по строкам)

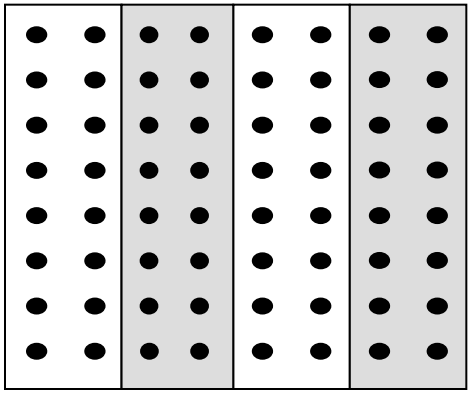
- **Результаты вычислительных экспериментов**
  - Ускорение вычислений

Размер матрицы	Последовательный алгоритм	Параллельный алгоритм					
		2 процессора		4 процессора		6 процессоров	
		Время	Ускорение	Время	Ускорение	Время	Ускорение
1000x1000	0,0291	0,0166	1,7471	0,0097	2,9871	0,0071	4,0686
2000x2000	0,1152	0,0601	1,9172	0,0313	3,6775	0,0217	5,1076
3000x3000	0,2565	0,1336	1,9203	0,0675	3,7991	0,0459	5,4181

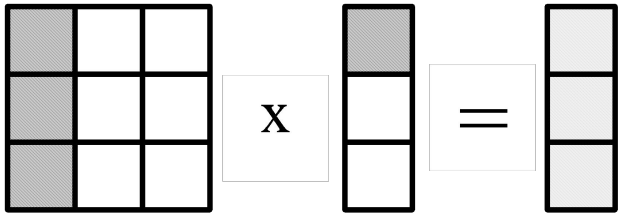


Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)…

- **Распределение данных** – ленточная схема (разбиение матрицы по столбцам)



- **Базовая подзадача** - операция умножения столбца матрицы *A* на один из элементов вектора *b*

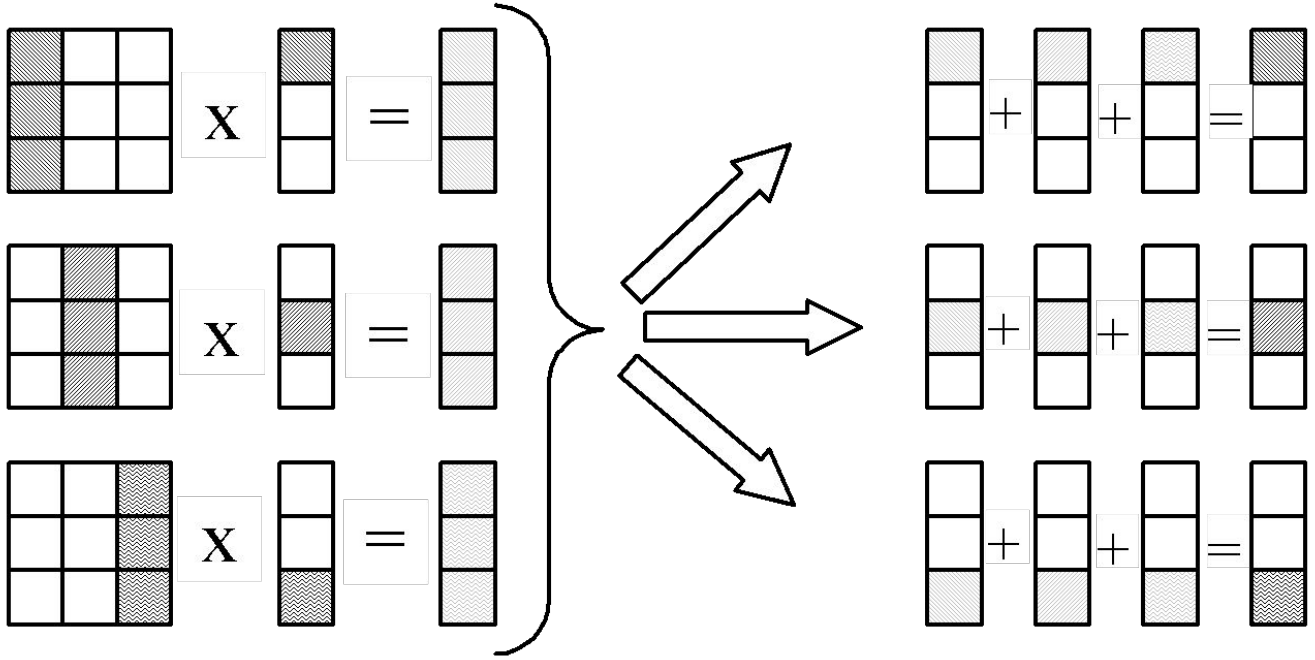


Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)…

- **Выделение информационных зависимостей**
  - Базовая подзадача для выполнения вычисления должна содержать должна содержать ***i***-й столбец матрицы ***A*** и ***i***-е элементы b_i и c_i векторов ***b*** и ***c***
  - Каждая базовая задача ***i*** выполняет умножение своего столбца матрицы ***A*** на элемент b_i, в итоге в каждой подзадаче получается вектор ***c'(i)*** промежуточных результатов
  - Для получения элементов результирующего вектора ***c*** подзадачи должны обменяться своими промежуточными данными

Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)…

- Схема информационного взаимодействия



Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)…

- **Масштабирование и распределение подзадач по процессорам**
  - В случае, когда количество столбцов матрицы ***n*** превышает число процессоров ***p***, базовые подзадачи можно укрупнить, объединив в рамках одной подзадачи несколько соседних столбцов (в этом случае исходная матрица ***A*** разбивается на ряд вертикальных полос). В этом случае, по окончании вычислений и проведения операции обмена каждая базовая подзадача будет содержать набор элементов результирующего вектора ***c***
  - Распределение подзадач между процессорами вычислительной системы может быть выполнено с учетом возможности эффективного выполнения операции обмена элементами векторов частичных результатов

Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)…

- **Анализ эффективности**
  - Общая оценка показателей ускорения и эффективности

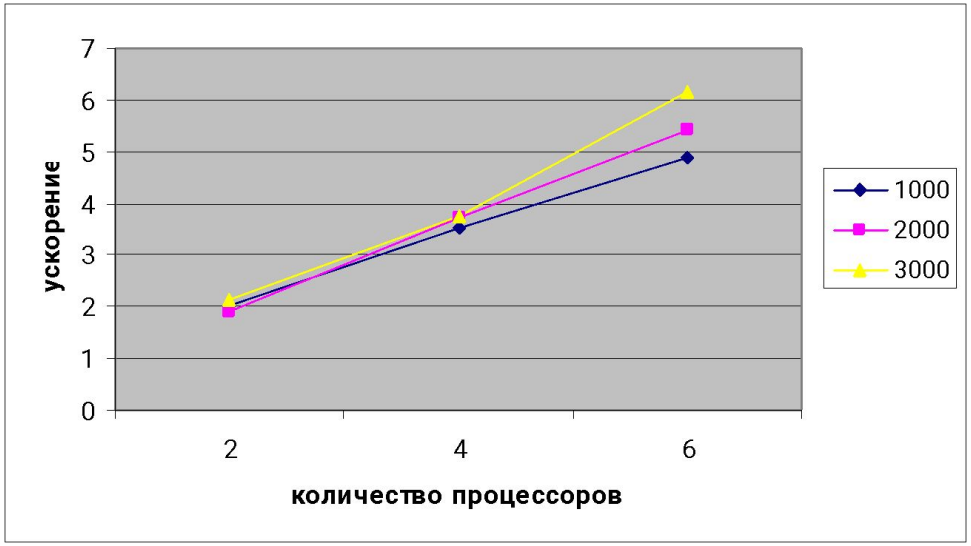
$$S_p = \frac{n^2}{n^2 / p} = p \qquad E_p = \frac{n^2}{p \cdot (n^2 / p)} = 1$$

Разработанный способ параллельных вычислений позволяет достичь идеальных показателей ускорения и эффективности

Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)

- **Результаты вычислительных экспериментов**
  - Ускорение вычислений

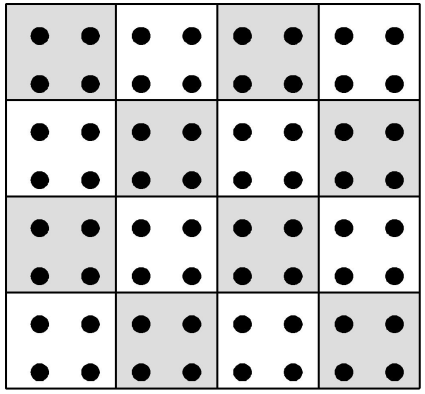
Размер матрицы	Последовательный алгоритм	2 процессора		4 процессора		6 процессоров	
		Время	Ускорение	Время	Ускорение	Время	Ускорение
1000x1000	0,0291	0,0144	2,0225	0,0083	3,5185	0,00595	4,8734
2000x2000	0,1152	0,0610	1,8869	0,0311	3,7077	0,0213	5,4135
3000x3000	0,2565	0,1201	2,1364	0,0683	3,7528	0,04186	6,1331



Алгоритм 3: блочная схема…

- **Распределение данных** – блочная схема

  предполагается, что количество процессоров ***p=s·q*** , количество строк матрицы является кратным ***s***, а количество столбцов – кратным ***q***, то есть ***m=k·s*** и ***l=n·q***.



$$A = \begin{pmatrix} A_{00} & A_{02} & \dots A_{0q-1} \\ & \dots & \\ A_{s-11} & A_{s-12} & \dots A_{s-1q-1} \end{pmatrix} \quad A_{ij} = \begin{pmatrix} a_{i_0 j_0} & a_{i_0 j_1} & \dots a_{i_0 j_{l-1}} \\ & \dots & \\ a_{i_{k-1} j_0} & a_{i_{k-1} j_1} & a_{i_{k-1} j_{l-1}} \end{pmatrix}$$

$$i_v = ik + v, 0 \le v < k, k = m / s$$
$$j_u = jl + u, 0 \le u \le l, l = n / q$$

Алгоритм 3: блочная схема…

- **Базовая подзадача** определяется на основе вычислений, выполняемых над матричными блоками:
  - Подзадачи нумеруются индексами *(i, j)* располагаемых в подзадачах матричных блоков
  - Подзадачи выполняют умножение содержащегося в них блока матрицы ***A*** на блок вектора ***b***

$$b(i,j) = (b_0(i,j), \ldots, b_{l-1}(i,j))^T, \quad b_u(i,j) = b_{j_u}, j_u = jl + u, 0 \le u < l, l = n/q$$

  - После перемножения блоков матрицы ***A*** и вектора ***b*** каждая подзадача *(i,j)* будет содержать вектор частичных результатов ***c'(i,j)***,

$$c'_v(i,j) = \sum_{u=0}^{l-1} a_{i_v j_u} b_{j_u}, i_v = ik + v, 0 \le v < k, k = m/s,$$
$$j_u = jl + u, 0 \le u \le l, l = n/q$$

Алгоритм 3: блочная схема…

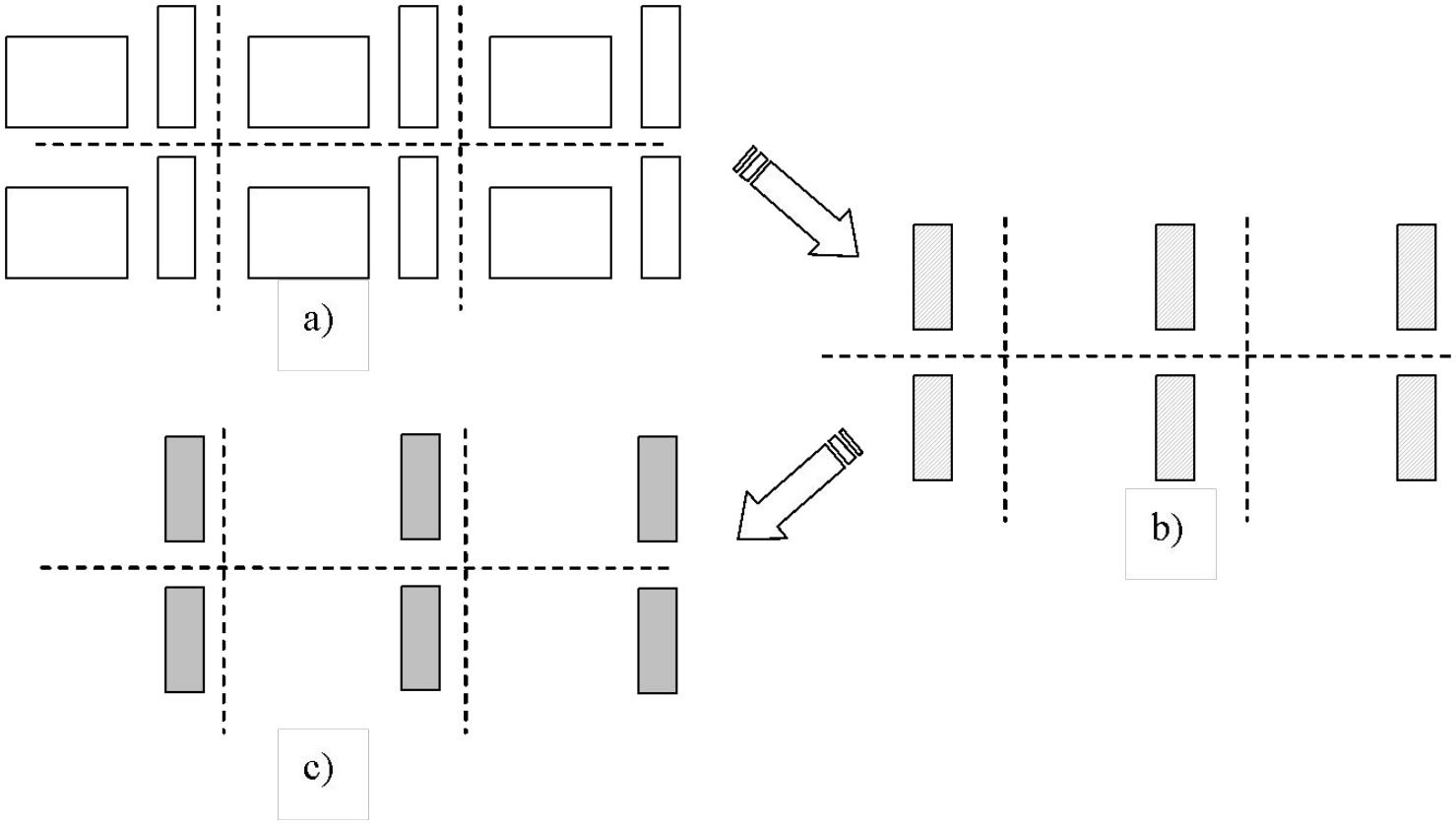
- **Выделение информационных зависимостей**
  - Поэлементное суммирование векторов частичных результатов для каждой горизонтальной полосы (*редукция*) блоков матрицы ***A*** позволяет получить результирующий вектор ***c***

$$c_\eta = \sum_{j=0}^{q-1} c'_\nu (i, j),\ 0 \le \eta < m,\ i = \eta / s,\ \nu = \eta - i \cdot s$$

  - организуем вычисления таким образом, чтобы при завершении расчетов вектор ***c*** располагался поблочно в каждой из вертикальных полос блоков матрицы ***A***
  - информационная зависимость базовых подзадач проявляется только на этапе суммирования результатов перемножения блоков матрицы ***A*** и блоков вектора ***b***

Алгоритм 3: блочная схема…

- Схема информационного взаимодействия



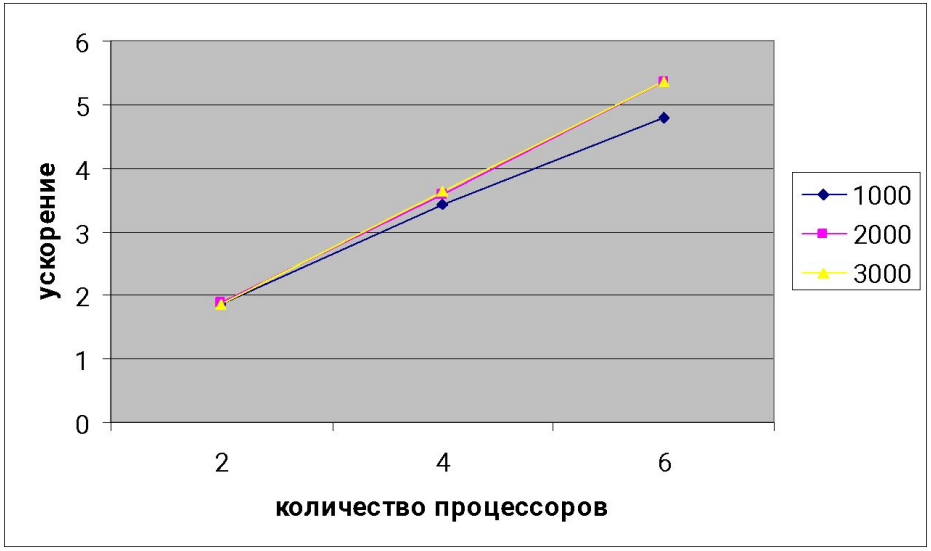
Алгоритм 3: блочная схема…

- **Масштабирование и распределение подзадач по процессорам**
  - Размер блоков матрицы ***A*** может быть подобран таким образом, чтобы общее количество базовых подзадач совпадало с числом процессоров ***p, p=s·q***.
    - Большое количество блоков по горизонтали (***s***) приводит к возрастанию числа итераций в операции редукции результатов блочного умножения,
    - увеличение размера блочной решетки по вертикали (***q***) повышает объем передаваемых данных между процессорами.
  - При решении вопроса распределения подзадач между процессорами должна учитываться возможность эффективного выполнения операции редукции.

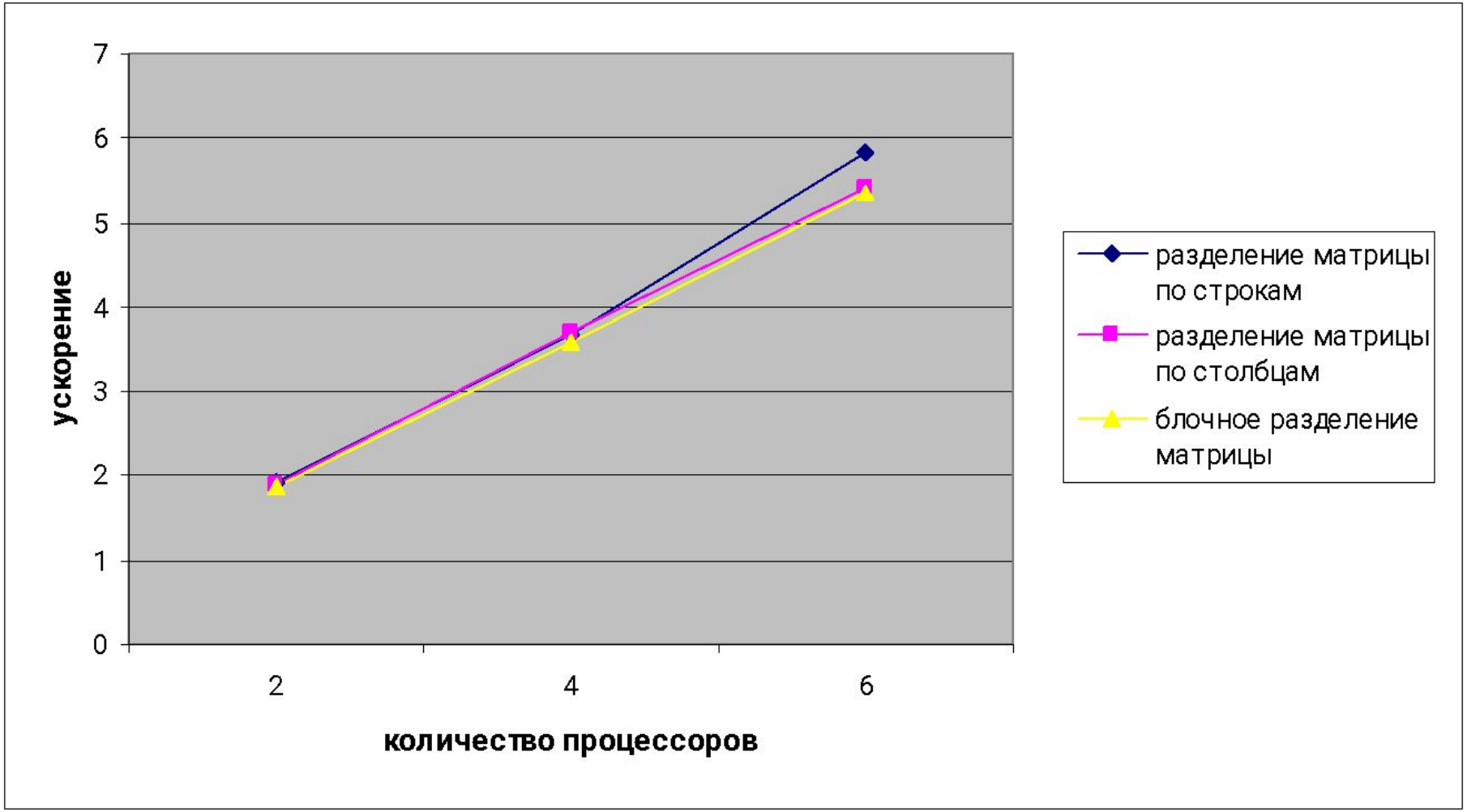
Алгоритм 3: блочная схема

- **Результаты вычислительных экспериментов**
  - Ускорение вычислений

Размер матрицы	Последовательный алгоритм	2 процессора		4 процессора		6 процессоров	
		Время	Ускорение	Время	Ускорение	Время	Ускорение
1000x1000	0,0291	0,0157	1,8515	0,0085	3,4252	0,0061	4,7939
2000x2000	0,1152	0,0614	1,8768	0,0322	3,5815	0,0215	5,3456
3000x3000	0,2565	0,1378	1,8606	0,0705	3,6392	0,0478	5,3620



Сравнение алгоритмов



Параллельные методы умножения матриц

- Постановка задачи
- Последовательный алгоритм
- Алгоритм 1 – ленточная схема
- Алгоритм 2 – метод Фокса
- Алгоритм 3 – метод Кэннона

Постановка задачи

Умножение матриц:

$$C = A \cdot B$$

или

$$\begin{pmatrix} c_{0,0}, & c_{0,1}, & \ldots, & c_{0,l-1} \\ & & \ldots & \\ c_{m-1,0}, & c_{m-1,1}, & \ldots, & c_{m-1,l-1} \end{pmatrix} = \begin{pmatrix} a_{0,0}, & a_{0,1}, & \ldots, & a_{0,n-1} \\ & & \ldots & \\ a_{m-1,0}, & a_{m-1,1}, & \ldots, & a_{m-1,n-1} \end{pmatrix} \begin{pmatrix} b_{0,0}, & b_{0,1}, & \ldots, & a_{0,l-1} \\ & & \ldots & \\ b_{n-1,0}, & b_{n-1,1}, & \ldots, & b_{n-1,l-1} \end{pmatrix}$$

- Задача умножения матрицы на вектор может быть сведена к выполнению ***m·n*** независимых операций умножения строк матрицы ***A*** на столбцы матрицы ***B***

$$c_{ij} = \left(a_i, b_j^T\right) = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}, 0 \le i < m, 0 \le j < l$$

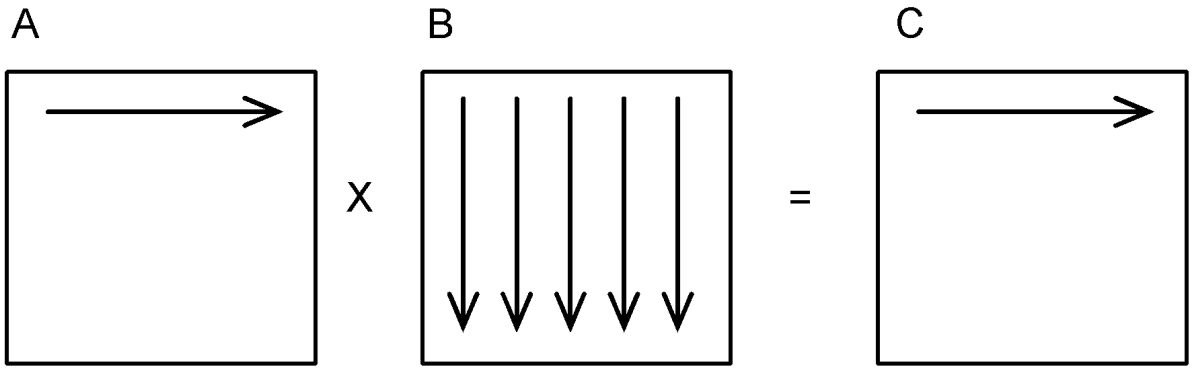
В основу организации параллельных вычислений может быть положен ***принцип распараллеливания по данным***

Последовательный алгоритм…

```
// Последовательный алгоритм матричного умножения
double MatrixA[Size][Size];
double MatrixB[Size][Size];
double MatrixC[Size][Size];
int i,j,k;
...
for (i=0; i<Size; i++){
  for (j=0; j<Size; j++){
    MatrixC[i][j] = 0;
    for (k=0; k<Size; k++){
      MatrixC[i][j] = MatrixC[i][j] + MatrixA[i][k]*MatrixB[k][j];
    }
  }
}
```

Последовательный алгоритм

- Алгоритм осуществляет последовательное вычисление строк матрицы ***C***
- На одной итерации цикла по переменной ***i*** используется первая строка матрицы ***A*** и все столбцы матрицы ***B***



- Для выполнения матрично-векторного умножения необходимо выполнить ***m·n*** операций вычисления скалярного произведения
- Трудоемкость вычислений имеет порядок *O(mnl)*.

Параллельный алгоритм 1: ленточная схема…

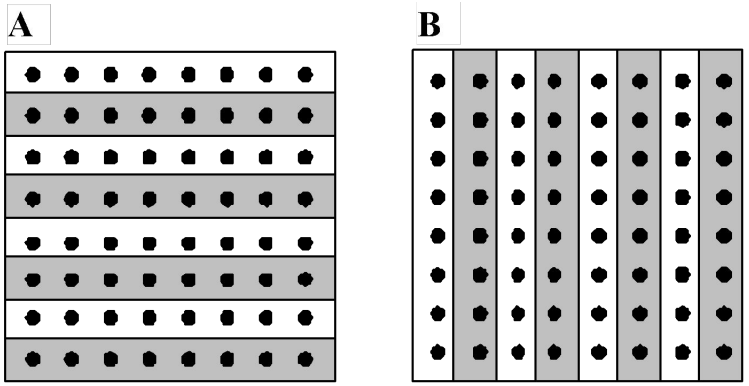
- **Возможный подход** – в качестве базовой подзадачи процедура вычисления одного из элементов матрицы ***C***

$$c_{ij} = \left(a_i, b_j^T\right),\ a_i = \left(a_{i0,} a_{i1}, \ldots, a_{in-1}\right),\ b_j^T = \left(b_{0j}, b_{1j}, \ldots, b_{n-1j}\right)^T$$

- Достигнутый уровень параллелизма - количество базовых подзадач равно n^2 – является избыточным!
- Как правило $p<n^2$ и необходимым является масштабирование параллельных вычислений

Параллельный алгоритм 1: ленточная схема…

- **Базовая подзадача** (агрегация) - процедура вычисления всех элементов одной из строк матрицы ***C*** (количество подзадач равно ***n***)
- **Распределение данных** – ленточная схема (разбиение матрицы ***A*** по строкам и матрицы ***B*** по столбцам)



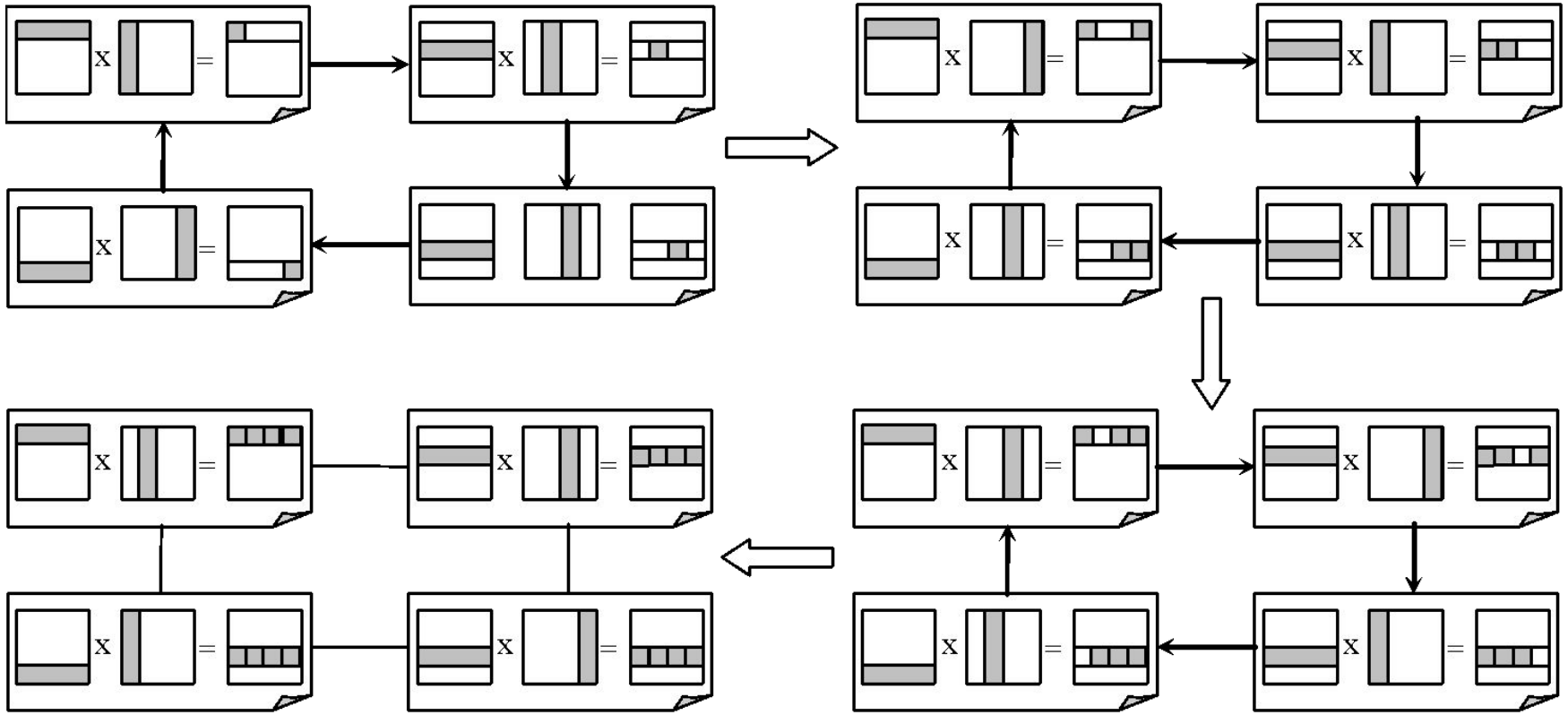
Параллельный алгоритм 1: ленточная схема…

- **Выделение информационных зависимостей**
  - Каждая подзадача содержит по одной строке матрицы ***A*** и одному столбцу матрицы ***B***,
  - На каждой итерации проводится скалярное умножение содержащихся в подзадачах строк и столбцов, что приводит к получению соответствующих элементов результирующей матрицы ***C***,
  - На каждой итерации каждая подзадача i, $0 \le i < n$, передает свой столбец матрицы ***B*** подзадаче с номером $(i+1) \bmod n$.

После выполнения всех итераций алгоритма в каждой подзадаче поочередно окажутся все столбцы матрицы ***B***.

Параллельный алгоритм 1: ленточная схема…

- Схема информационного взаимодействия



Параллельный алгоритм 1: ленточная схема…

- **Масштабирование и распределение подзадач по процессорам**
  - Если число процессоров ***p*** меньше числа базовых подзадач ***n*** ***(p<n)***, базовые подзадачи могут быть укрупнены с тем, чтобы каждый процессор вычислял несколько строк результирующей матрицы ***C***,
  - В этом случае, исходная матрица ***A*** разбивается на ряд горизонтальных полос, а матрица ***B*** представляется в виде набора вертикальных полос,
  - Для распределения подзадач между процессорами может быть использован любой способ, обеспечивающий эффективное представление кольцевой структуры информационного взаимодействия подзадач.

Параллельный алгоритм 1: ленточная схема…

- **Анализ эффективности**
  - Общая оценка показателей ускорения и эффективности

$$S_p = \frac{n^3}{(n^3/p)} = p \qquad E_p = \frac{n^3}{p \cdot (n^3/p)} = 1$$

Разработанный способ параллельных вычислений позволяет достичь идеальных показателей ускорения и эффективности

Параллельный алгоритм 1: ленточная схема…

- **Анализ эффективности** (уточненные оценки)

- Время выполнения параллельного алгоритма, связанное непосредственно с вычислениями, составляет

$$T_p(calc) = (n^2 / p) \cdot (2n - 1) \cdot \tau$$

- Оценка трудоемкости выполняемых операций передачи данных может быть определена как

$$T_p(comm) = (p - 1) \cdot (\alpha + w \cdot n \cdot (n/p) / \beta)$$

(предполагается, что все операции передачи данных между процессорами в ходе одной итерации алгоритма могут быть выполнены параллельно)

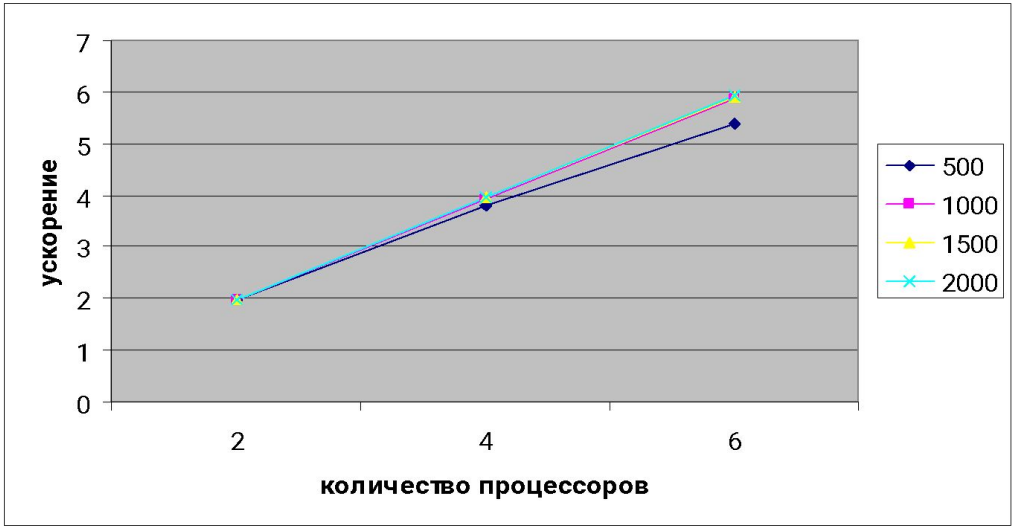
Общее время выполнения параллельного алгоритма составляет

$$T_p = (n^2 / p)(2n - 1) \cdot \tau + (p - 1) \cdot (\alpha + w \cdot n \cdot (n/p) / \beta)$$

Параллельный алгоритм 1: ленточная схема…

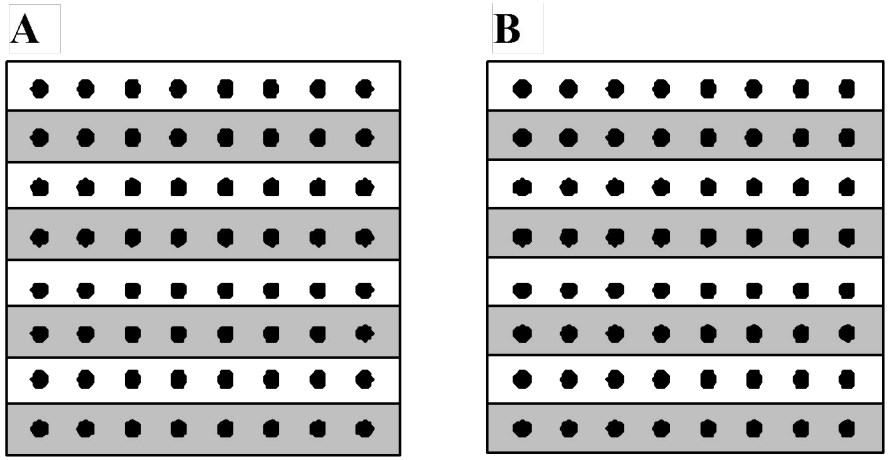
- **Результаты вычислительных экспериментов**
  - Ускорение вычислений

Размер матрицы	Последовательный алгоритм	2 процессора		4 процессора		6 процессоров	
		Время	Ускорение	Время	Ускорение	Время	Ускорение
500x500	2,0628	1,0521	1,9607	0,5454	3,7825	0,3825	5,3925
1000x1000	16,5152	8,3916	1,9681	4,2255	3,9084	2,8196	5,8573
1500x1500	56,5660	28,6602	1,9737	14,311	3,9526	9,5786	5,9055
2000x2000	133,9128	67,8705	1,9731	33,928	3,9469	22,545	5,9399



Параллельный алгоритм 1': ленточная схема…

- Другой возможный вариант распределения данных состоит в разбиении матриц ***A*** и ***B*** по строкам)



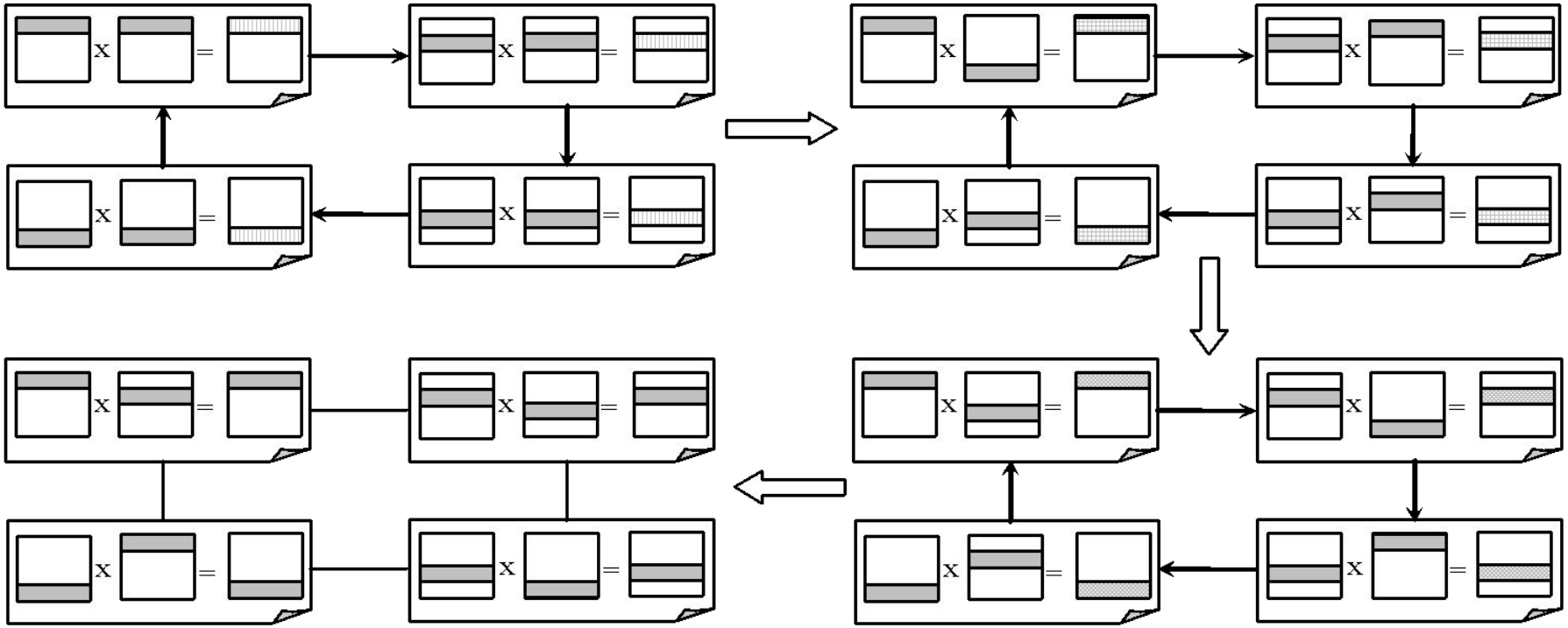
Параллельный алгоритм 1': ленточная схема…

- **Выделение информационных зависимостей**
  - Каждая подзадача содержит по одной строке матриц ***A*** и ***B***,
  - На каждой итерации подзадачи выполняют поэлементное умножение векторов, в результате в каждой подзадаче получается строка частичных результатов для матрицы ***C***,
  - На каждой итерации подзадача i, $0 \le i < n$, передает свою строку матрицы ***B*** подзадаче с номером $(i+1) \bmod n$.

После выполнения всех итераций алгоритма в каждой подзадаче поочередно окажутся все строки матрицы ***B***

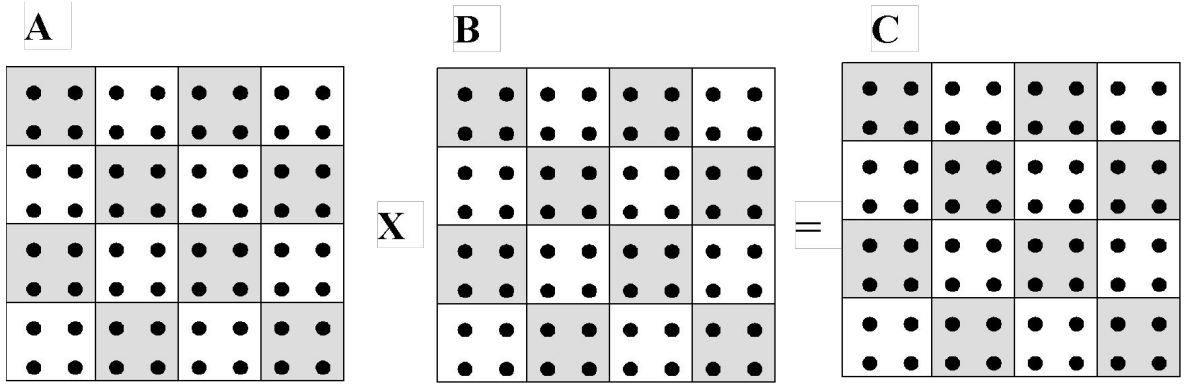
Параллельный алгоритм 1': ленточная схема

- Схема информационного взаимодействия



Параллельный алгоритм 2: метод Фокса

- **Распределение данных** – Блочная схема



- **Базовая подзадача** - процедура вычисления всех элементов одного из блоков матрицы *C*

$$\begin{pmatrix} A_{00}A_{01}\ldots A_{0q-1} \\ \boxtimes \\ A_{q-10}A_{q-11}\ldots A_{q-1q-1} \end{pmatrix} \times \begin{pmatrix} B_{00}B_{01}\ldots B_{0q-1} \\ \boxtimes \\ B_{q-10}B_{q-11}\ldots B_{q-1q-1} \end{pmatrix} = \begin{pmatrix} C_{00}C_{01}\ldots C_{0q-1} \\ \boxtimes \\ c_{q-10}C_{q-11}\ldots C_{q-1q-1} \end{pmatrix}, \quad C_{ij} = \sum_{s=1}^{q} A_{is}B_{sj}$$

Параллельный алгоритм 2: метод Фокса…

- **Выделение информационных зависимостей**
  - Подзадача *(i,j)* отвечает за вычисление блока C_{ij}, как результат, все подзадачи образуют прямоугольную решетку размером $q \times q$,
  - В ходе вычислений в каждой подзадаче располагаются четыре матричных блока:
    - блок C_{ij} матрицы C, вычисляемый подзадачей,
    - блок A_{ij} матрицы A, размещаемый в подзадаче перед началом вычислений,
    - блоки A'_{ij}, B'_{ij} матриц A и B, получаемые подзадачей в ходе выполнения вычислений.

Параллельный алгоритм 2: метод Фокса…

- **Выделение информационных зависимостей -** для каждой итерации *l, 0≤ l<q,* выполняется:
  - блок A_{ij} подзадачи *(i,j)* пересылается на все подзадачи той же строки *i* решетки; индекс *j*, определяющий положение подзадачи в строке, вычисляется в соответствии с выражением:

    $$j = (i+l) \bmod q,$$

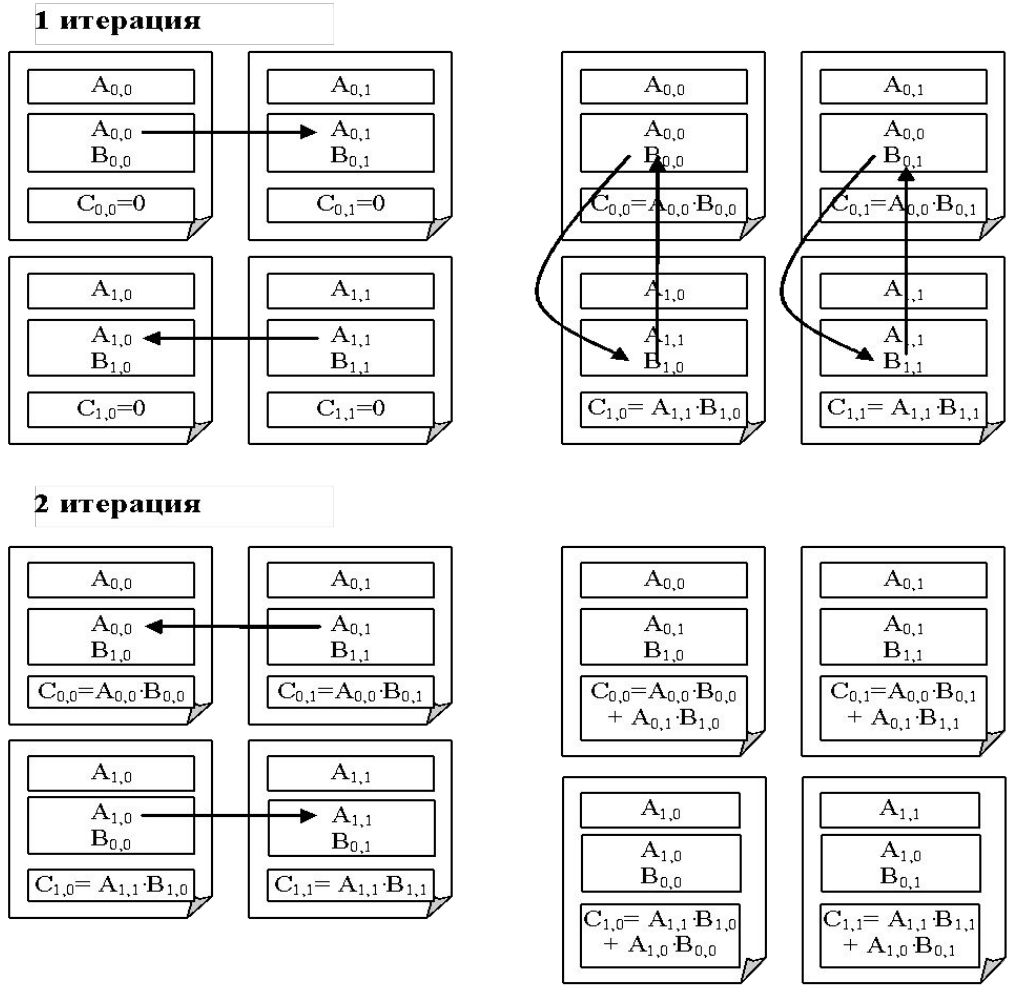
    где *mod* есть операция получения остатка от целого деления;
  - полученные в результате пересылок блоки A_{ij}', B_{ij}' каждой подзадачи *(i,j)* перемножаются и прибавляются к блоку C_{ij}

    $$C_{ij} = C_{ij} + A'_{ij} \times B'_{ij}$$

  - блоки B_{ij}' каждой подзадачи *(i,j)* пересылаются подзадачам, являющимися соседями сверху в столбцах решетки подзадач (блоки подзадач из первой строки решетки пересылаются подзадачам последней строки решетки).

Параллельный алгоритм 2: метод Фокса…

- Схема информационного взаимодействия



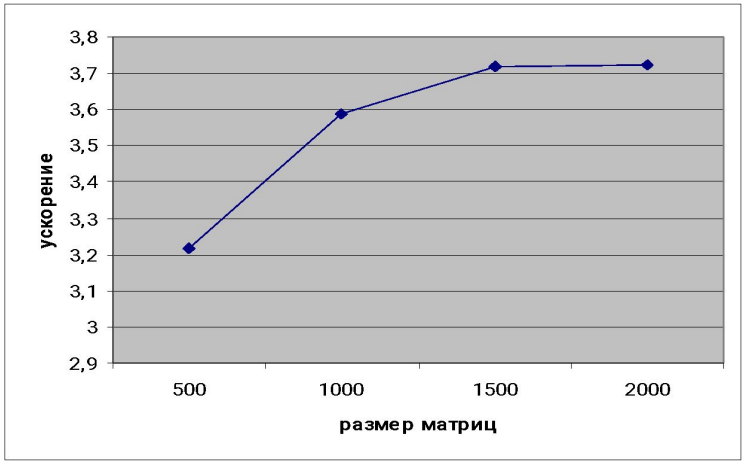
Параллельный алгоритм 2: метод Фокса…

- **Масштабирование и распределение подзадач по процессорам**
  - Размеры блоков могут быть подобраны таким образом, чтобы общее количество базовых подзадач совпадало с числом процессоров ***p***,
  - Наиболее эффективное выполнение метода Фокса может быть обеспечено при представлении множества имеющихся процессоров в виде квадратной решетки,
  - В этом случае можно осуществить непосредственное отображение набора подзадач на множество процессоров - базовую подзадачу (*i,j*) следует располагать на процессоре $p_{i,j}$

Параллельный алгоритм 2: метод Фокса

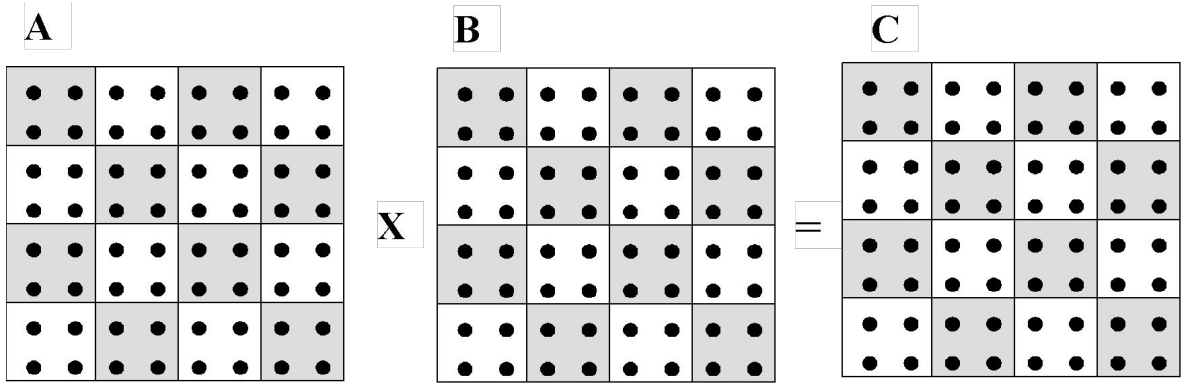
- **Результаты вычислительных экспериментов**
  - Ускорение вычислений

Размер матриц	Последовательный алгоритм	Параллельный алгоритм, 4 процессора	
		Время	Ускорение
500×500	2,0628	0,6417	3,2146
1000×1000	16,5152	4,6018	3,5889
1500×1500	56,566	15,2201	3,7165
2000×2000	133,9128	35,9625	3,7237



Параллельный алгоритм 3: метод Кэннона…

- **Распределение данных** – Блочная схема



- **Базовая подзадача** - процедура вычисления всех элементов одного из блоков матрицы ***C***

$$\begin{pmatrix} A_{00}A_{01}\ldots A_{0q-1} \\ \boxtimes \\ A_{q-10}A_{q-11}\ldots A_{q-1q-1} \end{pmatrix} \times \begin{pmatrix} B_{00}B_{01}\ldots B_{0q-1} \\ \boxtimes \\ B_{q-10}B_{q-11}\ldots B_{q-1q-1} \end{pmatrix} = \begin{pmatrix} C_{00}C_{01}\ldots C_{0q-1} \\ \boxtimes \\ c_{q-10}C_{q-11}\ldots C_{q-1q-1} \end{pmatrix}, \quad C_{ij} = \sum_{s=1}^{q} A_{is}B_{sj}$$

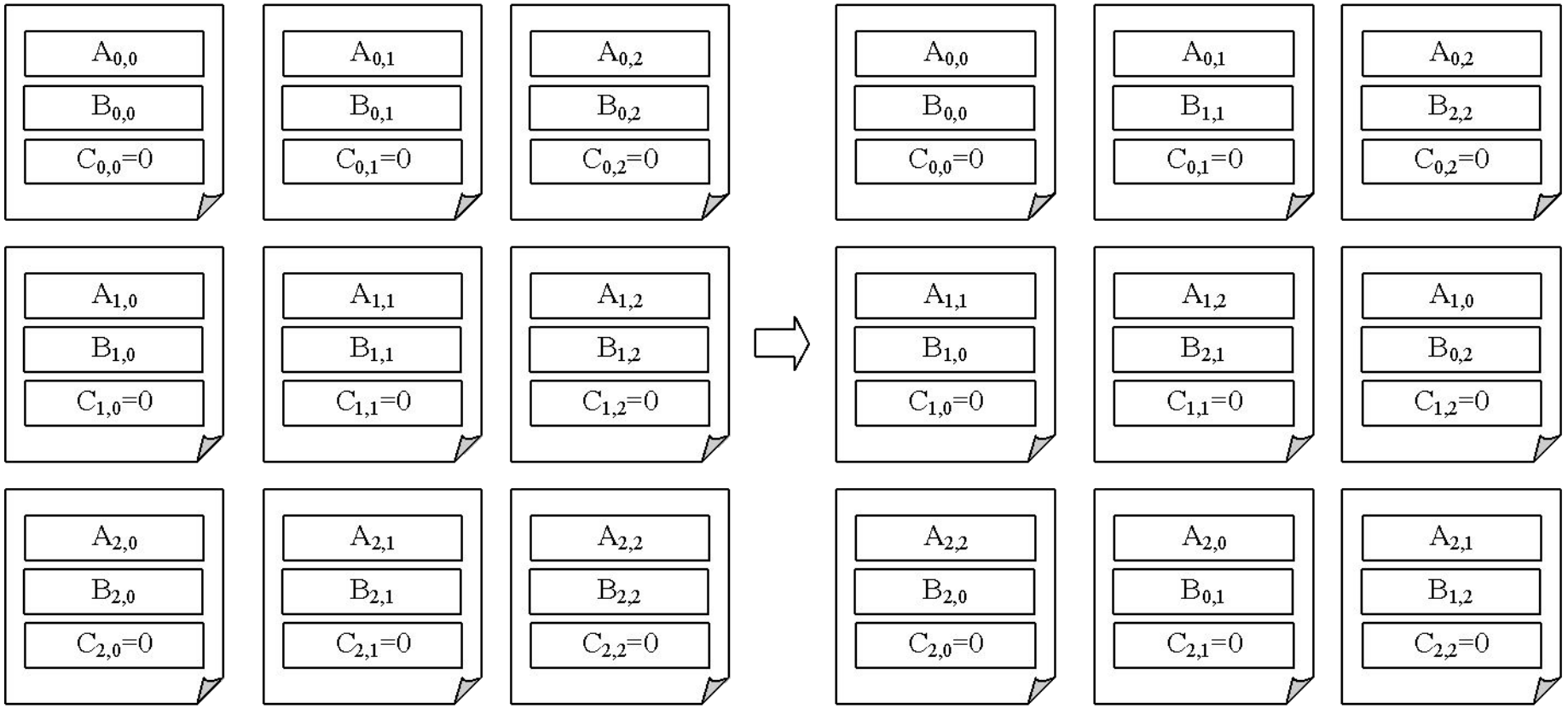
Параллельный алгоритм 3: метод Кэннона…

Выделение информационных зависимостей

- Подзадача *(i,j)* отвечает за вычисление блока C_{ij}, все подзадачи образуют прямоугольную решетку размером *q*x*q*,
- Начальное расположение блоков в алгоритме Кэннона подбирается таким образом, чтобы располагаемые блоки в подзадачах могли бы быть перемножены без каких-либо дополнительных передач данных:
  - в каждую подзадачу *(i,j)* передаются блоки A_{ij}, B_{ij},
  - для каждой строки *i* решетки подзадач блоки матрицы ***A*** сдвигаются на *(i-1)* позиций влево,
  - для каждого столбца *j* решетки подзадач блоки матрицы ***B*** сдвигаются на *(j-1)* позиций вверх,
- процедуры передачи данных являются примером операции *циклического сдвига*

Параллельный алгоритм 3: метод Кэннона…

- Перераспределение блоков исходных матриц на начальном этапе выполнения метода



Параллельный алгоритм 3: метод Кэннона…

- **Выделение информационных зависимостей**
  - В результате начального распределения в каждой базовой подзадаче будут располагаться блоки, которые могут быть перемножены без дополнительных операций передачи данных,
  - Для получения всех последующих блоков после выполнения операции блочного умножения:
    - каждый блок матрицы ***A*** передается предшествующей подзадаче влево по строкам решетки подзадач,
    - каждый блок матрицы ***B*** передается предшествующей подзадаче вверх по столбцам решетки.

Параллельный алгоритм 3: метод Кэннона…

- **Масштабирование и распределение подзадач по процессорам**
  - Размер блоков может быть подобран таким образом, чтобы количество базовых подзадач совпадало с числом имеющихся процессоров,
  - Множество имеющихся процессоров представляется в виде квадратной решетки и размещение базовых подзадач (*i,j*) осуществляется на процессорах $\boldsymbol{p}_{i,j}$ (соответствующих узлов процессорной решетки)

Параллельный алгоритм 3: метод Кэннона…

- **Анализ эффективности**
  - Общая оценка показателей ускорения и эффективности

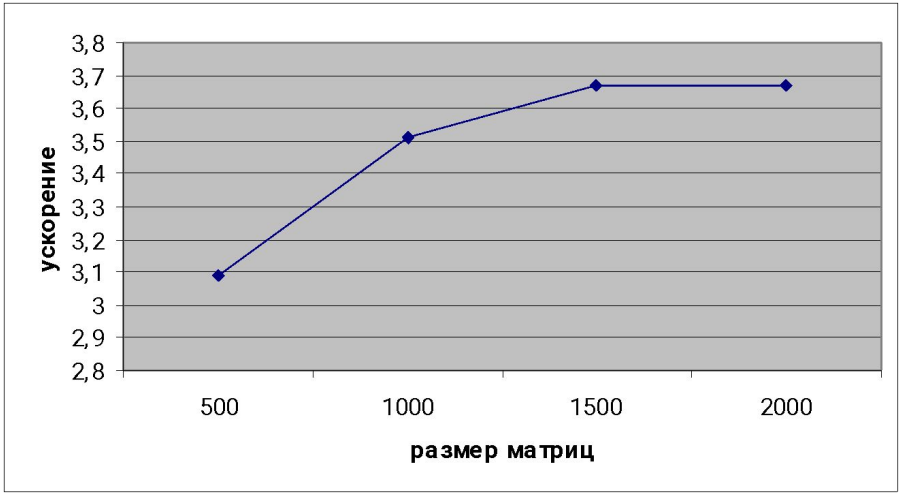
$$S_p = \frac{n^2}{n^2 / p} = p \qquad E_p = \frac{n^2}{p \cdot (n^2 / p)} = 1$$

Разработанный способ параллельных вычислений позволяет достичь идеальных показателей ускорения и эффективности

Параллельный алгоритм 3: метод Кэннона

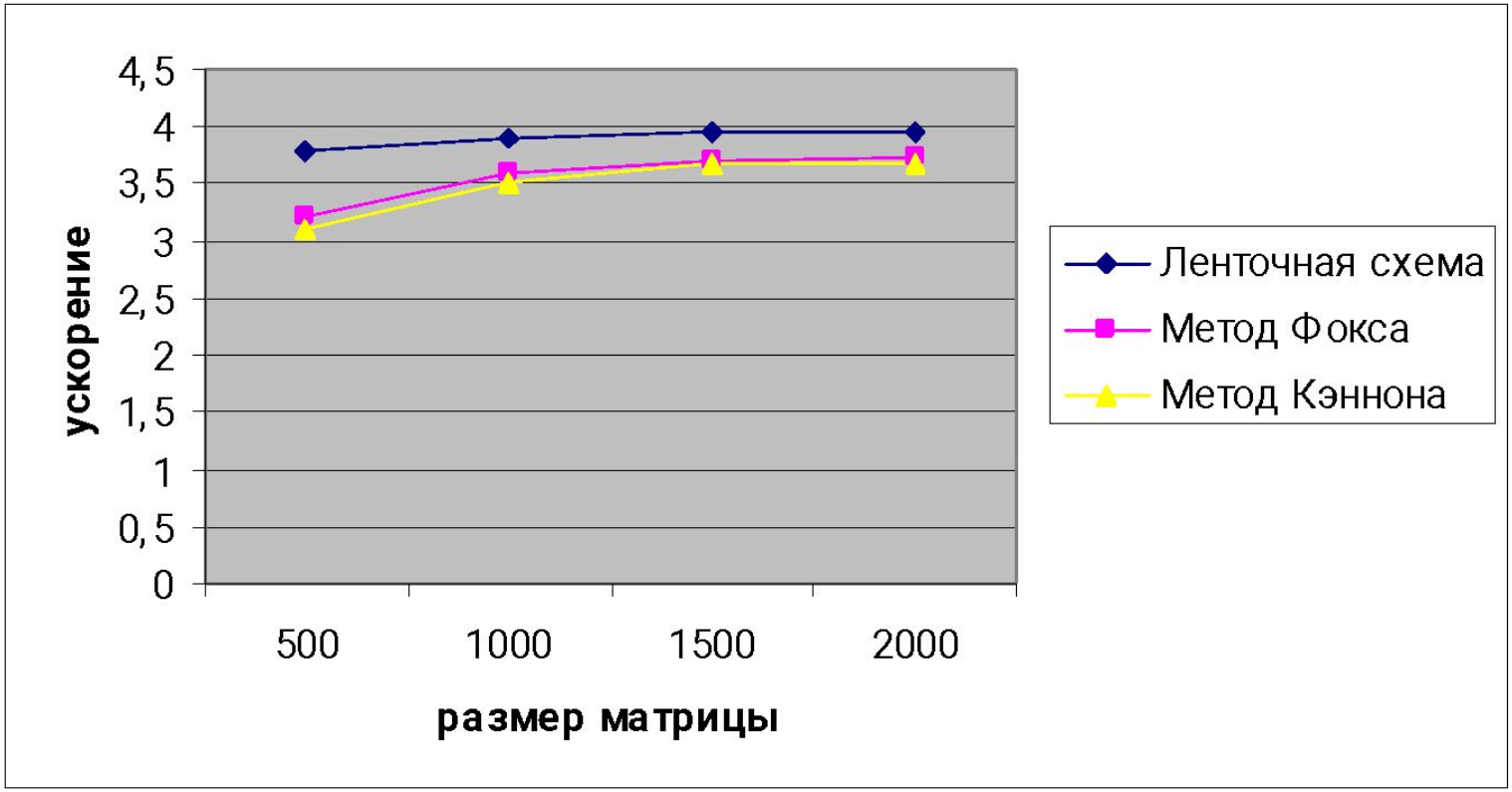
- **Результаты вычислительных экспериментов**
  - Ускорение вычислений

Размер матриц	Последовательный алгоритм	Параллельный алгоритм, 4 процессора	
		Время	Ускорение
500×500	2,0628	0,6676	3,0899
1000×1000	16,5152	4,7065	3,509
1500×1500	56,566	15,4247	3,6672
2000×2000	133,9128	36,5024	3,6686



Сравнение

- Показатели ускорения рассмотренных параллельных алгоритмов при умножении матриц по результатам вычислительных экспериментов для 4 процессоров



Методы параллельной сортировки

- Постановка задачи
- Принципы распараллеливания
- Пузырьковая сортировка
- Сортировка Шелла
- Параллельная быстрая сортировка
- Обобщенная быстрая сортировка
- Сортировка с использованием регулярного набора образцов
- Заключение

Постановка задачи

Сортировка является одной из типовых проблем обработки данных и обычно понимается как задача размещения элементов неупорядоченного набора значений

$$S = \{a_1, a_2, ..., a_n\}$$

в порядке монотонного возрастания или убывания

$$S \sim S' = \{(a_1', a_2', ..., a_n') : a_1' \le a_2' \le ... \le a_n'\}$$

Принципы распараллеливания…

Базовая операция – "*сравнить и переставить*" (*compare-exchange*)

```
// базовая операция сортировки
if ( A[i] > A[j] ) {
  temp = A[i];
  A[i] = A[j];
  A[j] = temp;
}
```

- Последовательное применение данной операции позволяет упорядочить данные
- В способах выбора пар значений для сравнения проявляется различие алгоритмов сортировки

Принципы распараллеливания…

Параллельное обобщение базовой операции при ***p = n***
(каждый процессор содержит 1 элемент данных):

- выполнить взаимообмен имеющихся на процессорах P_i и P_j значений (с сохранением на этих процессорах исходных элементов),
- сравнить на каждом процессоре P_i и P_j получившиеся одинаковые пары значений (a_i, a_j) и по результатам сравнения разделить данные между процессорами – на одном процессоре (например, P_i) оставить меньший элемент, на другом процессоре (т.е. P_j) запомнить большее значение пары

$$a_i' = \min(a_i, a_j) \qquad a_j' = \max(a_i, a_j)$$

Принципы распараллеливания…

Результат выполнения **параллельного алгоритма:**

- имеющиеся на процессорах данные упорядочены,
- порядок распределения данных по процессорам соответствует линейному порядку нумерации (т.е. значение элемента на процессоре P_i меньше или равно значения элемента на процессоре P_{i+1}, где $0 \le i < p-1$).

Принципы распараллеливания…

Параллельное обобщение базовой операции при ***p < n***

(каждый процессор содержит блок данных размера n / p):

- упорядочить блок на каждом процессоре в начале сортировки,
- выполнить взаимообмен блоков между процессорами P_i и P_{i+1},
- объединить блоки A_i и A_{i+1} на каждом процессоре в один отсортированный блок с помощью операции слияния,
- разделить полученный двойной блок на две равные части и оставить одну из этих частей (например, с меньшими значениями данных) на процессоре P_i, а другую часть (с большими значениями соответственно) – на процессоре P_{i+1}

$$[A_i \cup A_{i+1}]_{copm} = A_i \cup A_{i+1} : \forall a_i \in A_i, \forall a_j \in A_{i+1} \Rightarrow a_i \le a_j$$

Рассмотренная процедура обычно именуется в литературе как операция "*сравнить и разделить*" (*compare-split*).

Пузырьковая сортировка: *последовательный алгоритм*

```
// Последовательный алгоритм пузырьковой сортировки
BubbleSort(double A[], int n) {
  for (i=0; i<n-1; i++)
    for (j=0; j<n-i; j++)
      compare_exchange(A[j], A[j+1]);
}
```

- Трудоемкость вычислений имеет порядок $O(n^2)$
- В прямом виде сложен для распараллеливания

Пузырьковая сортировка: *алгоритм чет-нечетной перестановки…*

Вводятся два разных правила выполнения итераций метода:

- на всех нечетных итерациях сравниваются пары

  $(a_1, a_2), (a_3, a_4), \ldots, (a_{n-1}, a_n)$ (при четном n),

- на четных итерациях обрабатываются элементы

  $(a_2, a_3), (a_4, a_5), \ldots, (a_{n-2}, a_{n-1})$.

Пузырьковая сортировка: *алгоритм чет-нечетной перестановки///*

```
// Последовательный алгоритм чет-нечетной перестановки
OddEvenSort(double A[], int n) {
  for (i=1; i<n; i++) {
    if ( i%2==1) // нечетная итерация
      for (j=0; j<n/2-1; j++)
        compare_exchange(A[2j+1],A[2j+2]);
    if (i%2==0) // четная итерация
      for (j=1; j<n/2-1; j++)
        compare_exchange(A[2j],A[2j+1]);
  }
}
```

Разные правила для выполнения четных и нечетных итераций

⇩

Возможности распараллеливания

Пузырьковая сортировка: *параллельный алгоритм чет-нечетной перестановки*…

```
// Параллельный алгоритм чет-нечетной перестановки
ParallelOddEvenSort ( double A[], int n ) {
  int id = GetProcId();  // номер процесса
  int np = GetProcNum(); // количество процессов
  for ( int i=0; i<np; i++ ) {
    if ( i%2 == 1 ) { // нечетная итерация
      if ( id%2 == 1 ) // нечетный номер процесса
        compare_split_min(id+1); // сравнение-обмен справа
      else compare_split_max(id-1); // сравнение-обмен слева
    }
    if ( i%2 == 0 ) { // четная итерация
      if( id%2 == 0 )  // четный номер процесса
        compare_split_min(id+1); // сравнение-обмен справа
      else compare_split_max(id-1); // сравнение-обмен слева
    }
  }
}
```

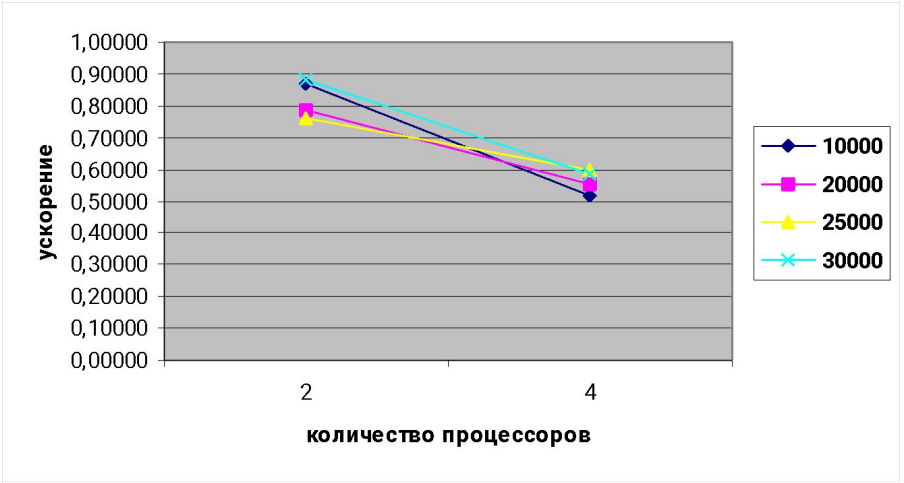
Пузырьковая сортировка: *параллельный алгоритм чет-нечетной перестановки…*

№ и тип итерации	Процессоры			
	1	2	3	4
Исходные данные	13 55 59 88	29 43 71 85	2 18 40 75	4 14 22 43
1 нечет (1,2),(3.4)	13 55 59 88	29 43 71 85	2 18 40 75	4 14 22 43
	13 29 43 55	59 71 85 88	2 4 14 18	22 40 43 75
2 чет (2,3)	13 29 43 55	59 71 85 88	2 4 14 18	22 40 43 75
	13 29 43 55	2 4 14 18	59 71 85 88	22 40 43 75
3 нечет (1,2),(3.4)	13 29 43 55	2 4 14 18	59 71 85 88	22 40 43 75
	2 4 13 14	18 29 43 55	22 40 43 59	71 75 85 88
4 чет (2,3)	2 4 13 14	18 29 43 55	22 40 43 59	71 75 85 88
	2 4 13 14	18 22 29 40	43 43 55 59	71 75 85 88

Пузырьковая сортировка: *параллельный алгоритм чет-нечетной перестановки…*

- **Результаты вычислительных экспериментов:**
  - Ускорение вычислений

Количество элементов	Последовательный алгоритм	Параллельный алгоритм			
		2 процессора		4 процессора	
		Время	Ускорение	Время	Ускорение
10000	0,003	0,003	0,868	0,005	0,516
20000	0,006	0,007	0,786	0,010	0,551
25000	0,007	0,010	0,764	0,012	0,600
30000	0,009	0,012	0,883	0,014	0,588



Пузырьковая сортировка: *параллельный алгоритм чет-нечетной перестановки*

- Параллельный вариант алгоритма *работает медленнее* исходного последовательного метода пузырьковой сортировки:
  - объем передаваемых данных между процессорами является достаточно большим и сопоставим с количеством выполняемых вычислительных операций,
  - этот дисбаланс объема вычислений и сложности операций передачи данных увеличивается с ростом числа процессоров

Быстрая сортировка: *последовательный алгоритм…*

- *Алгоритм быстрой сортировки*, предложенной Хоаром (*Hoare C.A.R.*), основывается на последовательном разделении сортируемого набора данных на блоки меньшего размера таким образом, что между значениями разных блоков обеспечивается отношение упорядоченности (для любой пары блоков все значения одного из этих блоков не превышают значений другого блока):
  - На первой итерации метода осуществляется деление исходного набора данных на первые две части – для организации такого деления выбирается некоторый *ведущий элемент* и все значения набора, меньшие ведущего элемента, переносятся в первый формируемый блок, все остальные значения образуют второй блок набора,
  - На второй итерации сортировки описанные правила применяются рекурсивно для обоих сформированных блоков и т.д.
- При оптимальном выборе ведущих элементов после выполнения $log_2 n$ итераций исходный массив данных оказывается упорядоченным.

Быстрая сортировка: *последовательный алгоритм…*

```
// Последовательный алгоритм быстрой сортировки
QuickSort(double A[], int i1, int i2) {
  if ( i1 < i2 ){
    double pivot = A[i1];
    int is = i1;
    for ( int i = i1+1; i<i2; i++ )
      if ( A[i] ≤ pivot ) {
        is = is + 1;
        swap(A[is], A[i]);
      }
    swap(A[i1], A[is]);
    QuickSort (A, i1, is);
    QuickSort (A, is+1, i2);
  }
}
```

Среднее количество операций $1.4n\log_2 n$

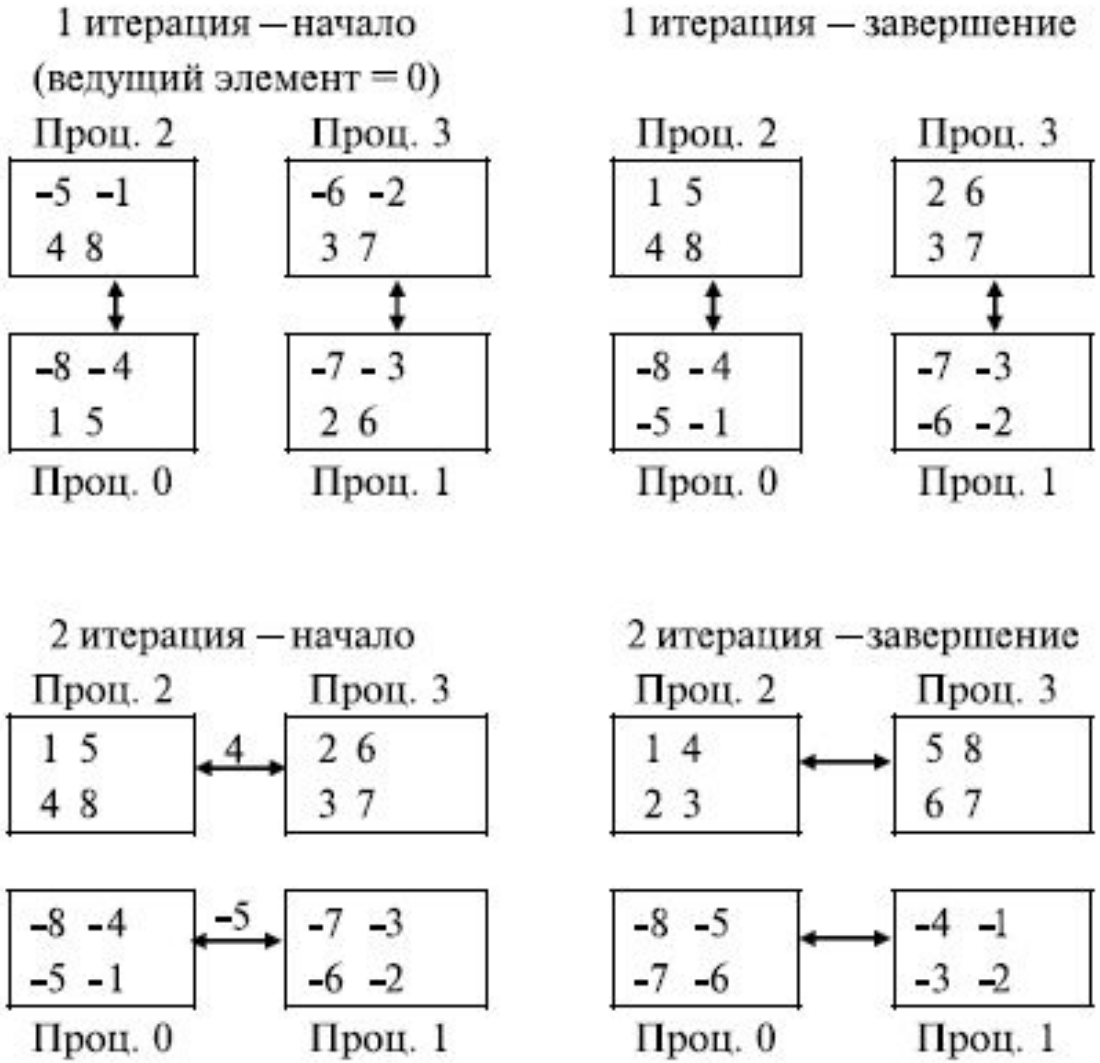
Быстрая сортировка: *параллельный алгоритм…*

Пусть топология коммуникационной сети имеет вид ***N***-мерного гиперкуба (т.е. количество процессоров равно $p=2^N$).

Действия алгоритма состоят в следующем:

- выбрать каким-либо образом ведущий элемент и разослать его по всем процессорам системы (например, в качестве ведущего элемента можно взять среднее арифметическое элементов, расположенных на ведущем процессоре),
- разделить на каждом процессоре имеющийся блок данных на две части с использованием полученного ведущего элемента,
- образовать пары процессоров, для которых битовое представление номеров отличается только в позиции ***N***, и осуществить взаимообмен данными между этими процессорами; в результате таких пересылок данных на процессорах, для которых в битовом представлении номера бит позиции ***N*** равен 0, должны оказаться части блоков со значениями, меньшими ведущего элемента; процессоры с номерами, в которых бит ***N*** равен 1, должны собрать, соответственно, все значения данных, превышающие значения ведущего элемента,
- перейти к подгиперкубам меньшей размерности и повторить описанную выше процедуру

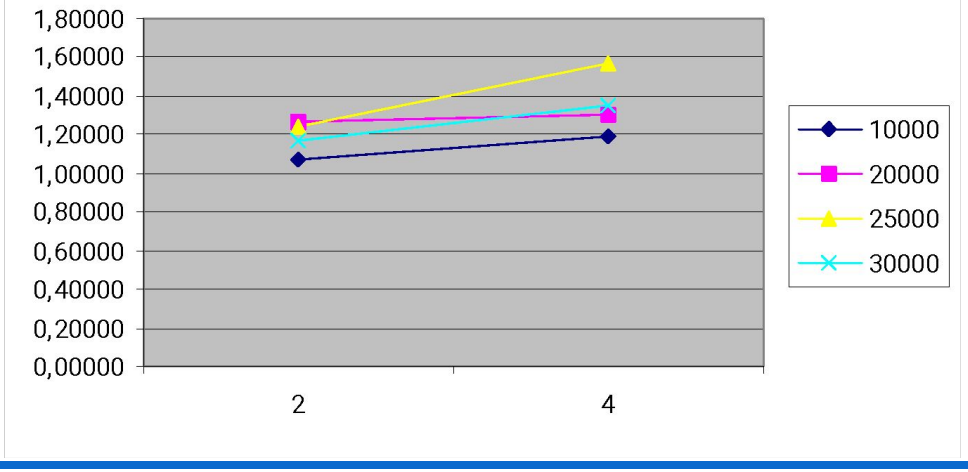
Быстрая сортировка: *параллельный алгоритм…*



Быстрая сортировка: *параллельный алгоритм*

- **Результаты вычислительных экспериментов:**
  - Ускорение вычислений

Размер данных	Последовательный алгоритм	Параллельный алгоритм			
		2 процессора		4 процессора	
		время	ускорение	время	ускорение
10 000	0,003	0,002	1,074	0,002	1,196
20 000	0,006	0,005	1,262	0,004	1,305
25 000	0,007	0,006	1,240	0,005	1,567
30 000	0,009	0,008	1,164	0,007	1,353



Обобщенная быстрая сортировка: *параллельный алгоритм…*

- Основное отличие от предыдущего алгоритма – конкретный способ выбора ведущего элемента
- Сортировка блоков выполняется в самом начале выполнения вычислений. В качестве ведущего выбирается средний элемент какого-либо блока (например, на первом процессоре вычислительной системы). Выбираемый подобным образом ведущий элемент в отдельных случаях может оказаться более близок к настоящему среднему значению всего сортируемого набора, чем какое-либо другое произвольно выбранное значение
- Для поддержки упорядоченности в ходе вычислений процессоры выполняют операцию слияния частей блоков, получаемых после разделения

Сортировка с использованием регулярного набора образцов: *параллельный алгоритм…*

Первый этап**:* упорядочивание блоков каждым процессором независимо друг от друга при помощи обычного алгоритма быстрой сортировки; далее каждый процессор формирует набор из элементов своих блоков с индексами ***0, m, 2m,…,(p-1)m, где ***$m=n/p^2$***,

Второй этап: все сформированные на процессорах наборы данных собираются на одном из процессоров системы и объединяются в ходе последовательного сливания в одно упорядоченное множество; из полученного множества значений из элементов с индексами

$$p+\lfloor p/2\rfloor-1,\quad 2p+\lfloor p/2\rfloor-1,\ldots,(p-1)p+\lfloor p/2\rfloor$$

формируется набор ведущих элементов, который передается всем процессорам; в завершение этапа каждый процессор выполняет разделение своего блока на ***p*** частей с использованием полученного набора ведущих значений,

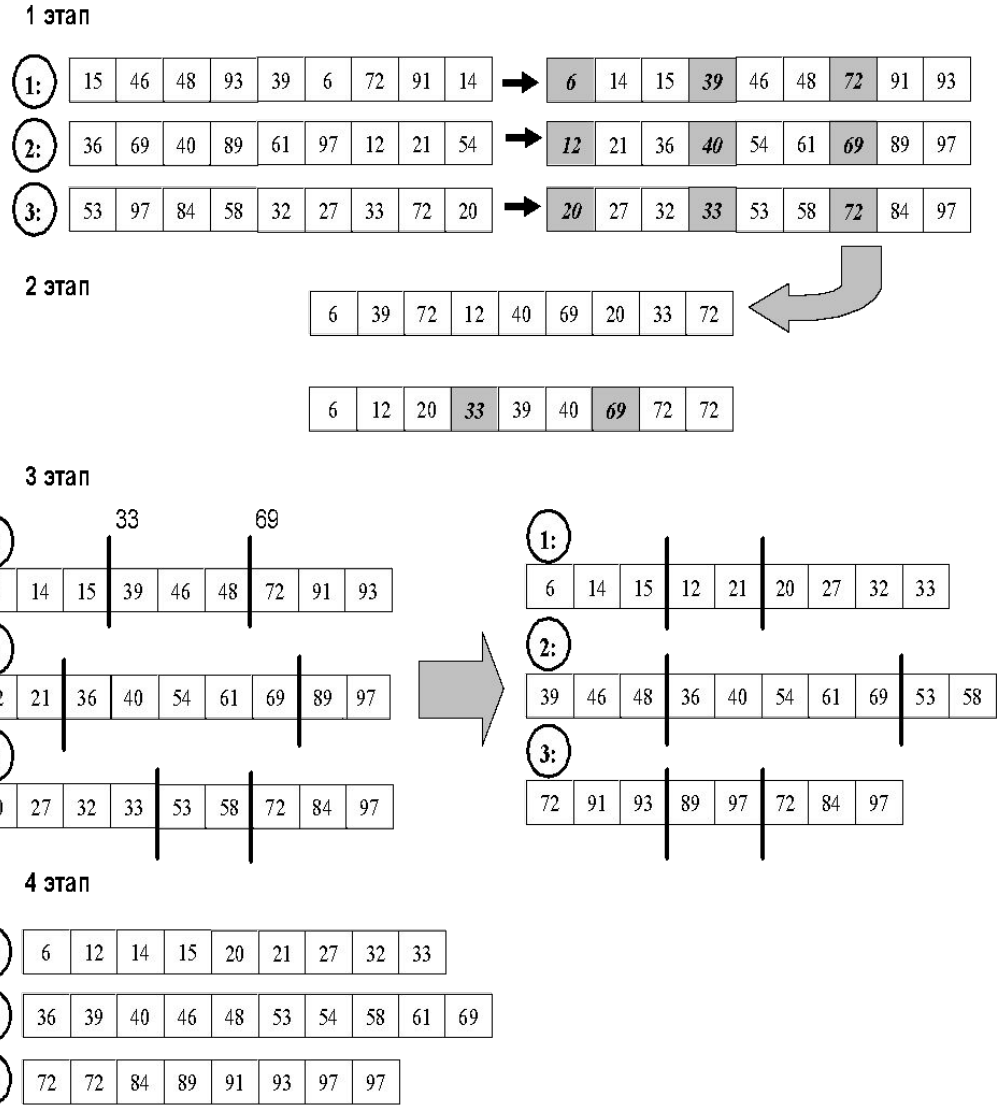
Третий этап: каждый процессор рассылает выделенные ранее части своего блока всем остальным процессорам системы в соответствии с порядком нумерации - часть ***j, 0≤ j<p***, каждого блока пересылается процессору с номером ***j***,

Четвертый этап: каждый процессор выполняет слияние ***p*** полученных частей в один отсортированный блок.

Сортировка с использованием регулярного набора образцов: *параллельный алгоритм…*

Пример:

(p=3)



Сортировка с использованием регулярного набора образцов: *параллельный алгоритм*

- **Результаты вычислительных экспериментов:**
  - Ускорение вычислений

Размер данных	Последовательный алгоритм	Параллельный алгоритм			
		2 процессора		4 процессора	
		время	ускорение	время	ускорение
10 000	0,003	0,003	1,045	0,002	1,231
20 000	0,006	0,005	1,212	0,003	1,751
25 000	0,007	0,006	1,269	0,004	1,920
30 000	0,009	0,007	1,291	0,004	2,098