Лекция 6

Параллельные алгоритмы умножения матриц и векторов

Часть 1. Умножение матрицы на вектор

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***

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

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

Способы распределения данных

Способы распределения данных из матрицы

горизонтальные полосы

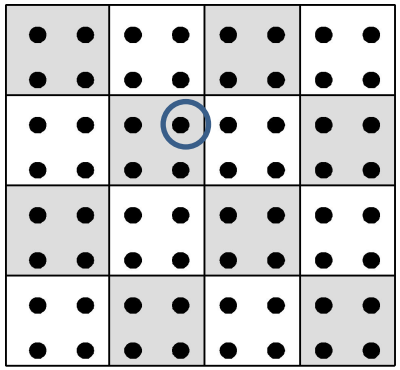
вертикальные полосы

Чередующееся (цикличное) горизонтальное разбиение

блочное разбиение

Блочная схема

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



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

$m = 8, s - 4, k = 8/4{=}2$

Тогда $a_{1_0 1_1} = a_{23}$

$1_0 = 1 * 2 + 0 = 2; 1_1 = 1 * 2 + 1 = 3$

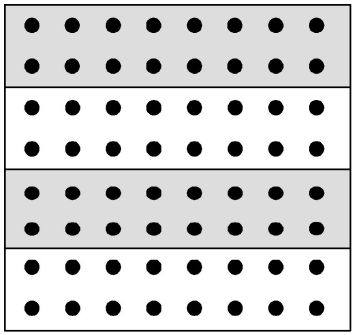
где: m – число строк матрицы A, n – размер вектора B, s – число процессоров, k – число строк в блоке, v – номер строки внутри блока, i – номер блока

Способы распределения данных: *ленточная схема*

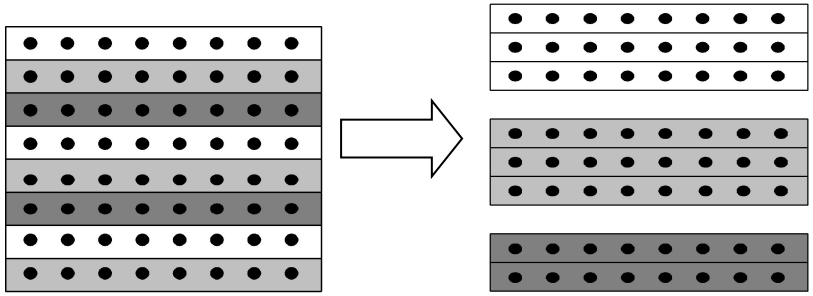
Непрерывное (последовательное) распределение

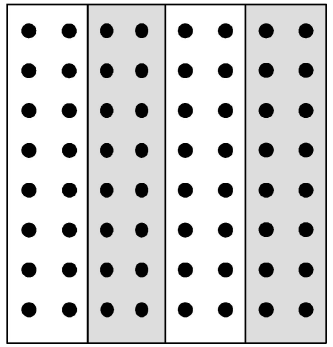
горизонтальные полосы

вертикальные полосы



Чередующееся (цикличное) горизонтальное разбиение





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

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

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

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

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

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

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

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

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

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

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

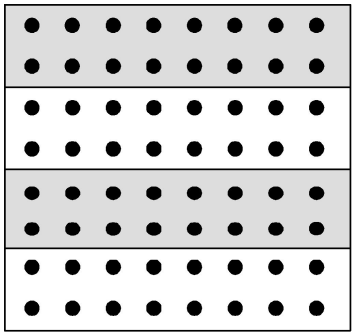
Последовательный алгоритм

```
// 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)*.

Базовая подзадача - минимальная задача, выполняемая всеми процессорами

Алгоритм 1: ленточная схема (разбиение матрицы по строкам)



$$A = (A_0, A_1, ..., A_{p-1})^T,$$
$$A_i = (a_{i_0}, a_{i_1}, ..., a_{i_{k-1}}),$$
$$i_j = ik + j, 0 \le j < k, k = m / p$$
$$(a_i, 0 \le i < m, - \text{строки матрицы } A)$$

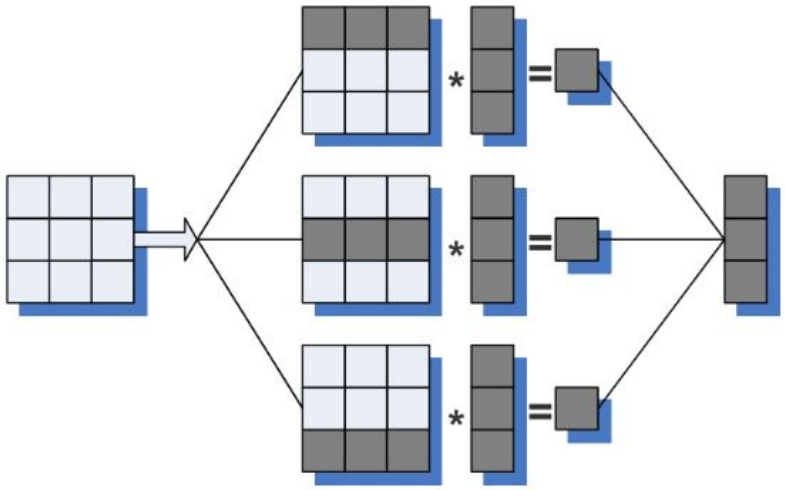
Базовая подзадача - операция скалярного умножения одной строки матрицы на вектор:

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

– Базовая подзадача для выполнения вычисления должна содержать:

*строку матрицы **A** и копию вектора **b**.*

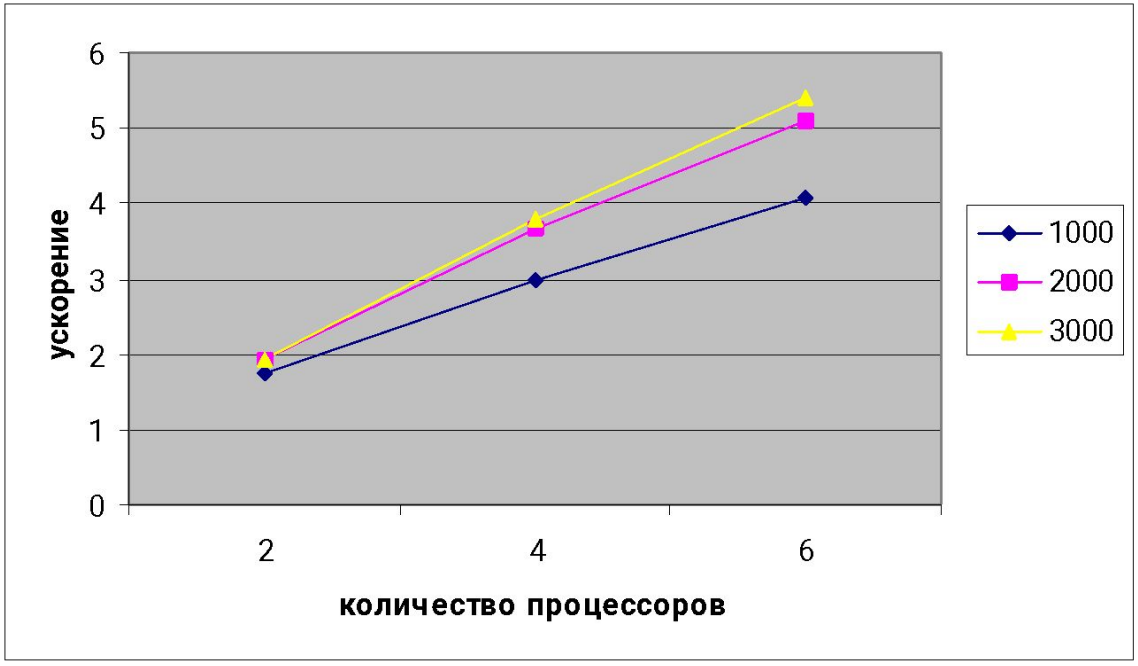
После завершения вычислений каждая базовая подзадача будет содержать один из элементов вектора результата ***c***



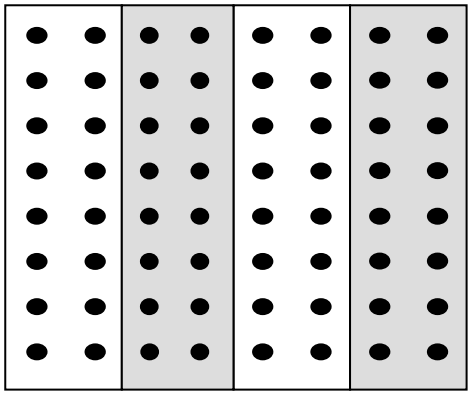
– Для объединения результатов расчетов и получения полного вектора ***c*** на каждом из процессоров вычислительной системы необходимо выполнить операцию обобщенного сбора данных

Результаты вычислительных экспериментов

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)



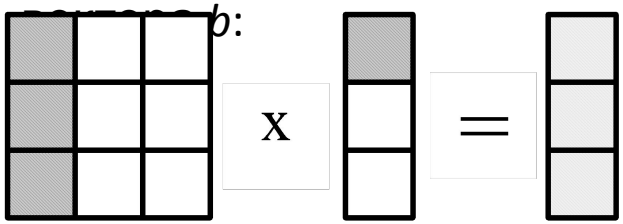
$$A = (A_0, A_1, ..., A_{p-1}),$$

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

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

$$(\alpha_i, 0 \le i < m, - \ столбцы \ матрицы \ A)$$

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



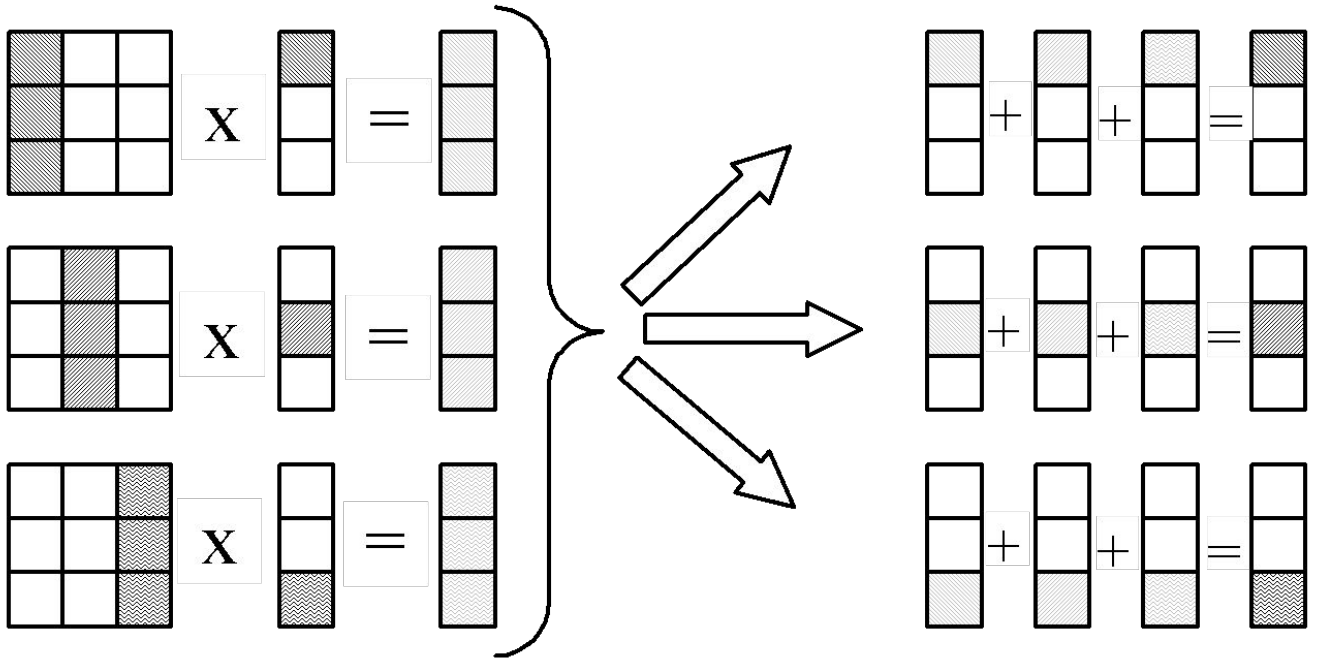
- **Базовая подзадача** для выполнения вычисления должна содержать

  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

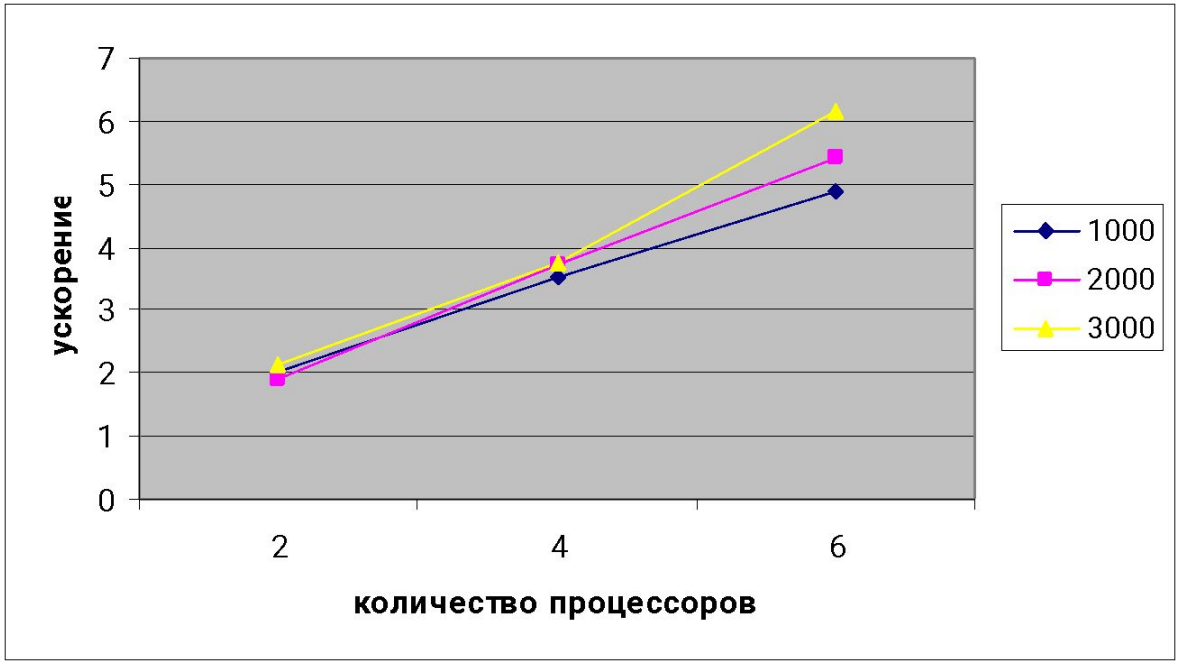
Схема информационного взаимодействия



- Для получения элементов результирующего вектора ***c*** подзадачи должны обменяться своими промежуточными данными

Результаты вычислительных экспериментов

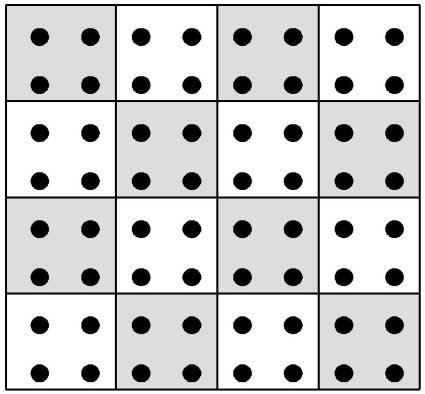
Размер матрицы	Последовательный алгоритм	2 процессора		4 процессора		6 процессоров	
		Время	Ускорение	Время	Ускорение	Время	Ускорение
1000x1000	0,0291	0,0144	2,0225	0,0083	3,5185	0,00595	4,8734
2000x2000	0,1152	0,0610	1,8869	0,0311	3,7077	0,0213	5,4135
3000x3000	0,2565	0,1201	2,1364	0,0683	3,7528	0,04186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



Алгоритм 3: блочная схема

Пусть:

- количество процессоров $p=s \cdot q$,
- количество строк матрицы является кратным s: $m=k \cdot s$
- количество столбцов является кратным q: $l=n \cdot q$.



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

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

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

✔ Подзадачи нумеруются индексами *(i, j)* располагаемых в подзадачах матричных блоков

✔ Подзадачи выполняют умножение содержащегося в них блока матрицы ***A*** на блок вектора ***b***

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

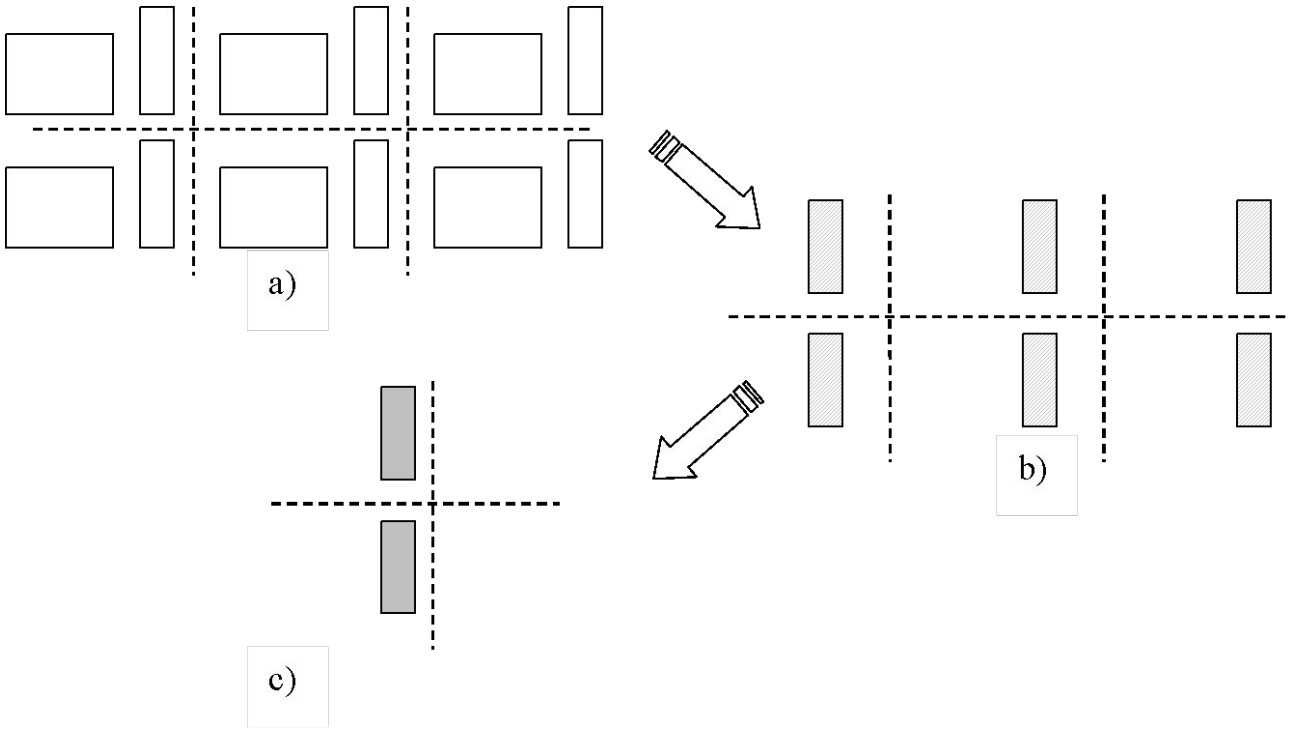
✔ После перемножения блоков матрицы ***A*** и вектора ***b*** каждая подзадача *(i,j)* будет содержать вектор частичных результатов *c'(i,j)*:

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

Алгоритм 3: блочная схема

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

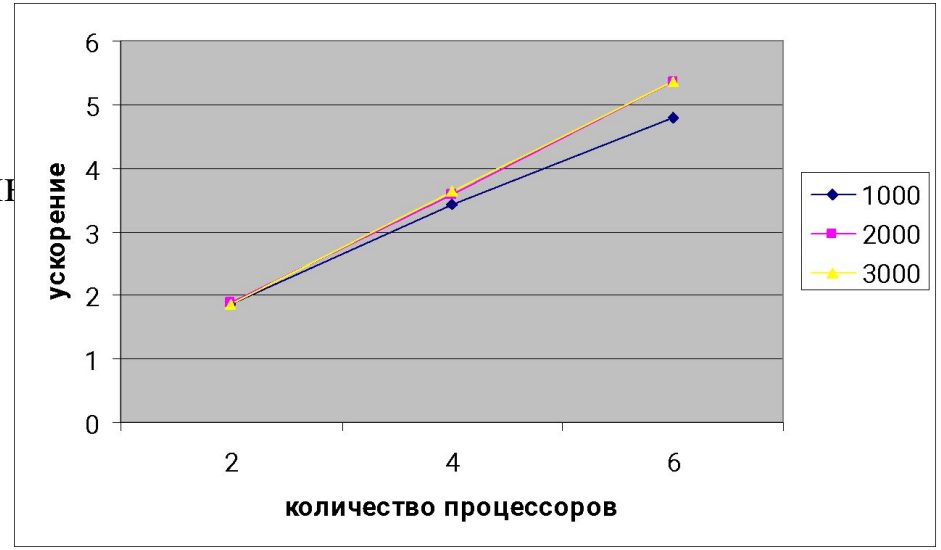
$$c_\eta = \sum_{j=0}^{q-1} c'_\nu (i, j), 0 \le \eta < m, i = \eta / s, \nu = \eta - i \cdot s$$



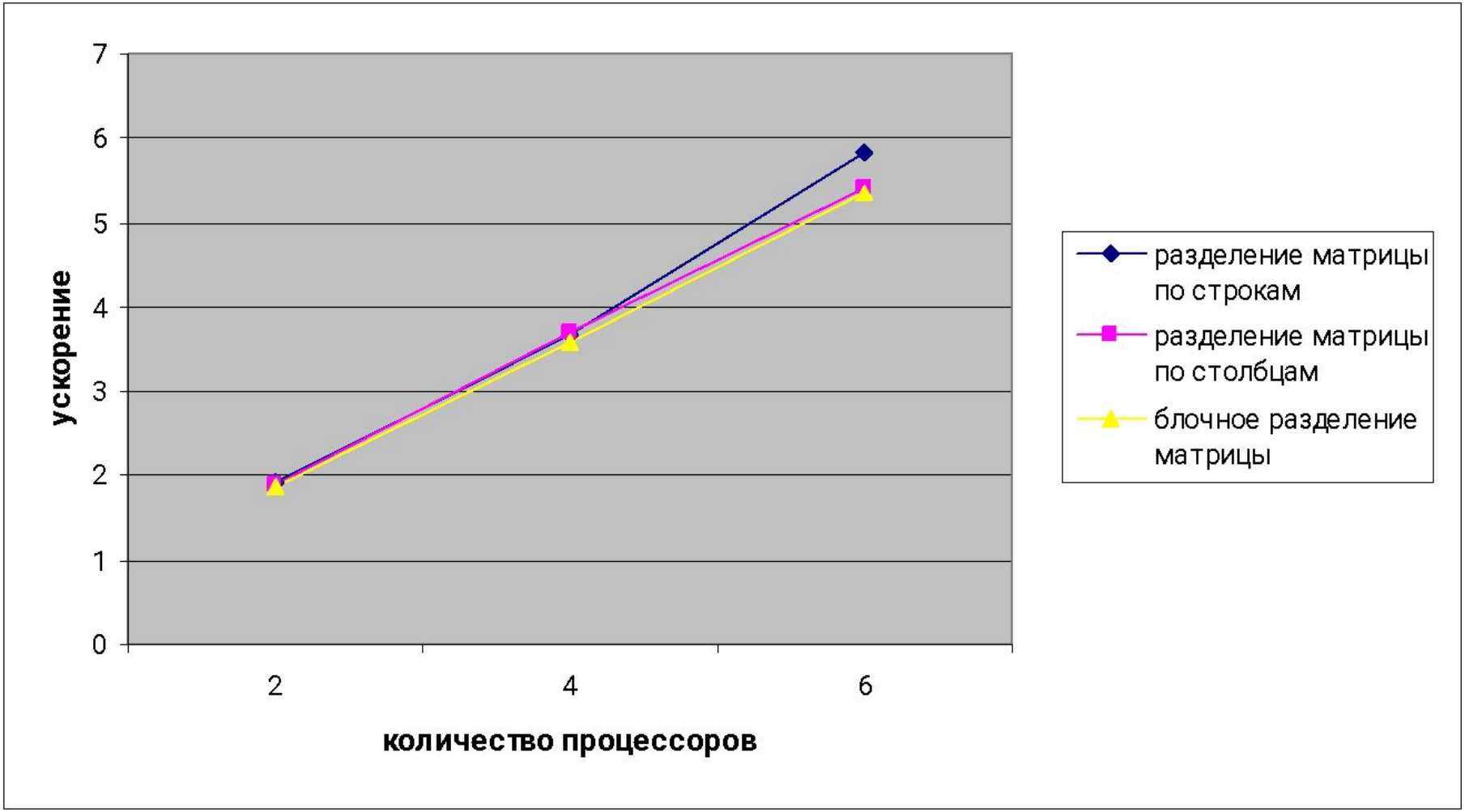
Результаты вычислительных экспериментов

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

- ✔ Общее количество базовых подзадач совпадает 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.



Сравнение алгоритмов



Часть 2. Умножение матриц

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

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

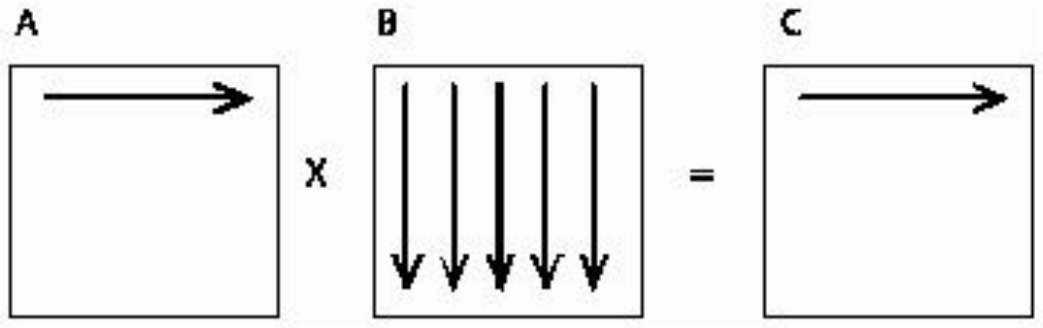
В основу организации параллельных вычислений может быть положен ***принцип распараллеливания по данным***

Последовательный базовый алгоритм

```
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];
}
```

Один проход по внутреннему циклу *j* – один элемент матрицы C

Один проход по внешнему циклу *i* – одна строка матрицы C



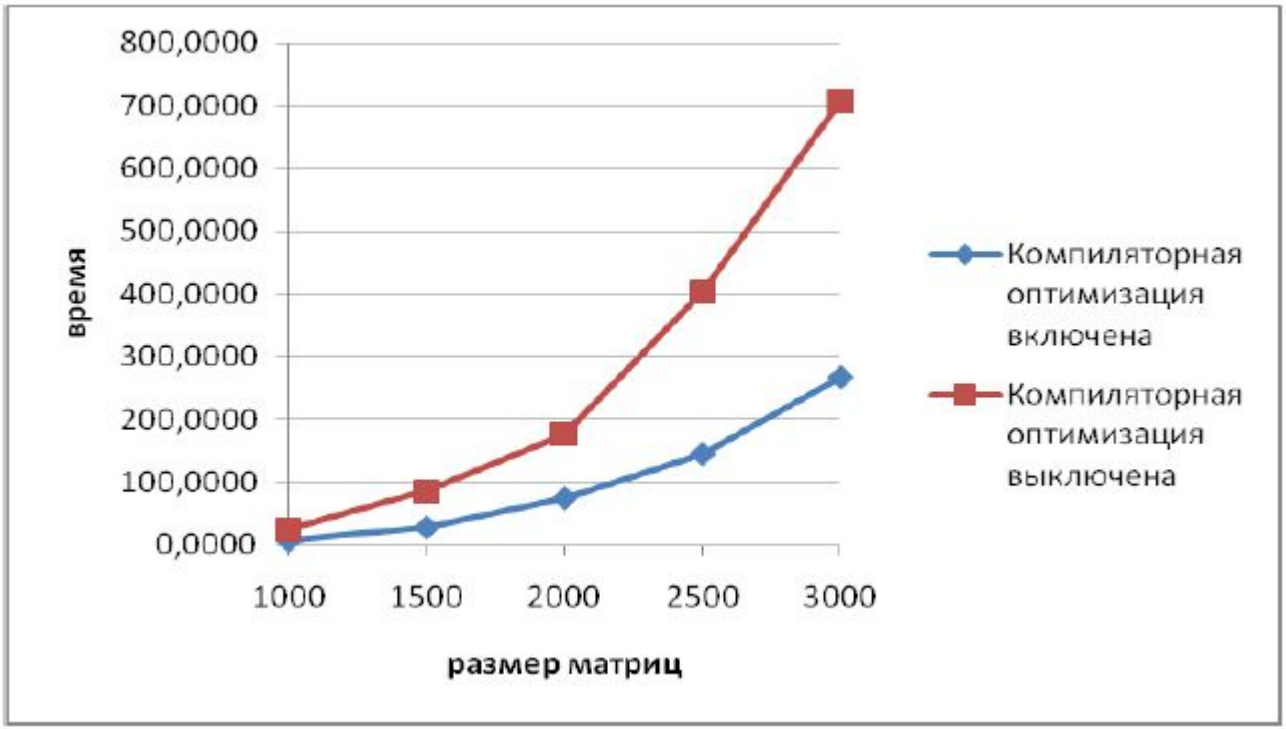
Результаты вычислительных экспериментов

Сравнение времени выполнения оптимизированной и неоптимизированной версий последовательного алгоритма умножения матриц

Размер матриц	Компиляторная оптимизация включена	Компиляторная оптимизация выключена
1000,0000	8,2219	24,8192
1500,0000	28,6027	85,8869
2000,0000	75,1572	176,5772
2500,0000	145,2053	403,2405
3000,0000	267,0592	707,1501

Эксперименты проводились на двухпроцессорном вычислитель-ном узле на базе четырех-ядерных процессоров Intel Xeon E5320, 1.86 ГГц, 4 Гб RAM под управлением операционной системы Micro-soft Windows HPC Server 2008. Разработка программ проводилась в среде Microsoft Visual Studio 2008, для компиляции использовался Intel C++ Compiler 10.0 for Windows.

Графики зависимости времени выполнения оптимизированной и неоптимизированной версий последовательного алгоритма



Часть 2. Умножение матриц

Умножение матрицы A размера $m \times n$ и матрицы B размера $n \times l$:

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}, 0 \le i < m, 0 \le j < l$$

Произведение матриц можно рассматривать как:

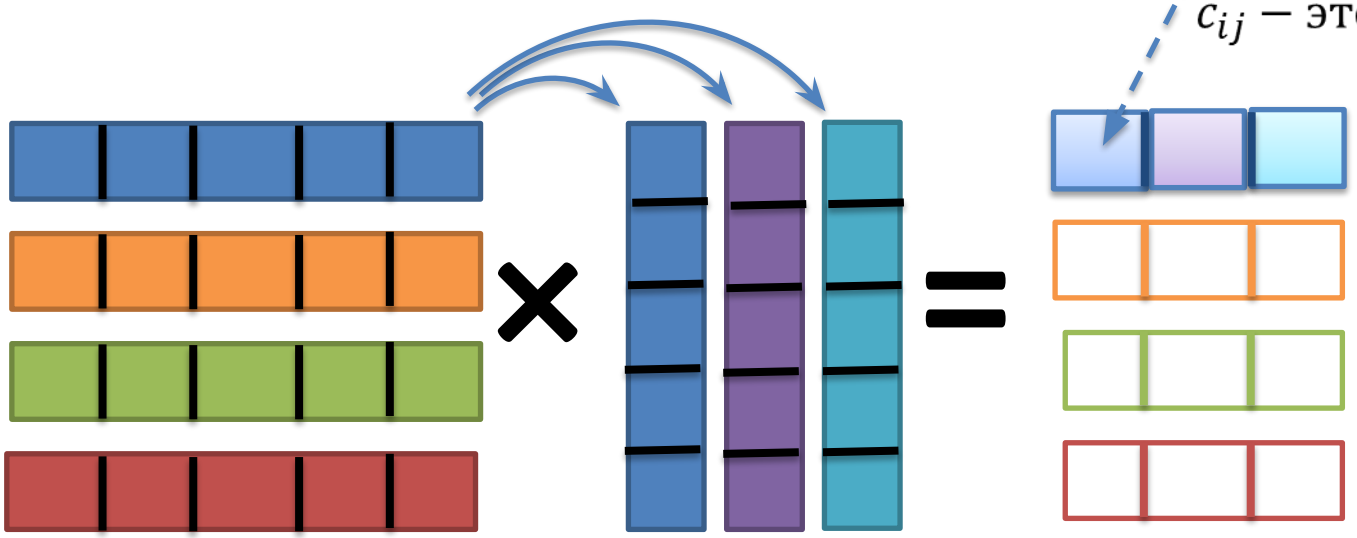
1. Как $m \times l$ скалярных произведений векторов: m строк матрицы A на l столбцов матрицы B:

$A = (A_0, A_1, \dots, A_m), A_i = (a_{i1}, a_{i2}, \dots a_{in}), i = 1 \dots m$

$B = \left({B_0}^T, {B_1}^T, \dots, {B_l}^T\right), {B_j}^T = \left(b_{1j}, b_{2j}, \dots b_{nj}\right), j = 1 \dots l$

$c_{ij} = (A_i, B_j)$

c_{ij} – это 1 элемент



Умножение матриц

Произведение матриц можно рассматривать как:

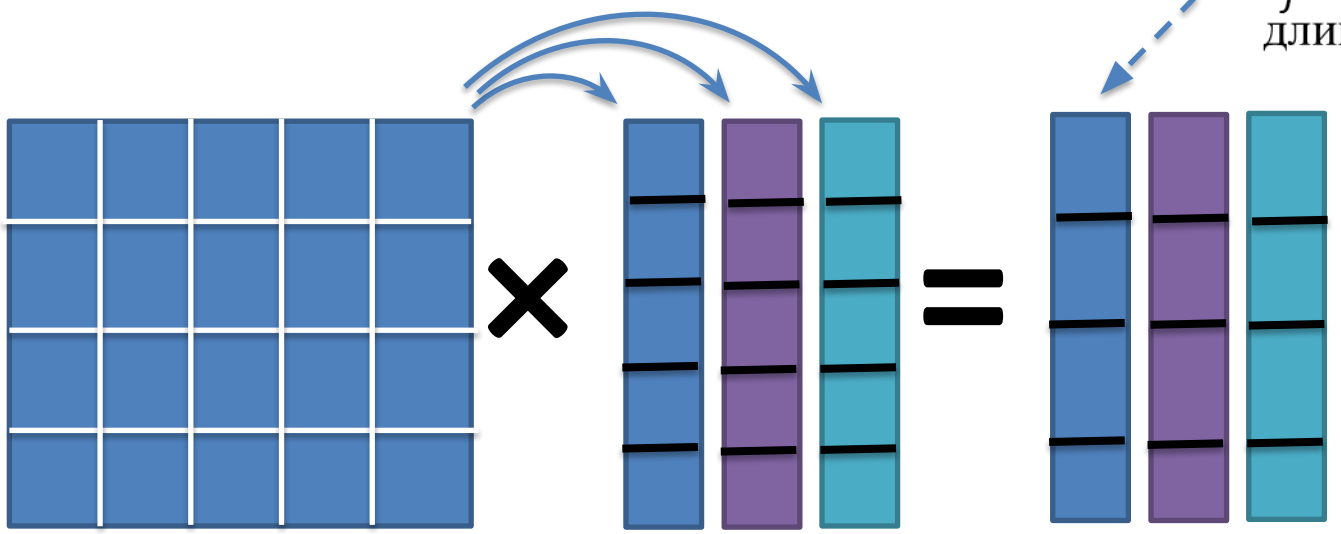
2. Как l произведений матрицы A на векторы B_j:

$$B = \left({B_0}^T, {B_1}^T, \dots, {B_l}^T\right),\ {B_j}^T = \left(b_{1j}, b_{2j}, \dots b_{nj}\right),\ j = 1 \dots l$$

$$C = \left({C_0}^T, {C_1}^T, \dots, {C_l}^T\right),\ {C_j}^T = \left(c_{1j}, c_{2j}, \dots c_{nj}\right),\ j = 1 \dots l$$

$$C_j = \left(A, B_j\right)$$

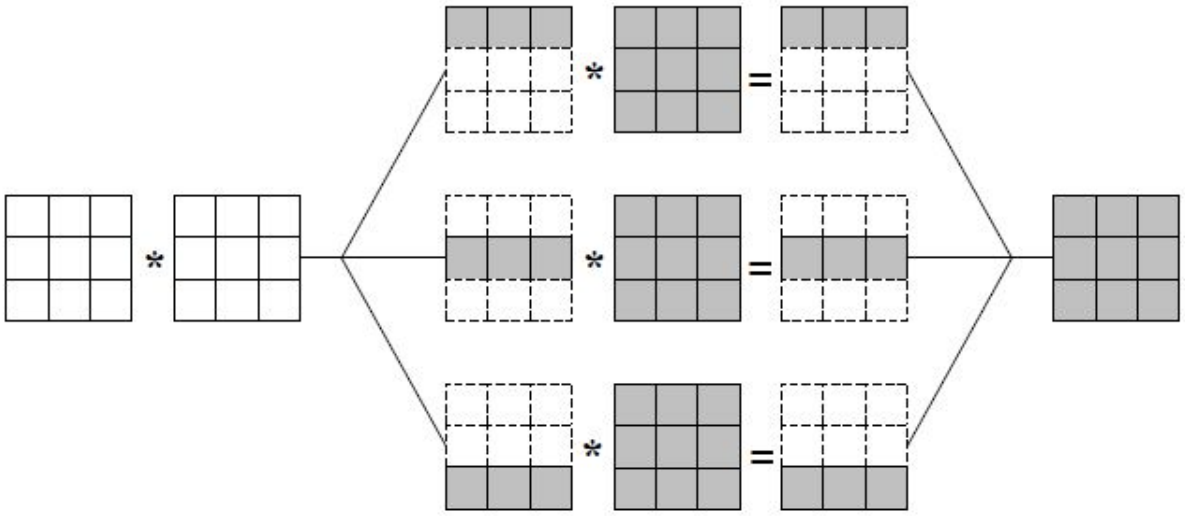
c_j – это 1 вектор длиною m



Умножение матриц

Произведение матриц можно рассматривать как:

3. Как m произведений строк матрицы A_i на матрицу B:
   $A = \left(A_0{}^T, A_1{}^T, \dots, A_m{}^T\right), A_j{}^T = \left(a_{1j}, a_{2j}, \dots a_{mj}\right), j = 1 \dots n$



Процесс хранит одну строку матрицы A и все столбцы матрицы B

Трудоемкость

Размер A $- \ m \times n$
Размер B $- n \times l$

Для скалярных произведений векторов: $c_{ij} = (A_i, B_j)$

- длина векторов A_i и $B_j = n$, поэтому для нахождения одного элемента c_{ij} необхоимо n произведений и $n - 1$ сложений. Всего $(n + n - 1)$ операций.
- число векторов $A_i - m$, число векторов $B_j - l$. Потому всего операций будет $m\ l(n + n - 1) = ml(2n - 1)$.
- Трудоемкость: $O(mln)$
- Для случая квадратных матриц размера n: $O(n^3)$ и

$$T = n^2(2n - 1)\tau \approx (t_{mult} + t_{add})n^3$$

Для произведений матрицы на вектор: $C_j = (A, B_j)$

- Размер матрицы A $- mn$, поэтому для нахождения одного вектора c_j необходимо $m(n + n - 1)$ операций.
- число векторов $C_i - l$. Потому всего операций будет $m\ l(n + n - 1) = ml(2n - 1)$.
- Трудоемкость: $O(mln)$
- Для случая квадратных матриц размера n: $O(n^3)$ и

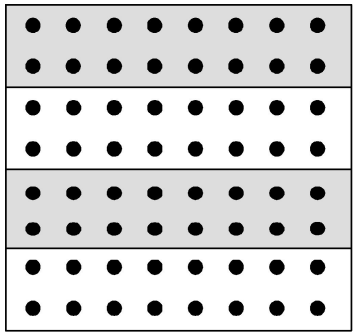
$$T = n^2(2n - 1)\tau \approx (t_{mult} + t_{add})n^3$$

$$S_p \approx \frac{(t_{mult} + t_{add}) \cdot n^3}{(t_{mult} + t_{add}) \cdot n^3 / p} = p$$

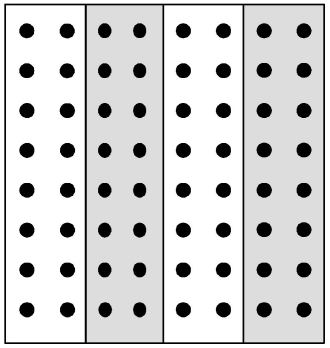
Способы распределения данных

Способы распределения данных матрицы

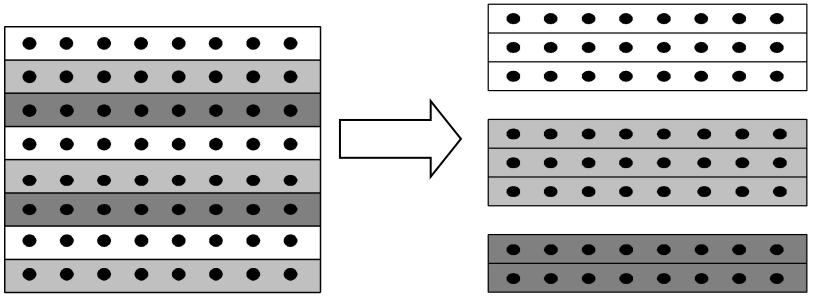
горизонтальные полосы



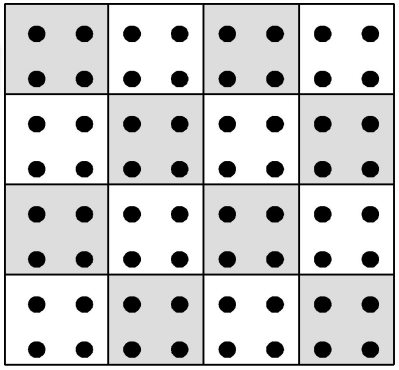
вертикальные полосы



Чередующееся (цикличное) горизонтальное разбиение



блочное разбиение



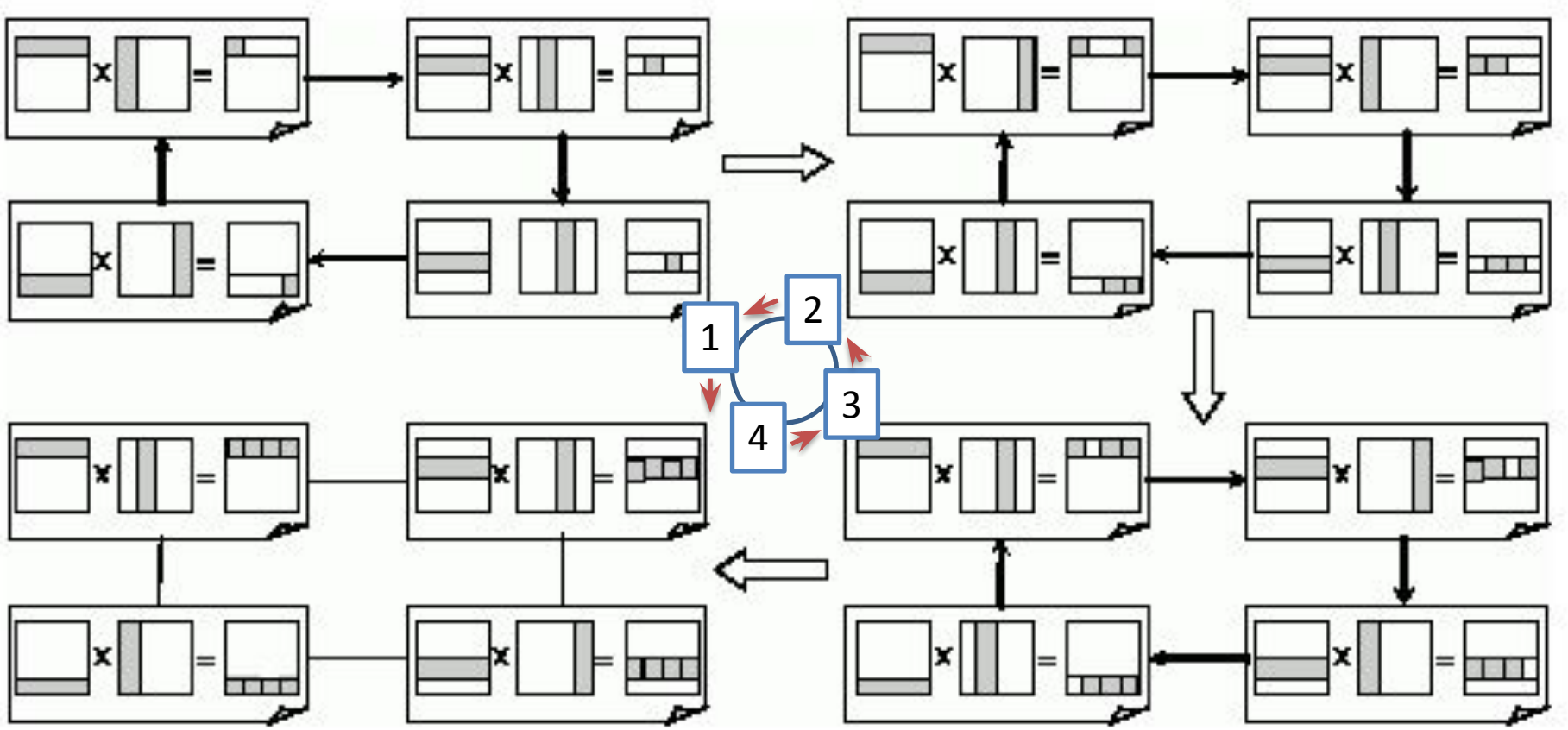
Параллельный алгоритм 1: ленточная схема

- ✔ 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 $(i+1) \bmod n$. Т.е. по завершении вычислений в конце каждой итерации столбцы матрицы B должны быть переданы между подзадачами с тем, чтобы в каждой подзадаче оказались новые столбцы матрицы B и могли быть вычислены новые элементы матрицы C.
- ✔ После выполнения всех итераций алгоритма в каждой подзадаче поочередно окажутся все столбцы матрицы ***B***.
- ✔ По завершении итераций строки собираются в единую матрицу C.

Примечание. Для распределения подзадач между процессорами может быть использован любой способ, обеспечивающий эффективное представление кольцевой структуры информационного взаимодействия подзадач.

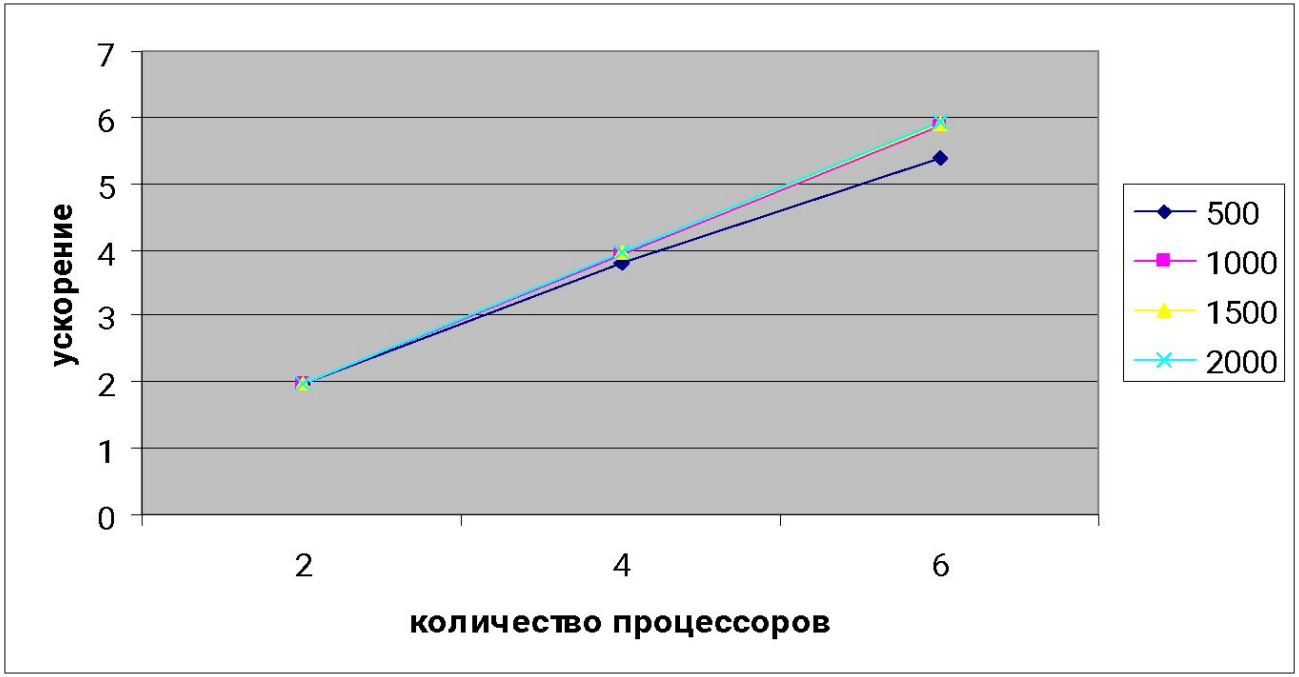
Параллельный алгоритм 1: ленточная схема

Топология информационных связей подзадач в виде кольцевой структуры:



Результаты вычислительных экспериментов

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



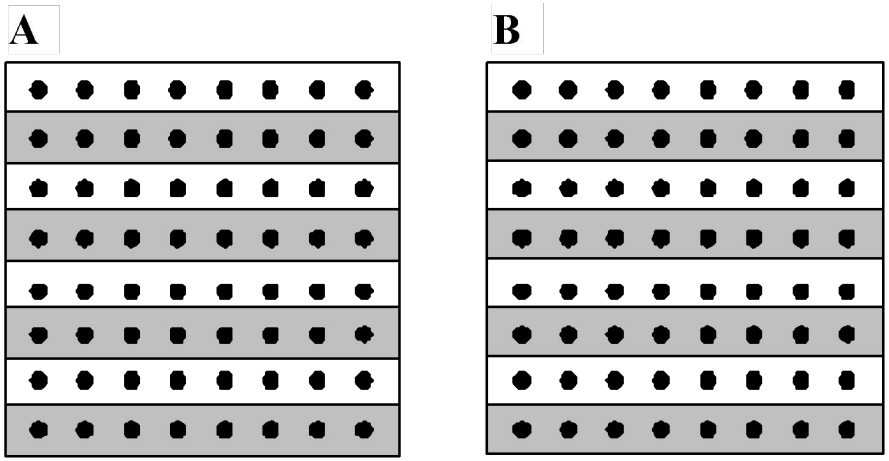
Параллельный алгоритм 1: ленточная схема

```
void MatrixMultiplicationMPI(double *&A, double *&B, double *&C, int &Size)
{
 int dim = Size; int i, j, k, p, ind;
double temp;

MPI_Status Status;
int ProcPartSize = dim/ProcNum;
int ProcPartElem = ProcPartSize*dim;
double* bufA = new double[ProcPartElem];
double* bufB = new double[ProcPartElem];
double* bufC = new double[ProcPartElem];
int ProcPart = dim/ProcNum,
part = ProcPart*dim;

if (ProcRank == 0) { Flip(B, Size); }
MPI_Scatter(A, part, MPI_DOUBLE, bufA, part, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Scatter(B, part, MPI_DOUBLE, bufB, part, MPI_DOUBLE, 0, MPI_COMM_WORLD);
temp = 0.0;
for (i=0; i < ProcPartSize; i++)
        { for (j=0; j < ProcPartSize; j++)
                { for (k=0; k < dim; k++) temp += bufA[i*dim+k]*bufB[j*dim+k];
                 bufC[i*dim+j+ProcPartSize*ProcRank] = temp;
                temp = 0.0;
                }
        }
```

```
int NextProc;
int PrevProc;
for (p=1; p < ProcNum; p++)
        { NextProc = ProcRank+1;
          if (ProcRank == ProcNum-1) NextProc = 0;
          PrevProc = ProcRank-1;
          if (ProcRank == 0) PrevProc = ProcNum-1;
        MPI_Sendrecv_replace(bufB, part, MPI_DOUBLE, NextProc, 0, PrevProc, 0, MPI_COMM_WORLD,
&Status);
        temp = 0.0;
        for (i=0; i < ProcPartSize; i++)
                { for (j=0; j < ProcPartSize; j++)
                        { for (k=0; k < dim; k++)
                                { temp += bufA[i*dim+k]*bufB[j*dim+k];
                                }
                        if (ProcRank-p >= 0 ) ind = ProcRank-p;
                                else ind = (ProcNum-p+ProcRank);
                        bufC[i*dim+j+ind*ProcPartSize] = temp;
                        temp = 0.0;
                         }
                }
        }
MPI_Gather(bufC, ProcPartElem, MPI_DOUBLE, C, ProcPartElem, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
delete []bufA;
delete []bufB;
delete []bufC;
 }
```

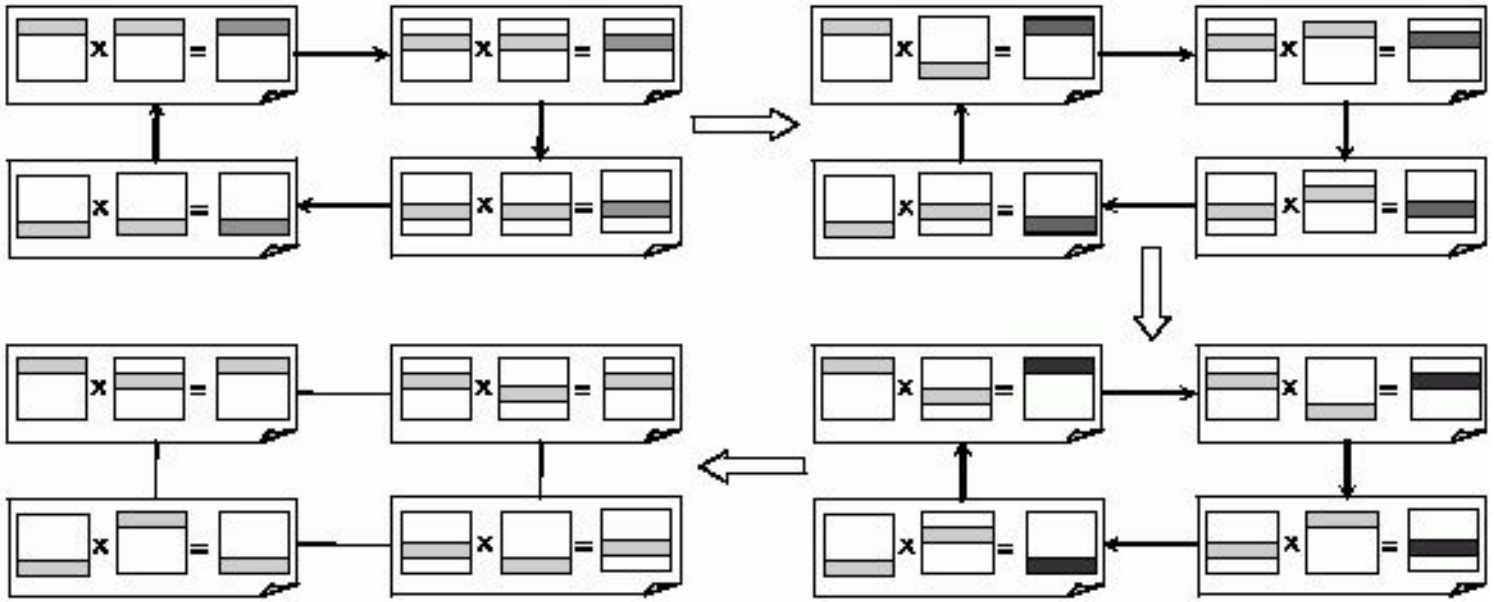
Параллельный алгоритм 2: ленточная схема

Идея: распределение данных в разбиении матриц ***A*** и ***B*** по строкам



- 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 2: ленточная схема



a_{00}	a_{01}	+	b_{00}	b_{01}	=	$c_{00}+=a_{00}*b_{00}$	$c_{01}+=a_{00}*b_{01}$
a_{10}	a_{11}		b_{10}	b_{11}		c_{10}	c_{11}

a_{00}	a_{01}	+	b_{00}	b_{01}	=	$c_{00}+=a_{01}*b_{10}$	$c_{01}+=a_{01}*b_{11}$
a_{10}	a_{11}		b_{10}	b_{11}		c_{10}	c_{11}

a_{00}	a_{01}	+	b_{00}	b_{01}	=	c_{00}	c_{01}
a_{10}	a_{11}		b_{10}	b_{11}		$c_{10}+=a_{10}*b_{00}$	$c_{11}+=a_{10}*b_{01}$

a_{00}	a_{01}	+	b_{00}	b_{01}	=	c_{00}	c_{01}
a_{10}	a_{11}		b_{10}	b_{11}		$c_{10}+=a_{11}*b_{10}$	$c_{11}+=a_{11}*b_{11}$

Параллельный алгоритм 2: ленточная схема

Алгоритм

1. Вначале производится рассылка в процесс ранга K элементов K-й строки матрицы A и элементов K-й строки матрицы B.
2. Элементы строки c, в которой будет содержаться соответствующая строка произведения AB, обнуляются.
3. Затем запускается цикл (число итераций равно N), в ходе которого выполняются два действия:

   1) выполняется перемножение элементов строк матрицы A и матрицы B с одинаковыми номерами, и результаты добавляются к соответствующему элементу строки c;

   2) выполняется циклическая пересылка строк матрицы B в соседние процессы (направление пересылки может быть произвольным: как по возрастанию рангов процессов, так и по их убыванию).

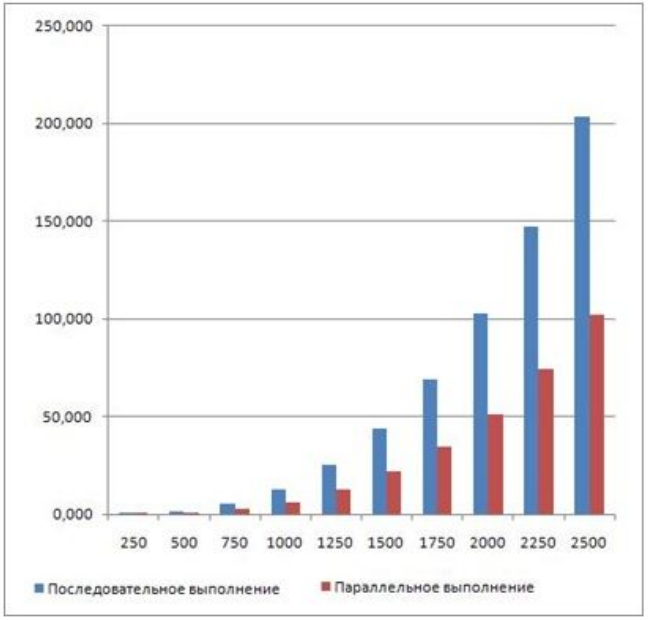
   После завершения цикла в каждом процессе будет содержаться соответствующая строка произведения AB.
4. Останется переслать эти строки главному процессу.

Результаты вычислительных экспериментов

OpenMP

Intel(R) Core(TM) 2 CPU 6300 @ 1.86GHz
RAM: 1 Гб ОЗУ

Размер	Последовательное выполнение, с	Параллельное выполнение, с	Ускорение
250	0,203	0,109	1,86
500	1,594	0,797	2,00
750	5,437	2,719	1,99
1000	12,891	6,406	2,01
1250	25,204	12,531	2,01
1500	43,546	21,719	2,00
1750	69,187	34,516	2,00
2000	103,078	51,391	2,00
2250	146,953	74,094	1,98
2500	203,390	101,907	1,99



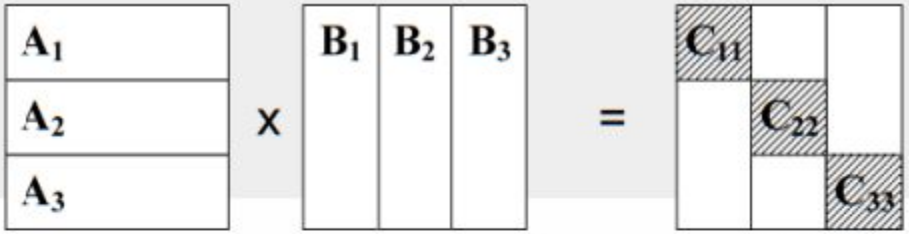
MPI

Inte(R) Core(TM) 2 QUAD CPU @2.40Ghz
RAM: 4 Гб ОЗУ

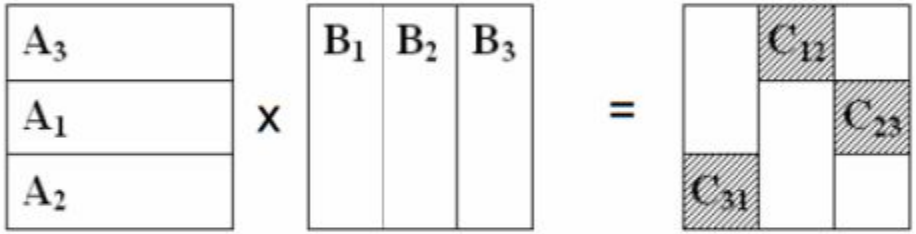
Размер	Последова-тельное выполнение, с	Парал-лельное выполнение, с	Уско-рение	Парал-лельное выполнение, с	Уско-рение	Парал-лельное выполнение, с	Уско-рение
250	0,114	0,061	1,87	0,046	2,48	0,039	2,92
500	1,002	0,474	2,11	0,328	3,05	0,269	3,72
750	4,309	1,708	2,52	1,101	3,91	0,857	5,03
1000	11,528	5,369	2,14	3,187	3,62	2,096	5,50
1250	25,920	12,090	2,14	7,023	3,96	4,856	5,34
1500	44,857	22,229	2,02	14,711	3,05	9,913	4,53
1750	72,031	36,256	1,99	24,341	2,96	18,091	3,98
2000	106,999	53,743	1,99	36,241	2,95	28,005	3,82
2250	155,712	78,885	1,97	53,194	2,93	41,272	3,77
2500	215,040	108,760	1,97	73,283	2,93	58,263	3,69

Простой блочный алгоритм

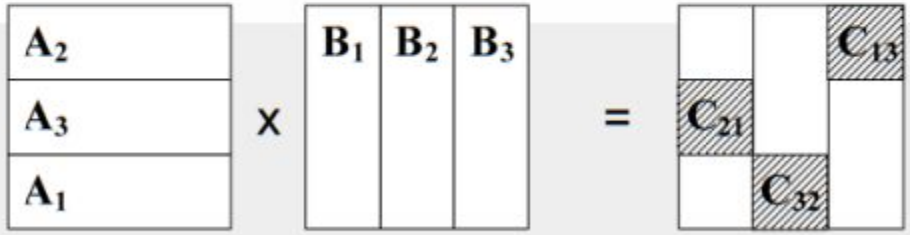
- Этап 1



- Этап 2



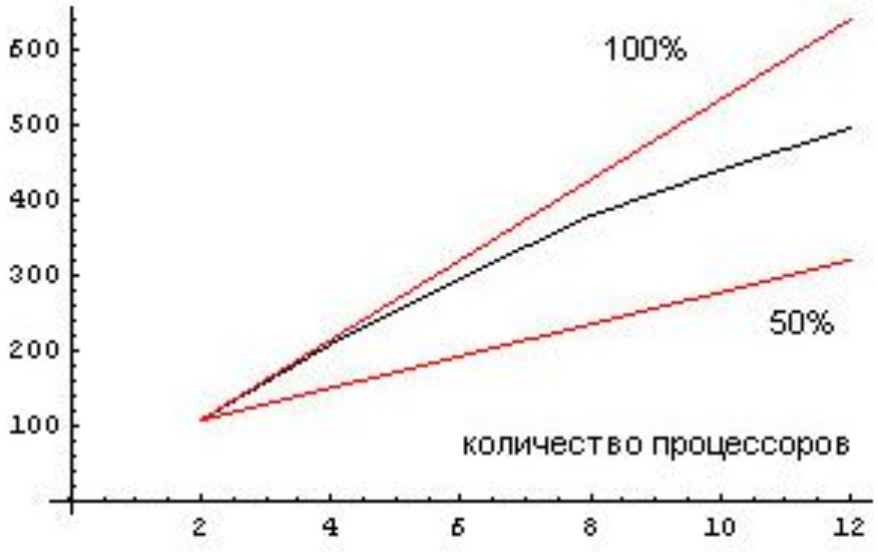
- Этап 3



- Объем памяти для 1ПЭ – $3n^2/p$ чисел; суммарный объем пересылаемых данных – n^2 чисел

Простой блочный алгоритм

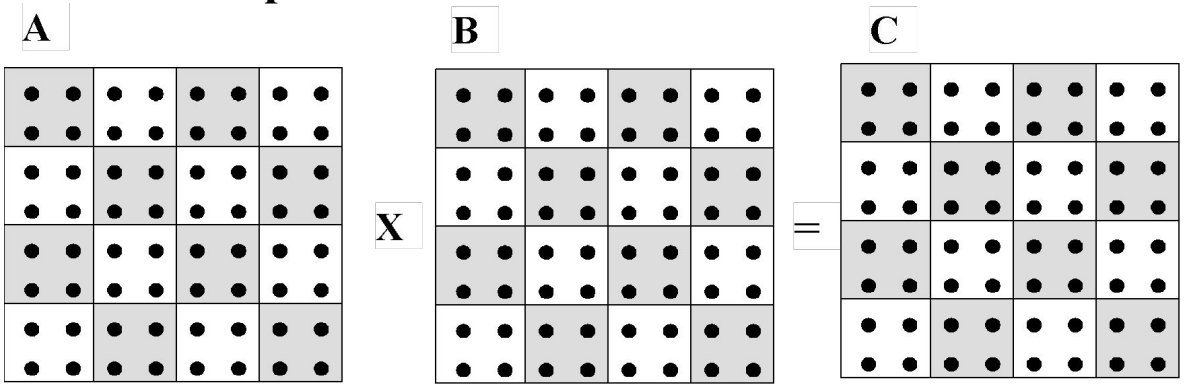
- Матрица A распределяется по процессорам блоками, содержащими $\left(\frac{n}{p}\right) \times n$ элементов, B - $n \times \left(\frac{n}{p}\right)$ элементов.
- Проводится матричное умножение соответствующих блоков, и в результате определяются блоки матрицы C размером $\left(\frac{n}{p}\right) \times \left(\frac{n}{p}\right)$, стоящие на главной диагонали.
- На каждой следующей итерации производится обмен блоками матрицы A между ПЭ циклическим сдвигом по уменьшению номера процессора (или по увеличению).
- После p итераций расчет матриц C заканчивается.



Вычисление произведения матриц в конечном поле с помощью простой параллельной схемы умножения. Коэффициент ускорения равен 77%.

Метод Фокса

Распределение данных происходит по Блочной схеме



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

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

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

Параллельный алгоритм 2: метод Фокса

- **Выделение информационных зависимостей -** для каждой итерации l, $0 \le l < q$:
  - 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:

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

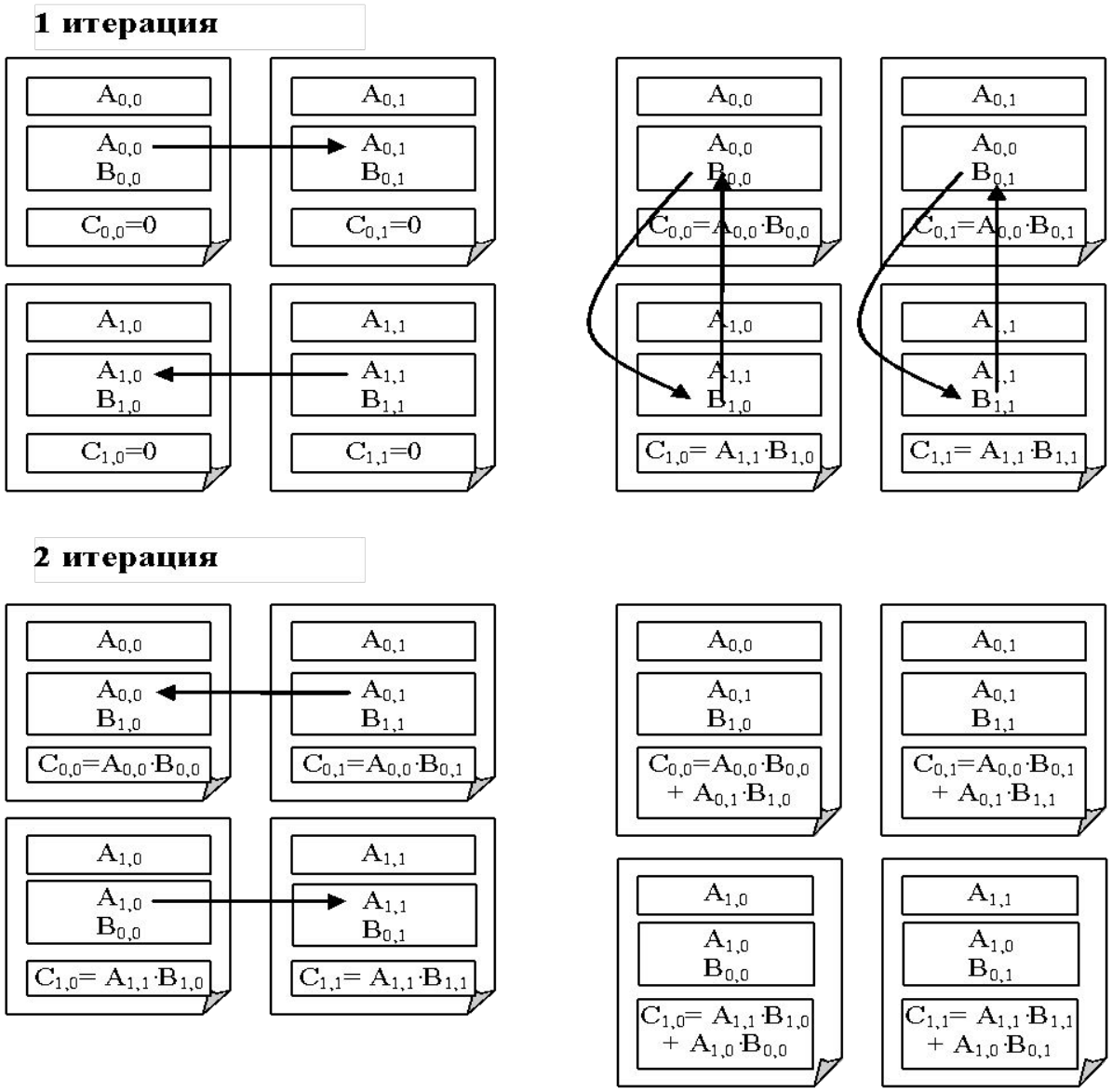
    где mod есть операция получения остатка от целого деления;
  - полученные в результате пересылок блоки A_{ij}', B_{ij}' каждой подзадачи (i,j) перемножаются и прибавляются к блоку C_{ij}

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

  - 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



Масштабирование и распределение подзадач по процессорам

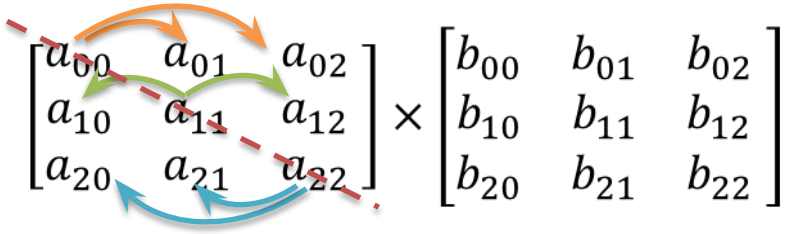
- Размеры блоков могут быть подобраны таким образом, чтобы общее количество базовых подзадач совпадало с числом процессоров ***p***,
- Наиболее эффективное выполнение метода Фокса может быть обеспечено при представлении множества имеющихся процессоров в виде квадратной решетки,
- В этом случае можно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

Пример: метод Фокса

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{20} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix} =$$

$$= \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Итерация 1. Диагональные процессы раздают свои элементы матрицы A направо соседям, матрица B без изменений:



$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

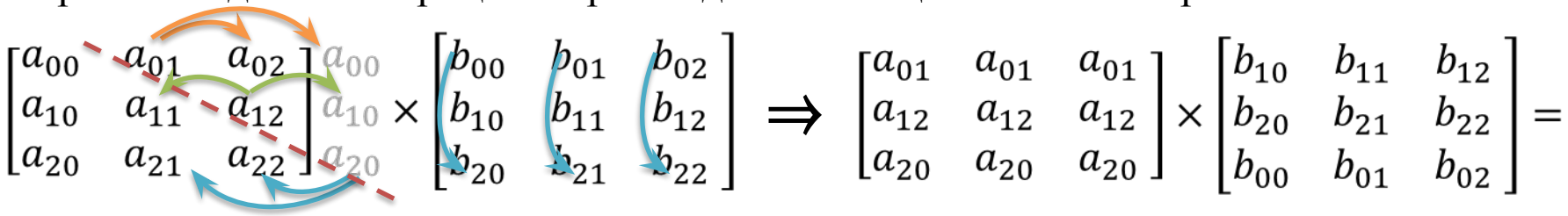
$$\begin{bmatrix} a_{00} & a_{00} & a_{00} \\ a_{11} & a_{11} & a_{11} \\ a_{22} & a_{22} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix} == \begin{bmatrix} a_{00}b_{00} & a_{00}b_{01} & a_{00}b_{02} \\ a_{11}b_{10} & a_{11}b_{11} & a_{11}b_{12} \\ a_{22}b_{20} & a_{22}b_{21} & a_{22}b_{22} \end{bmatrix}$$

Пример: метод Фокса

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{20} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix} =$$

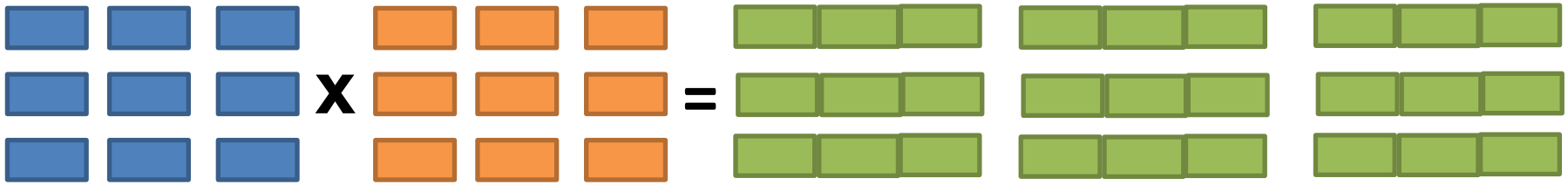
$$= \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Итерация 2. Верхнедиагональные процессы раздают свои элементы матрицы A направо соседям. В матрице *B* строки сдвигаются циклически вверх:

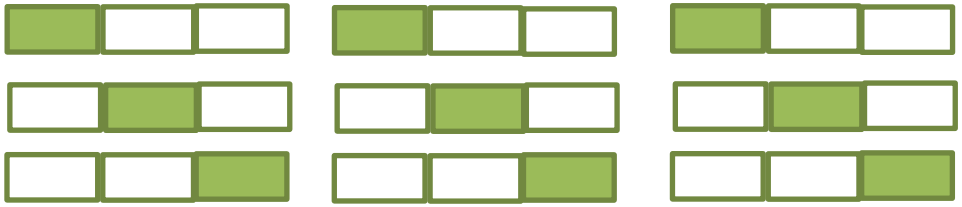


$$= \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + \\ + a_{11}b_{10} + a_{12}b_{20} \\ a_{20}b_{00} + + + a_{22}b_{20} \end{bmatrix} \begin{bmatrix} a_{00}b_{01} + a_{01} + \\ + a_{11}b_{11} + a_{12}b_{21} \\ a_{20}b_{01} + + + a_{22}b_{21} \end{bmatrix} \begin{bmatrix} a_{00}b_{02} + a_{01} + \\ + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{02} + + + a_{22}b_{22} \end{bmatrix}$$

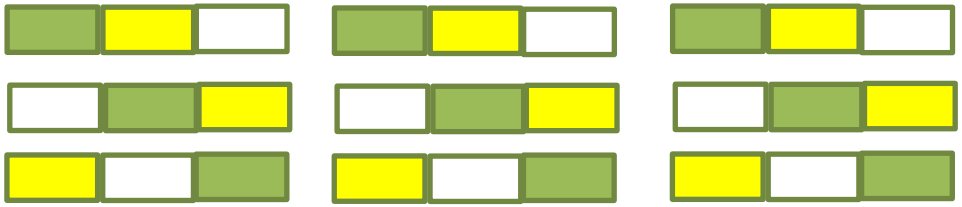
Пример: метод Фокса



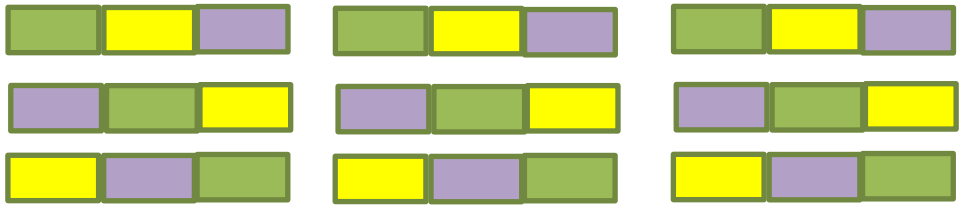
Итерация 1.



Итерация 2.

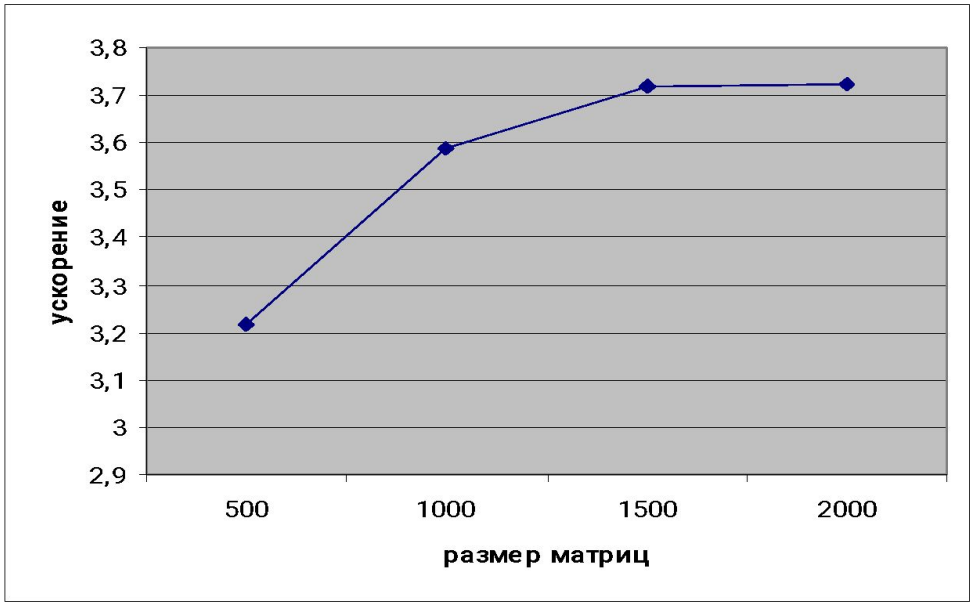


Итерация 3.

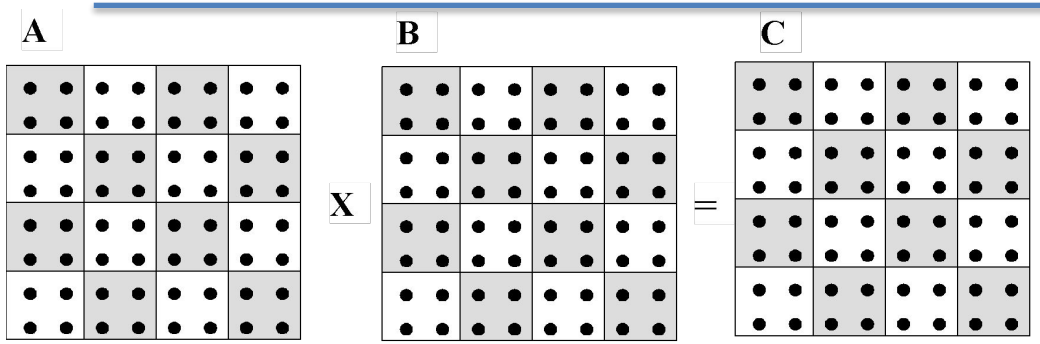


Результаты вычислительных экспериментов

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



Алгоритм Кэннона при блочном разделении данных



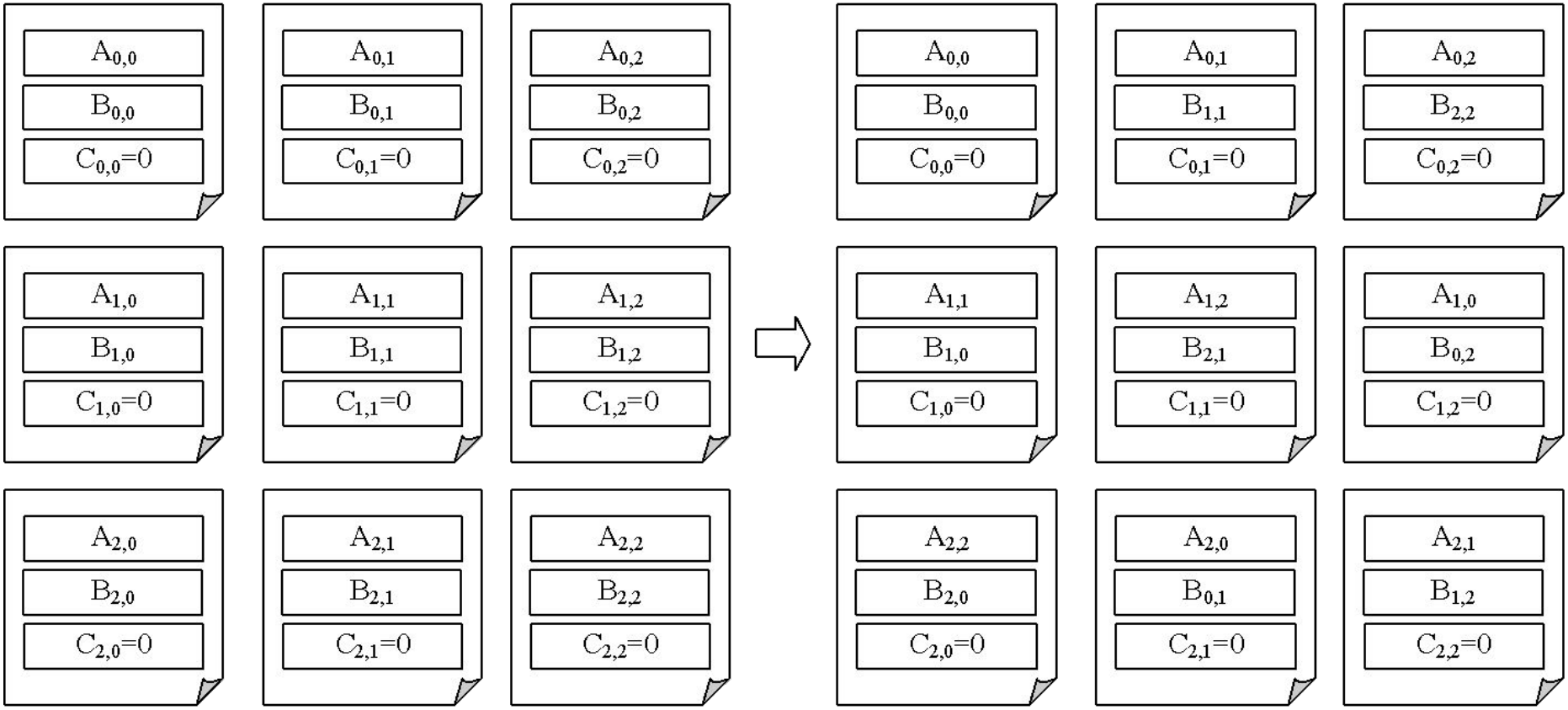
Базовая подзадача - процедура вычисления всех элементов одного из блоков матрицы C

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

- Подзадача *(i,j)* отвечает за вычисление блока $\boldsymbol{C_{ij}}$, все подзадачи образуют прямоугольную решетку размером $q \boldsymbol{x} q$,
- 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 $\boldsymbol{A_{ij}}$, $\boldsymbol{B_{ij}}$,
  - 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



Пример: метод Кэннона

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{20} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix} =$$

$$= \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Итерация 1. Циклический сдвиг по строкам: 0-я строка на 0 элементов влево

1-я строка на 1 элемент влево

2-я строка на 2 элемента влево

Циклический сдвиг по столбцам: 0-й на 0 элементов вверх

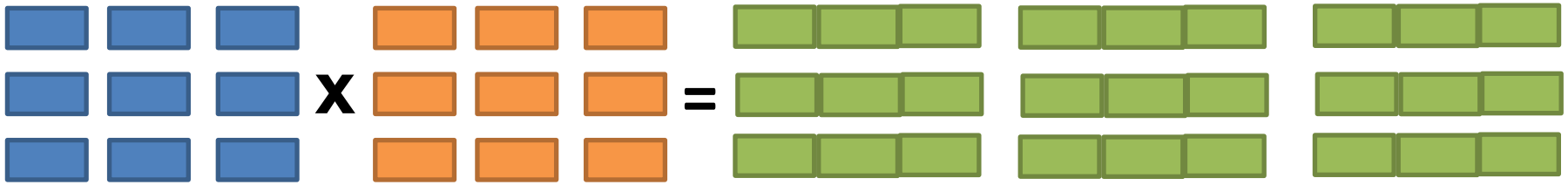
1-й на 1 элемент вверх

2-й на 2 элемента вверх

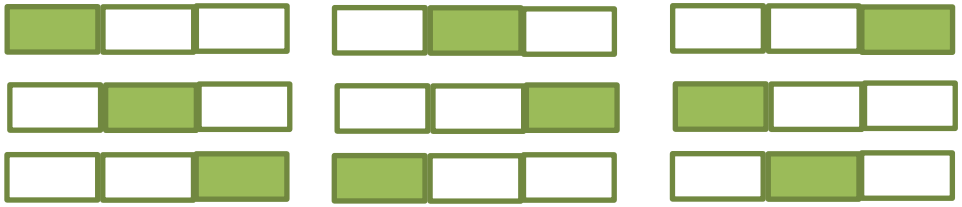
$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix} \Rightarrow \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{11} & a_{12} & a_{10} \\ a_{22} & a_{20} & a_{21} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{11} & b_{22} \\ b_{10} & b_{21} & b_{02} \\ b_{20} & b_{01} & b_{12} \end{bmatrix}$$

$$a_{20} \quad a_{21} \quad a_{22} \Rightarrow a_{21} \quad a_{22} \quad a_{20} \Rightarrow a_{22} \quad a_{20} \quad a_{21}$$

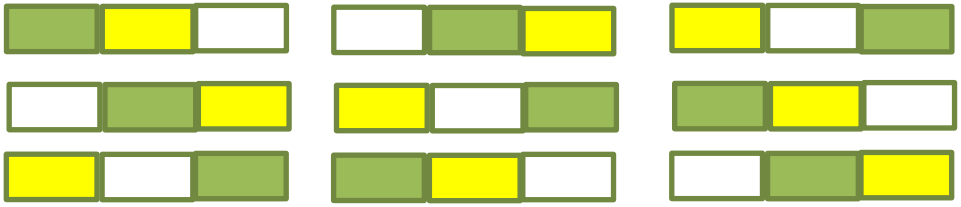
Пример: метод Фокса



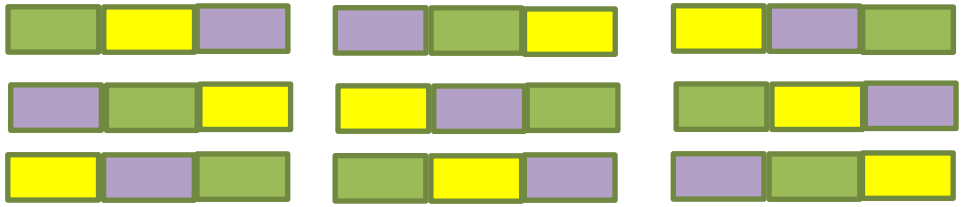
Итерация 1.



Итерация 2.



Итерация 3.



Параллельный алгоритм 3: метод Кэннона

Выделение информационных зависимостей

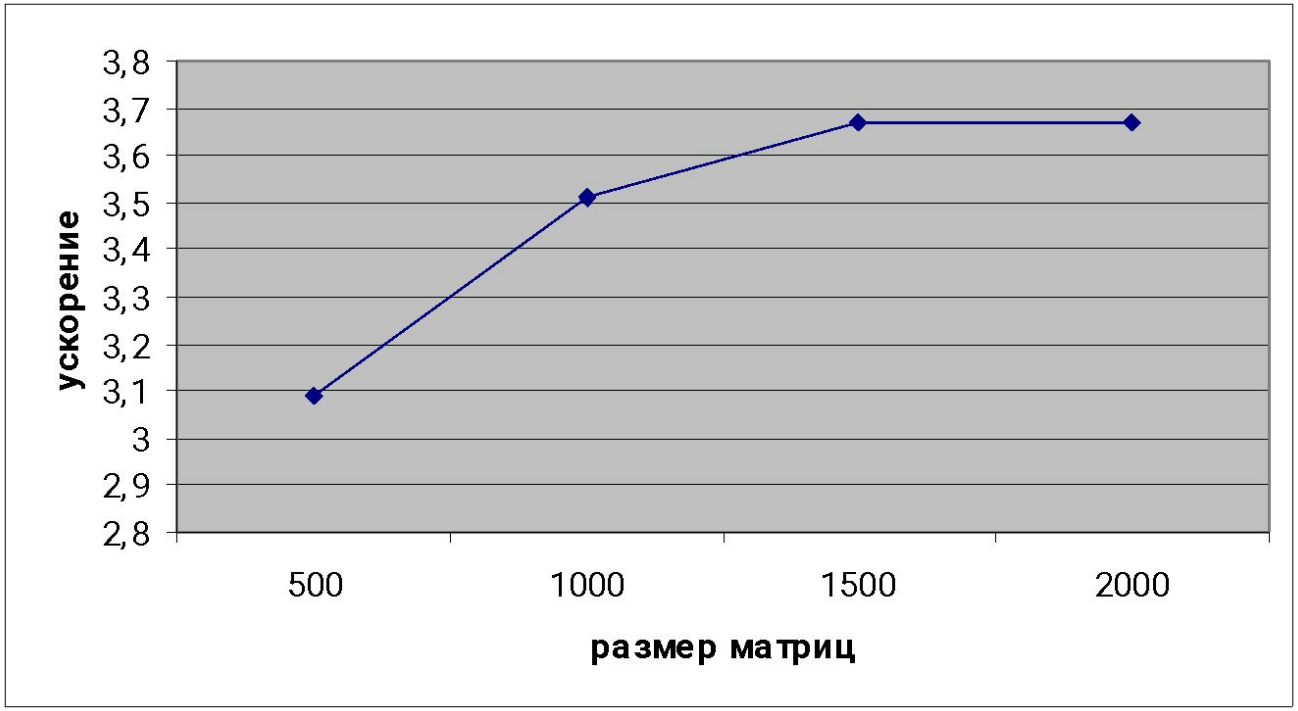
- В результате начального распределения в каждой базовой подзадаче будут располагаться блоки, которые могут быть перемножены без дополнительных операций передачи данных,
- Для получения всех последующих блоков после выполнения операции блочного умножения:
  - каждый блок матрицы A передается предшествующей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

Масштабирование и распределение подзадач по процессорам

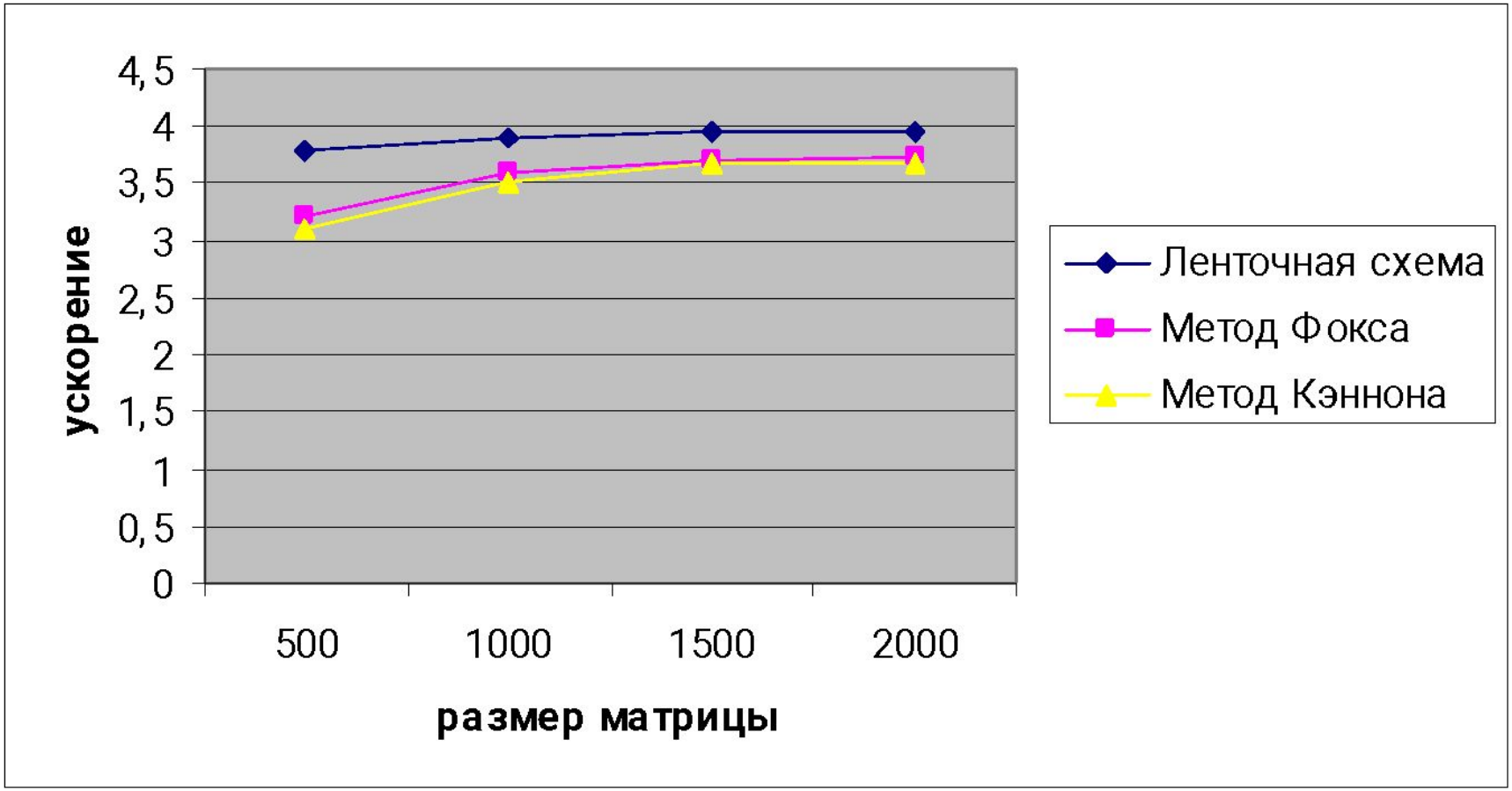
- Размер блоков может быть подобран таким образом, чтобы количество базовых подзадач совпадало с числом имеющихся процессоров,
- Множество имеющихся процессоров представляется в виде квадратной решетки и размещение базовых подзадач (i,j) осуществляется на процессорах $p_{i,j}$ (соответствующих узлов процессорной решетки)

Результаты вычислительных экспериментов

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


4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
ускорение
500
1000
1500
2000
размер матрицы
Ленточная схема
Метод Фокса
Метод Кэннона