Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Образовательный комплекс

Введение в методы параллельного программирования

Раздел 8.

Параллельные методы матричного умножения

Гергель В.П., профессор, д.т.н.
Кафедра математического обеспечения ЭВМ

Содержание

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

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***

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

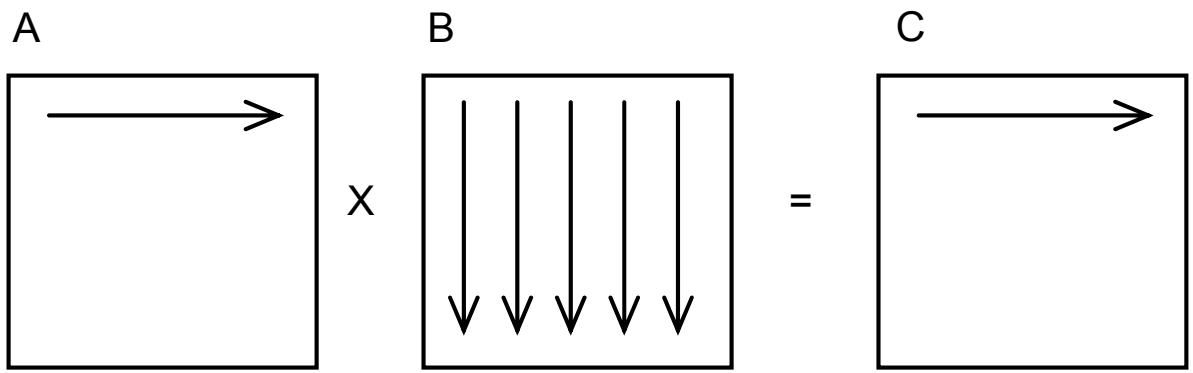
В основу организации параллельных вычислений может быть положен ***принцип распараллеливания по данным***

Последовательный алгоритм…

```
// Алгоритм 8.1
// 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…

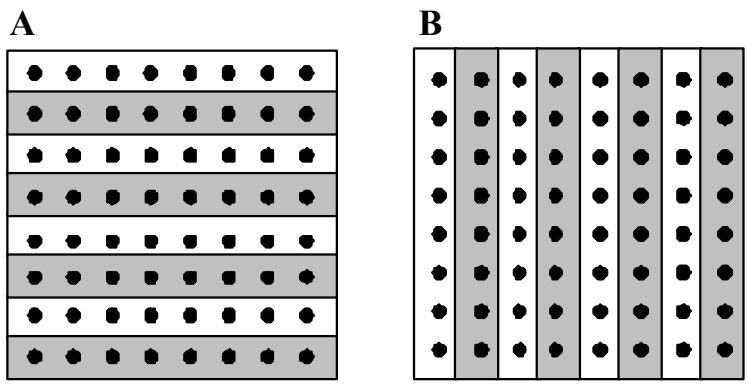
- **Возможный подход** – в качестве базовой подзадачи процедура вычисления одного из элементов матрицы ***C***

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

- 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)



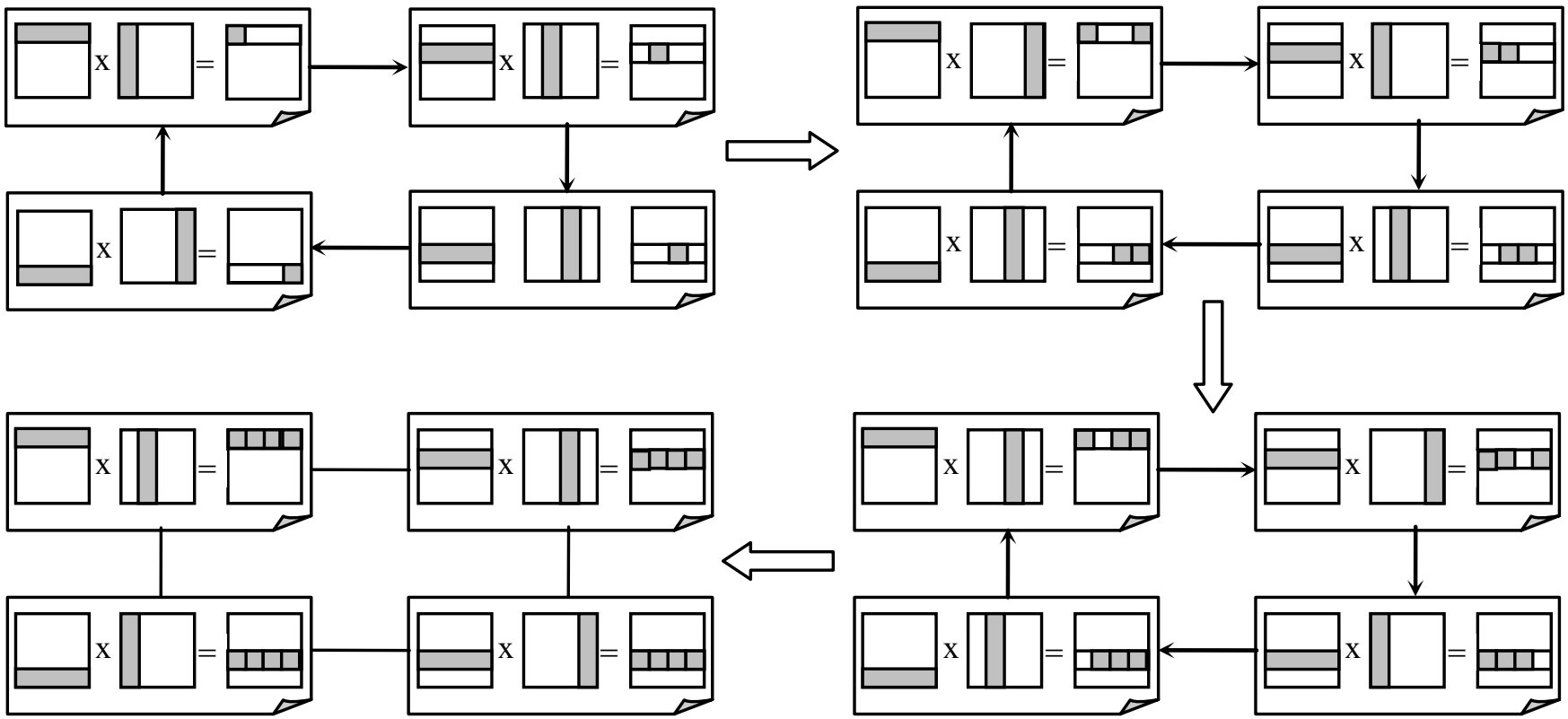
Параллельный алгоритм 1: ленточная схема…

- **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)

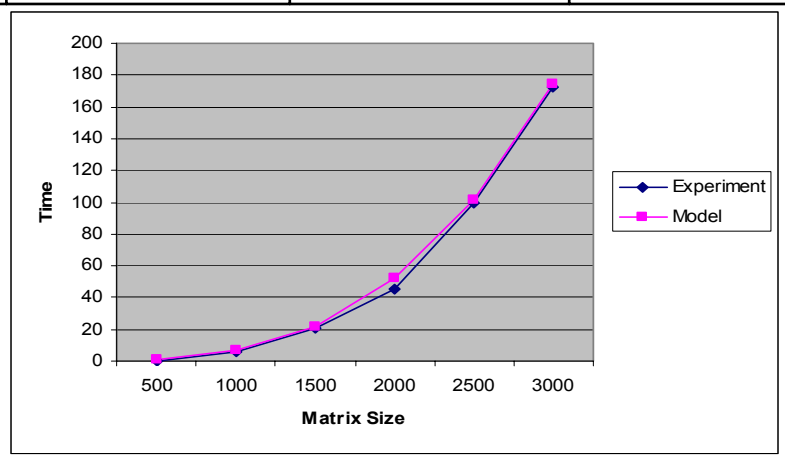
Общее время выполнения параллельного алгоритма составляет

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

Параллельный алгоритм 1: ленточная схема…

- **Результаты вычислительных экспериментов**
  - Сравнение теоретических оценок и экспериментальных данных

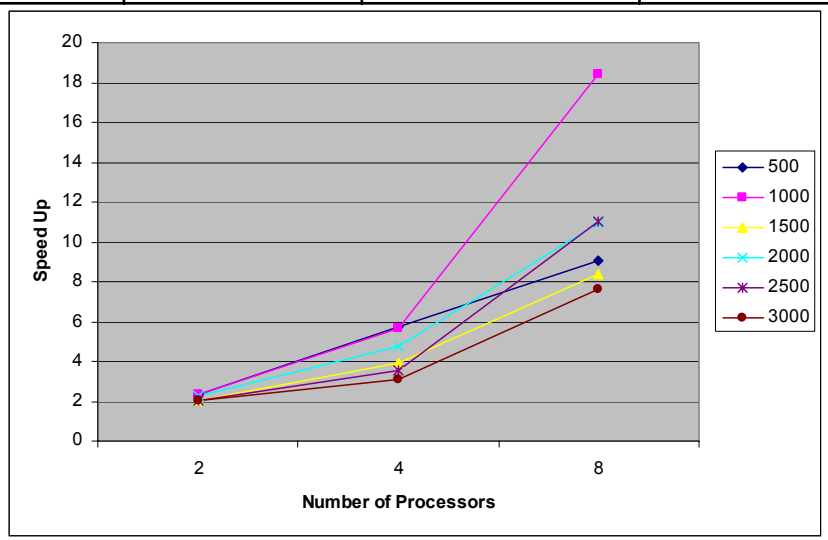
Размер матрицы	2 процессора		4 процессора		8 процессоров	
	Модель	Эксперимент	Модель	Эксперимент	Модель	Эксперимент
500	0,8243	0,3758	0,4313	0,1535	0,2353	0,0968
1000	6,51822	5,4427	3,3349	2,2628	1,7436	0,6998
1500	21,9137	20,9503	11,1270	11,0804	5,7340	5,1766
2000	51,8429	45,7436	26,2236	21,6001	13,4144	9,4127
2500	101,1377	99,5097	51,0408	56,9203	25,9928	18,3303
3000	174,6301	171,9232	87,9946	111,9642	44,6772	45,5482



Параллельный алгоритм 1: ленточная схема…

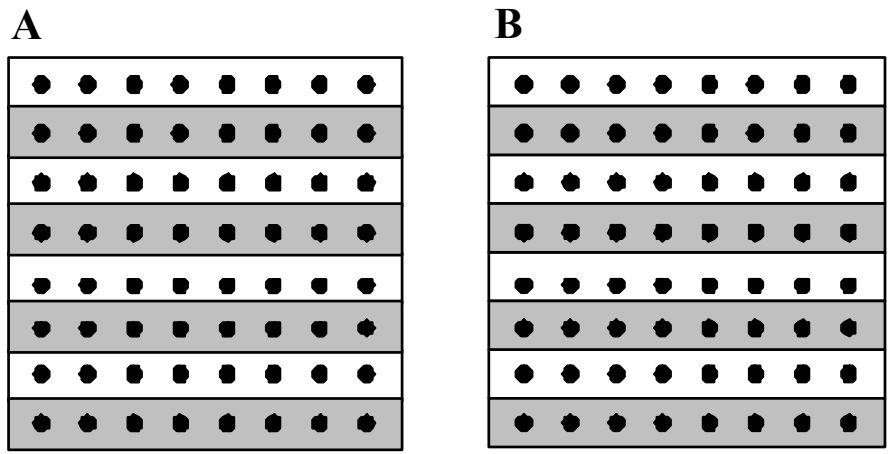
- **Результаты вычислительных экспериментов**
  - Ускорение вычислений

Размер матрицы	Последовательный алгоритм	2 процессора		4 процессора		8 процессоров	
		Время	Ускорение	Время	Ускорение	Время	Ускорение
500	0,8752	0,3758	2,3287	0,1535	5,6982	0,0968	9,0371
1000	12,8787	5,4427	2,3662	2,2628	5,6912	0,6998	18,4014
1500	43,4731	20,9503	2,0750	11,0804	3,9234	5,1766	8,3978
2000	103,0561	45,7436	2,2529	21,6001	4,7710	9,4127	10,9485
2500	201,2915	99,5097	2,0228	56,9203	3,5363	18,3303	10,9813
3000	347,8434	171,9232	2,0232	111,9642	3,1067	45,5482	7,6368



Параллельный алгоритм 1': ленточная схема…

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



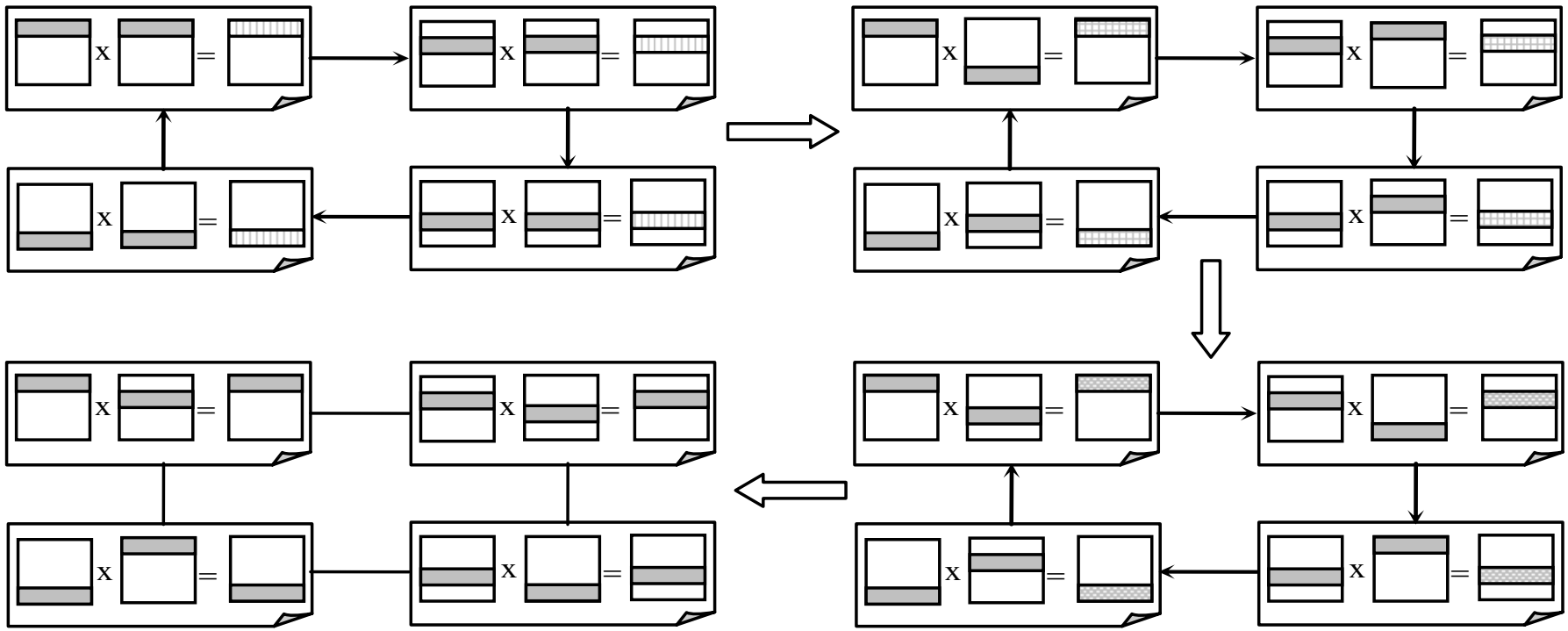
Параллельный алгоритм 1': ленточная схема…

- **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***

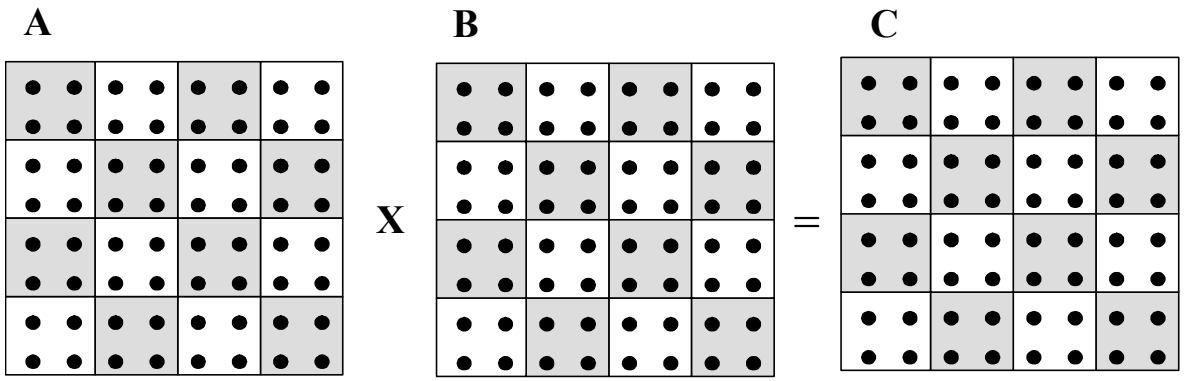
Параллельный алгоритм 1': ленточная схема

- Схема информационного взаимодействия



Параллельный алгоритм 2: метод Фокса

- **Распределение данных** – Блочная схема



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

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

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, $0 \le l < q$, 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:

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

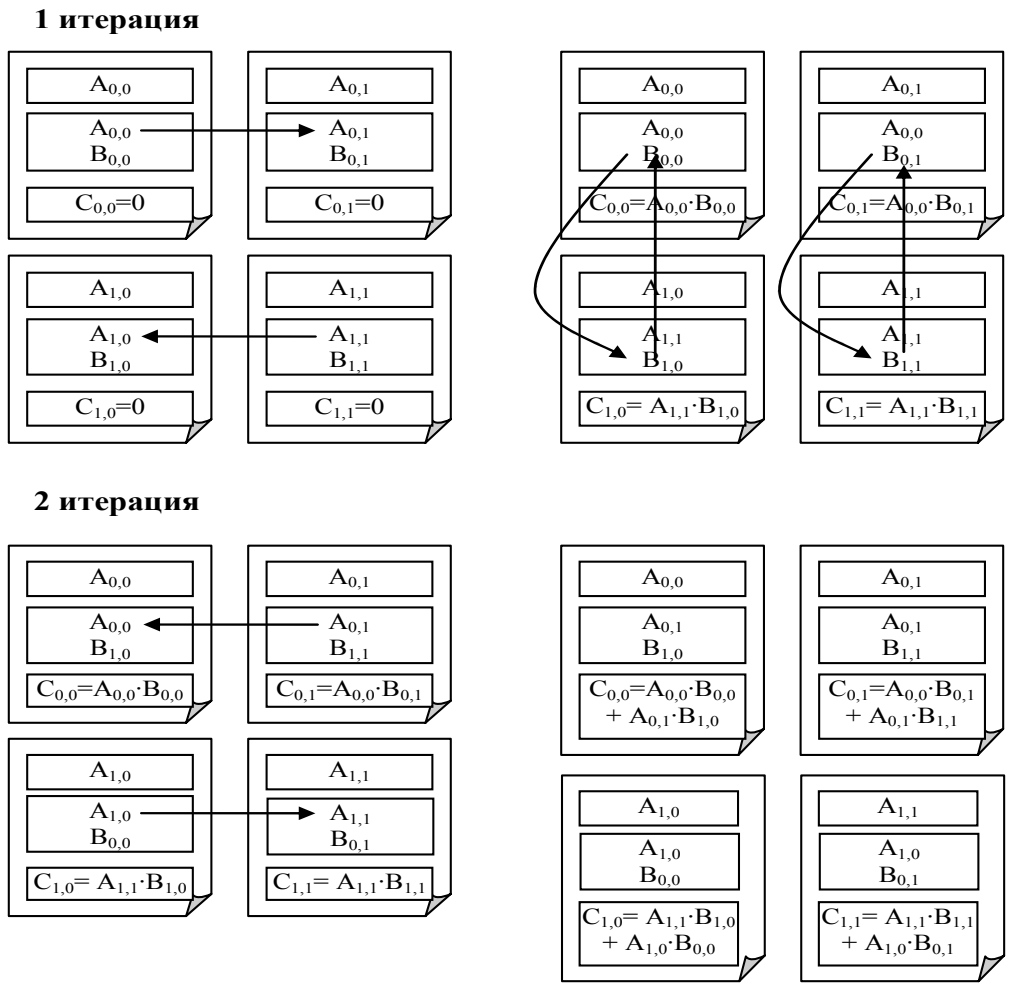
    где *mod* есть операция получения остатка от целого деления;
  - полученные в результате пересылок блоки A_{ij}', B_{ij}' каждой подзадачи (i,j) перемножаются и прибавляются к блоку C_{ij}

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

  - 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 $\boldsymbol{p}_{i,j}$

Параллельный алгоритм 2: метод Фокса…

- **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 2: метод Фокса…

❑ Анализ эффективности (уточненные оценки)

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

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

- На каждой итерации алгоритма один из процессоров строки процессорной решетки рассылает свой блок матрицы ***A*** остальным процессорам своей строки

$$T_p^1(comm) = \log_2 q\,(\alpha + w(n^2 / p) / \beta)$$

- После умножения матричных блоков процессоры передают свои блоки матрицы ***B*** предыдущим процессорам по столбцам процессорной решетки

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

Общее время выполнения параллельного алгоритма составляет

$$T_p = q[(n^2 / p) \cdot (2n / q - 1) + (n^2 / p)] \cdot \tau + (q \log_2 q + (q - 1))(\alpha + w(n^2 / p) / \beta)$$

Параллельный алгоритм 2: метод Фокса…

- **Программная реализация…**
  - **Основная функция**
    - Реализует логику работы алгоритма, последовательно вызывает необходимые процедуры

Программа

Параллельный алгоритм 2: метод Фокса…

- **Программная реализация …**
  - **Функция *CreateGridCommunicators*:**
    - Создает коммуникатор для двухмерной квадратной решетки,
    - Определяет координаты всех процессоров в рамках решетки,
    - Создает коммуникаторы отдельно для каждой строки и каждого столбца процессорной решетки.

Программа

Параллельный алгоритм 2: метод Фокса…

- **Программная реализация …**
  - **Функция *ProcessInitialization*:**
    - Определяет размеры матриц,
    - Выделяет память для хранения матриц и их блоков,
    - Определяет значения элементов исходных матриц при помощи вызовов функций *DummyDataInitialization* и *RandomDataInitialization*

Программа

Параллельный алгоритм 2: метод Фокса…

- **Программная реализация …**
  - **Функция *ParallelResultCalculation*:**
    - Выполняет параллельный алгоритм Фокса умножения матриц (матричные блоки и размер матричных блоков передаются функции в качестве аргументов)

Программа

Параллельный алгоритм 2: метод Фокса…

- **Программная реализация …**
  - Выполнение итерации: передача блоков матрицы ***A*** по строкам процессорной решетки (функция ***AblockCommunication***):
    - Определяется номер процессора *Pivot*, который должен выполнить рассылку,
    - Для рассылки используется блок *pMatrixAblock* матрицы A. Этот блок был размещен в памяти процессора до начала вычислений,
    - Рассылка блока вдоль строки осуществляется при помощи функции *MPI_Bcast* (используется коммуникатор *RowComm*).

Программа

Параллельный алгоритм 2: метод Фокса…

- **Программная реализация …**
  - Выполнение итерации: Выполнение умножения матричных блоков (функция ***BlockMultiplication***):
    - Эта функция перемножает блоки матрицы ***A*** (*pAblock*) и ***B*** (*pBblock*) и прибавляет результат умножения к блоку матрицы ***C***

Программа

Параллельный алгоритм 2: метод Фокса…

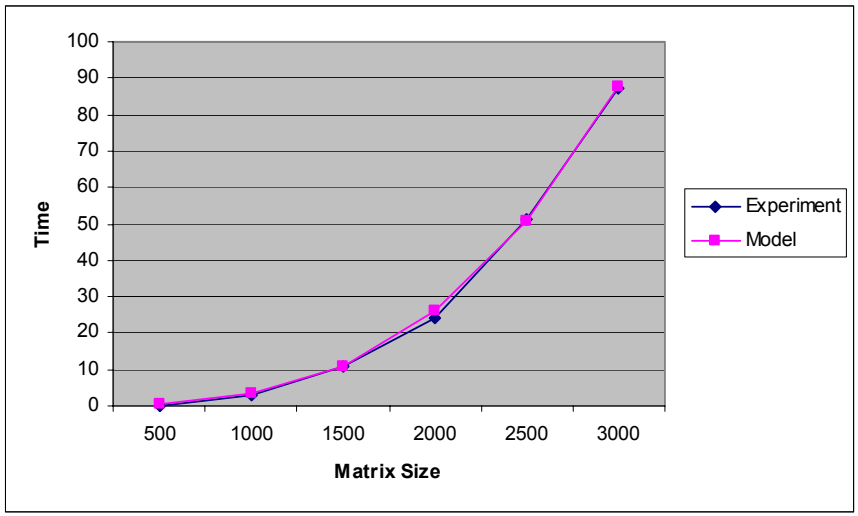
- **Программная реализация**
  - Выполнение итерации: Циклический сдвиг блоков матрицы ***B*** вдоль столбцов процессорной решетки (функция ***BblockCommunication***):
    - Каждый процессор передает свой блок процессору того же столбца процессорной решетки с номером *NextProc,*
    - Каждый процесс получает блок от процессора того же столбца процессорной решетки с номером *PrevProc,*
    - Для выполнения циклического сдвига используется функция *MPI_Sendrecv_replace.*

Программа

Параллельный алгоритм 2: метод Фокса…

- **Результаты вычислительных экспериментов**
  - Сравнение теоретических оценок и экспериментальных данных

Размер матрицы	4 процессора		9 процессоров	
	Модель	Эксперимент	Модель	Эксперимент
500	0,4217	0,2190	0,2200	0,1468
1000	3,2970	3,0910	1,5924	2,1565
1500	11,0419	10,8678	5,1920	7,2502
2000	26,0726	24,1421	12,0927	21,4157
2500	50,8049	51,4735	23,3682	41,2159
3000	87,6548	87,0538	40,0923	58,2022

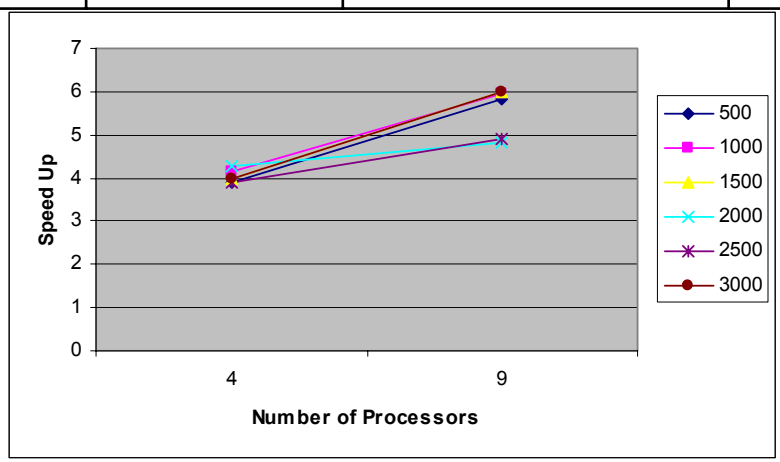


Параллельный алгоритм 2: метод Фокса

❑ Результаты вычислительных экспериментов

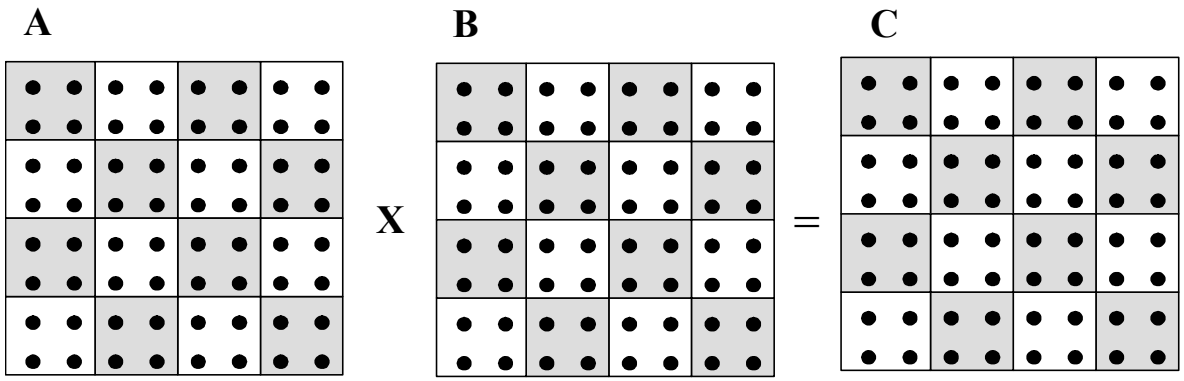
– Ускорение вычислений

Размер матрицы	Последовательный алгоритм	Параллельный алгоритм			
		4 процессора		9 процессоров	
		Время	Ускорение	Время	Ускорение
500	0,8527	0,2190	3,8925	0,1468	5,8079
1000	12,8787	3,0910	4,1664	2,1565	5,9719
1500	43,4731	10,8678	4,0001	7,2502	5,9960
2000	103,0561	24,1421	4,2687	21,4157	4,8121
2500	201,2915	51,4735	3,9105	41,2159	4,8838
3000	347,8434	87,0538	3,9957	58,2022	5,9764



Параллельный алгоритм 3: метод Кэннона…

- **Распределение данных** – Блочная схема



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

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

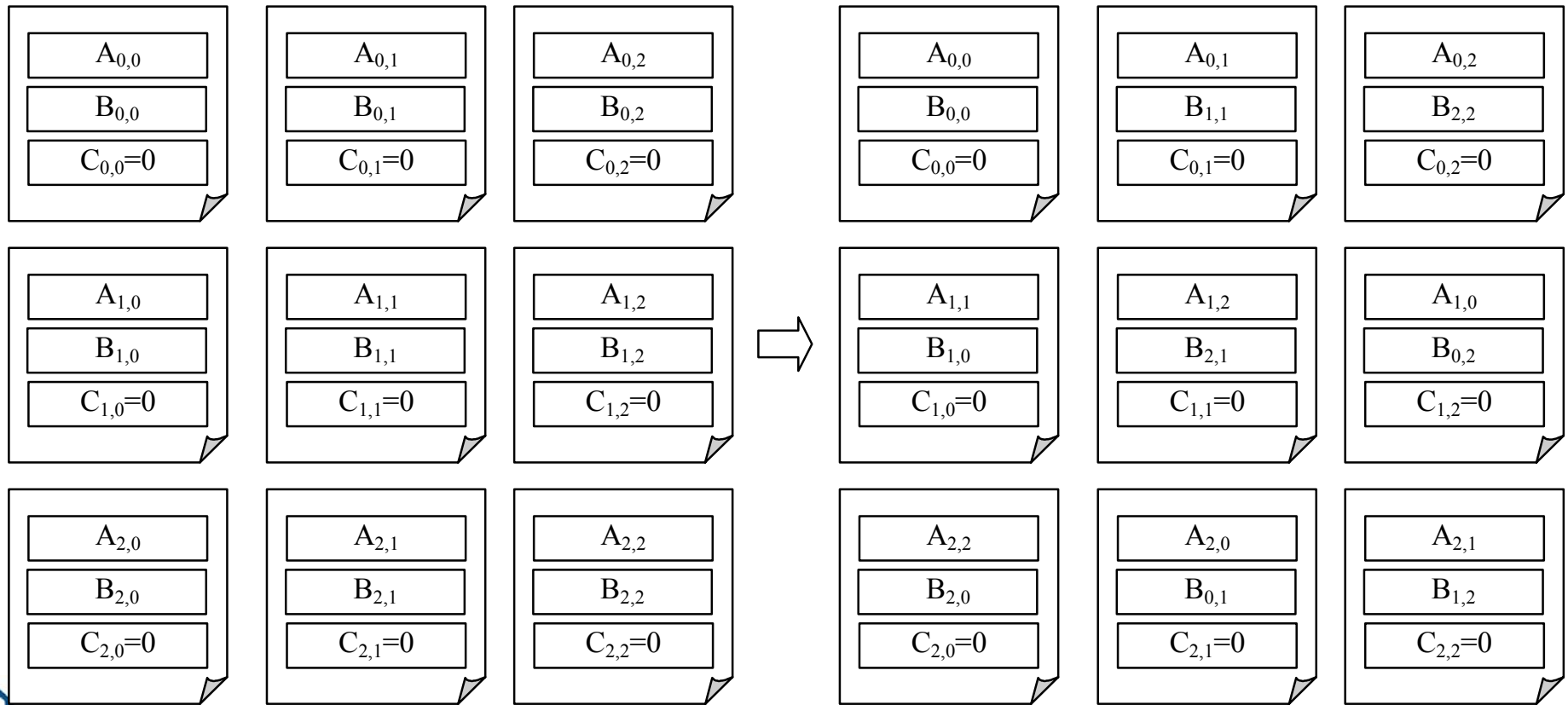
Параллельный алгоритм 3: метод Кэннона…

❑ Выделение информационных зависимостей

- Подзадача *(i,j)* отвечает за вычисление блока C_{ij}, все подзадачи образуют прямоугольную решетку размером *qxq*,
- 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…

- **Анализ эффективности** (уточненные оценки)

- Алгоритм Кэннона отличается от метода Фокса только видом выполняемых в ходе вычислений коммуникационных операций, следовательно:

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

- На этапе инициализации производится перераспределение блоков матриц ***A*** и ***B*** при помощи циклического сдвига матричных блоков по строкам и столбцам процессорной решетки (предполагаем, что топология системы представляет собой полный граф)

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

На каждой итерации алгоритма после умножения матричных блоков процессоры передают свои блоки предыдущим процессорам по строкам (для блоков матрицы ***A***) и столбцам (для блоков матрицы ***B***)

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

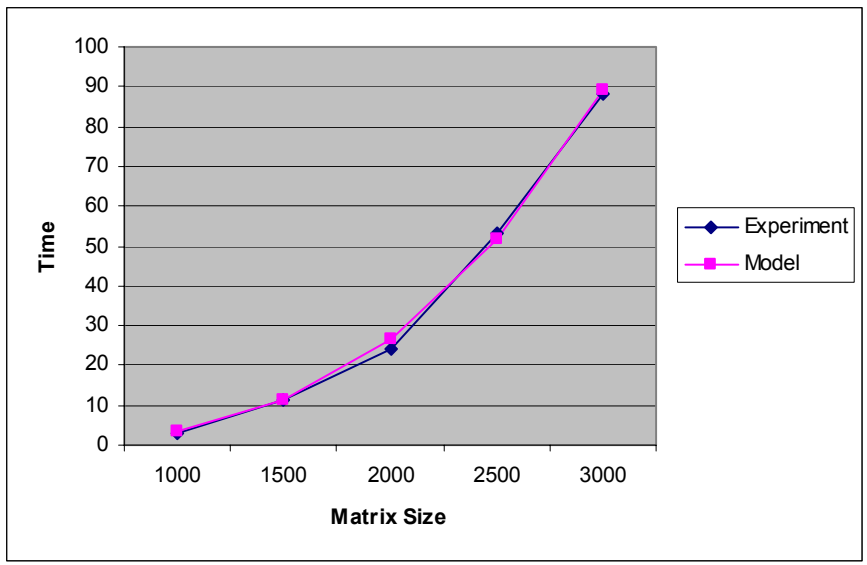
Общее время выполнения параллельного алгоритма составляет

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

Параллельный алгоритм 3: метод Кэннона…

- **Результаты вычислительных экспериментов**
  - Сравнение теоретических оценок и экспериментальных данных

Размер матрицы	4 процессора		9 процессоров	
	Модель	Эксперимент	Модель	Эксперимент
1000	3,4485	3,0806	1,5669	1,1889
1500	11,3821	11,1716	5,1348	4,6310
2000	26,6769	24,0502	11,9912	14,4759
2500	51,7488	53,1444	23,2098	23,5398
3000	89,0138	88,2979	39,8643	36,3688

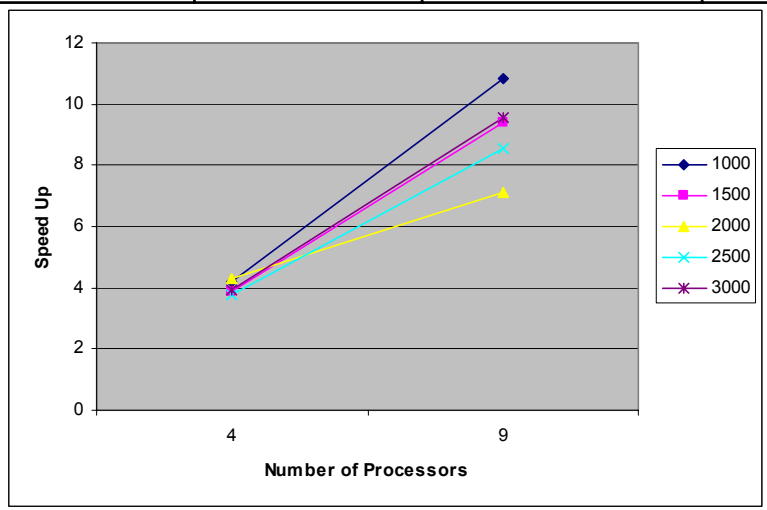


Параллельный алгоритм 3: метод Кэннона

❑ Результаты вычислительных экспериментов

– Ускорение вычислений

Размер матриц	Последовательный алгоритм	Параллельный алгоритм			
		4 процессора		9 процессоров	
		Время	Ускорение	Время	Ускорение
1000	12,8787	3,0806	4,1805	1,1889	10,8324
1500	43,4731	11,1716	3,8913	4,6310	9,3872
2000	103,0561	24,0502	4,2850	14,4759	7,1191
2500	201,2915	53,1444	3,7876	23,5398	8,5511
3000	347,8434	88,2979	3,9394	36,3688	9,5643

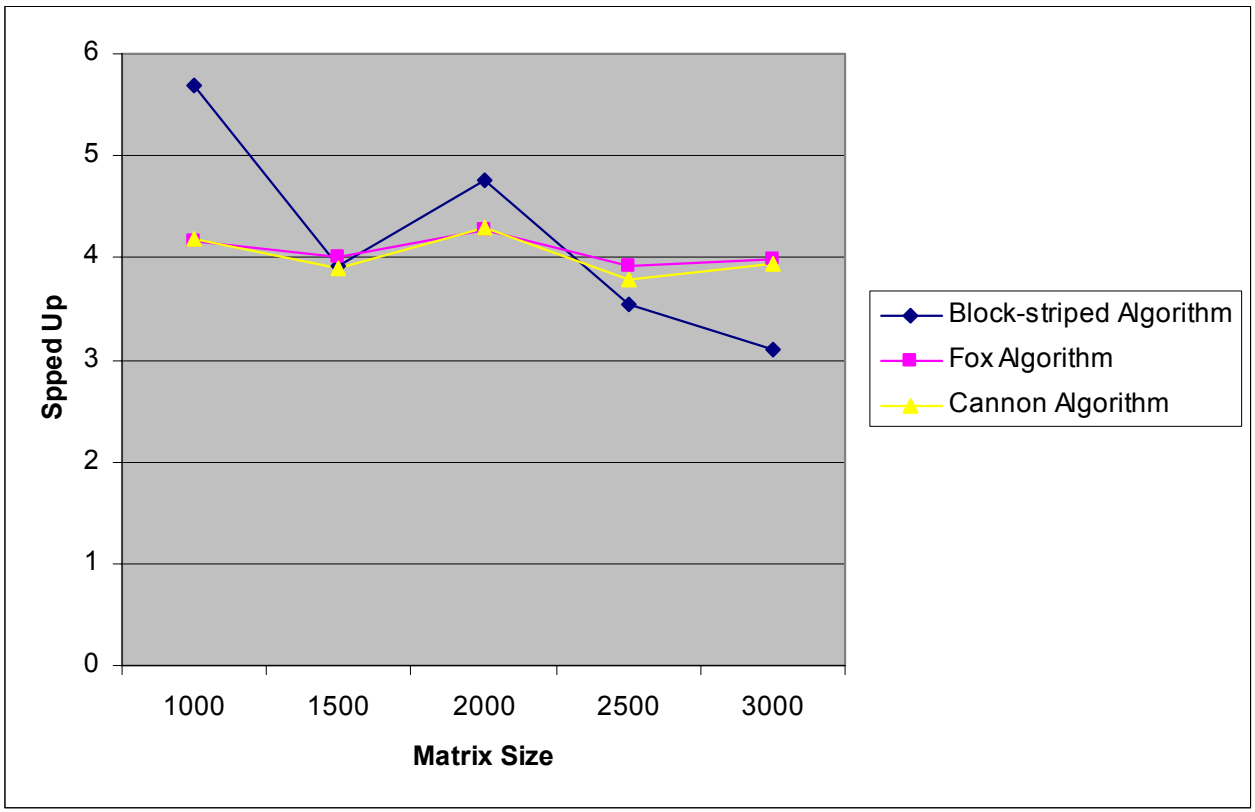


Заключение…

- Рассмотрены три возможных параллельных реализации одной из наиболее часто используемых матричных операций – матричного умножения:
  - Алгоритм 1 – ленточное разбиение данных,
  - Алгоритм 2 – метод Фокса (блочная схема),
  - Алгоритм 3 – метод Кэннона (блочная схема)
- Представлена программная реализация метода Фокса
- Теоретические оценки позволяют достаточно точно определить показатели эффективности параллельных вычислений

Заключение

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



Вопросы для обсуждения

- Какие последовательные алгоритмы выполнения операции умножения матриц вы знаете? Какова их вычислительная трудоемкость?
- Какой основной подход используется при разработке параллельных алгоритмов матричного умножения?
- Какой из алгоритмов обладает наилучшими показателями ускорения и эффективности?
- Какой из рассмотренных алгоритмов характеризуется наименьшими и наибольшими требованиями к объему необходимой памяти?
- Какие операции передачи данных необходимы в параллельных алгоритмах матричного умножения?

Темы заданий для самостоятельной работы

- Выполните реализацию двух ленточных алгоритмов умножения матриц. Сравните время выполнения этих алгоритмов.
- Выполните реализацию алгоритма Кэннона.
- Постройте теоретические оценки времени работы этих алгоритмов с учетом параметров используемой вычислительной системы
- Проведите вычислительные эксперименты. Сравните результаты реальных экспериментов с полученными теоретическими оценками

Литература

- **Kumar** V., Grama, A., Gupta, A., Karypis, G. (1994). Introduction to Parallel Computing. - The Benjamin/Cummings Publishing Company, Inc. (2nd edn., 2003)
- **Quinn**, M. J. (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.
- **Fox**, G.C., Otto, S.W. and Hey, A.J.G. (1987) Matrix Algorithms on a Hypercube I: Matrix Multiplication. Parallel Computing. 4 H. 17-31.

Следующая тема

- **Параллельные методы решения систем линейных уравнений**

Авторский коллектив

Гергель В.П., профессор, д.т.н., руководитель

Гришагин В.А., доцент, к.ф.м.н.

Абросимова О.Н., ассистент (раздел 10)

Лабутин Д.Ю., ассистент (система ПараЛаб)

Курылев А.Л., ассистент (лабораторные работы 4, 5)

Сысоев А.В., ассистент (раздел 1)

Гергель А.В., аспирант (раздел 12, лабораторная работа 6)

Лабутина А.А., аспирант (разделы 7,8,9, лабораторные работы 1, 2, 3, система ПараЛаб)

Сенин А.В., аспирант (раздел 11, лабораторные работы по Microsoft Compute Cluster)

Ливерко С.В. (система ПараЛаб)

О проекте

Целью проекта является создание образовательного комплекса "Многопроцессорные вычислительные системы и параллельное программирование", обеспечивающий рассмотрение вопросов параллельных вычислений, предусматриваемых рекомендациями Computing Curricula 2001 Международных организаций IEEE-CS и ACM. Данный образовательный комплекс может быть использован для обучения на начальном этапе подготовки специалистов в области информатики, вычислительной техники и информационных технологий.

Образовательный комплекс включает **учебный курс "Введение в методы параллельного программирования" и лабораторный практикум "Методы и технологии разработки параллельных программ"**, что позволяет органично сочетать фундаментальное образование в области программирования и практическое обучение методам разработки масштабного программного обеспечения для решения сложных вычислительно-трудоемких задач на высокопроизводительных вычислительных системах.

Проект выполнялся в Нижегородском государственном университете им. Н.И. Лобачевского на кафедре математического обеспечения ЭВМ факультета вычислительной математики и кибернетики (http://www.software.unn.ac.ru). Выполнение проекта осуществлялось при поддержке компании Microsoft.