Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Образовательный комплекс

Введение в методы параллельного программирования

Раздел 7.

Параллельные методы умножения матрицы на вектор

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

Содержание

- 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
- Заключение

Введение

- Матрицы и матричные операции широко используются в разных областях приложений науки и техники
- Матричные вычисления требуют выполнения вычислительно-трудоемких расчетов
- Матричные операции предоставляют прекрасную возможность для демонстрации многих приемов и методов параллельного программирования

Матричные операции представляют собой классическую область применения параллельных вычислений

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***

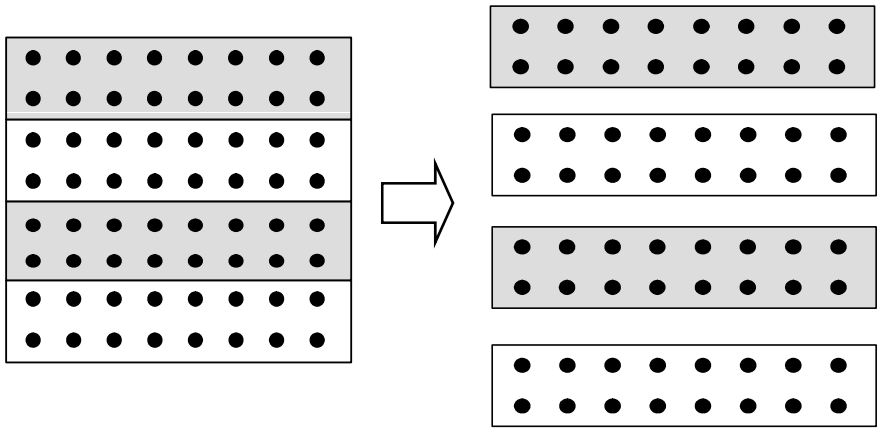
$$c_i = (a_i, b) = \sum_{j=1}^{n} a_{ij} b_j, \quad 0 \le i < m$$

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

Способы распределения данных: *ленточная схема*

Непрерывное (последовательное) распределение

горизонтальные полосы



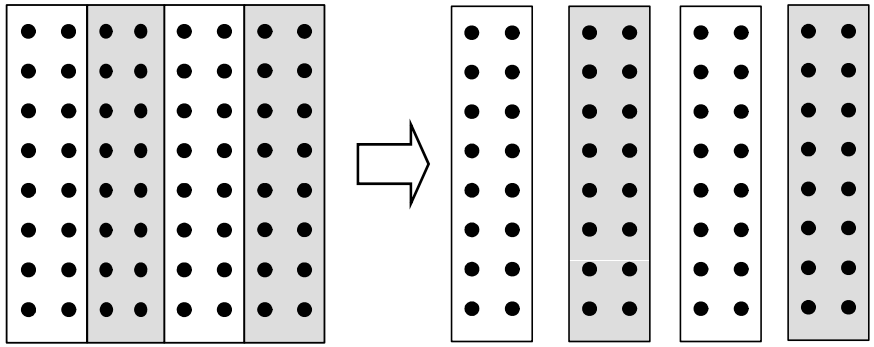
$$A = (A_0, A_1, ..., A_{p-1})^T,$$

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

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

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

вертикальные полосы



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

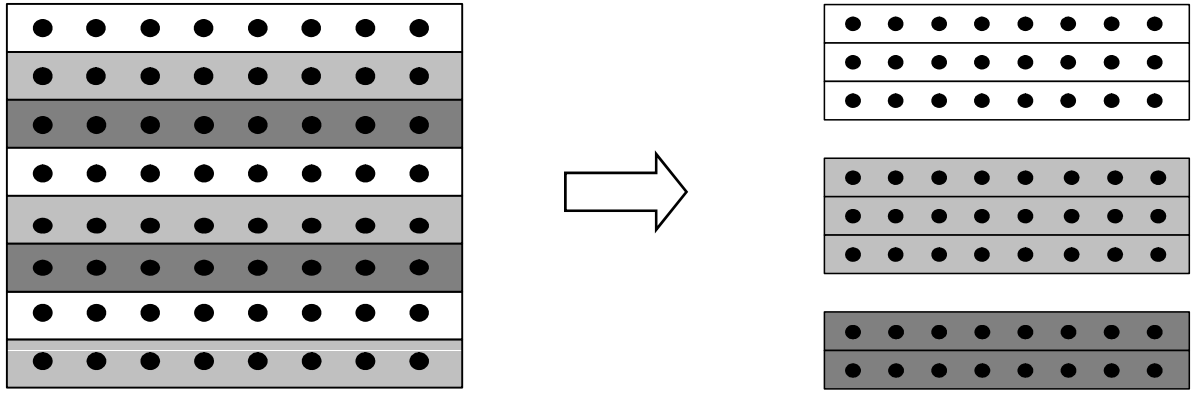
$$A_i = (\alpha_{i_0}, \alpha_{i_1}, ..., \alpha_{i_{k-1}}) \ ,$$

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

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

Способы распределения данных: *ленточная схема*

Чередующееся (цикличное) горизонтальное разбиение

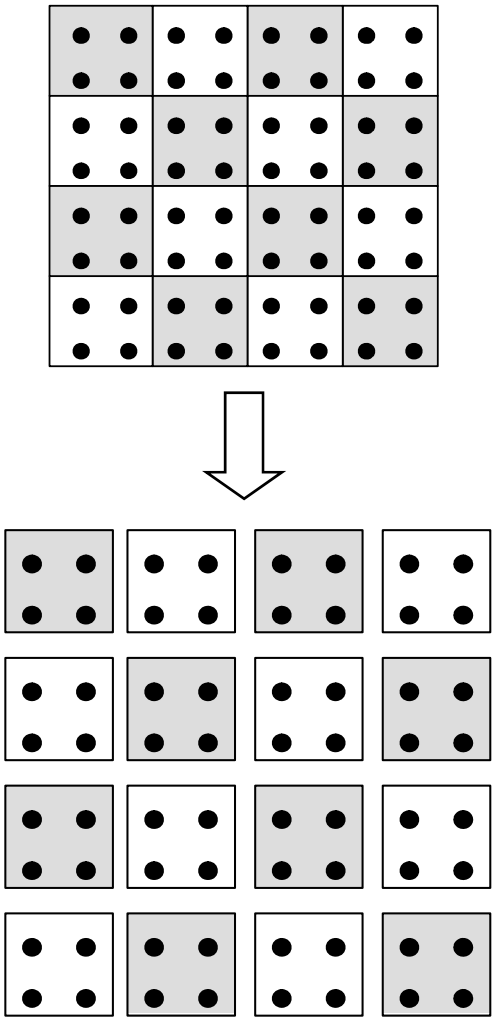


$$A = (A_0, A_2, ..., A_{p-1})^T,$$

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

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

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

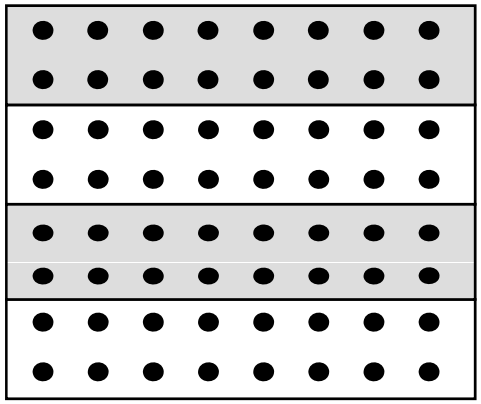
```
// Последовательный алгоритм умножения матрицы на вектор
for ( i = 0; i < m; i++ ) {
  c[i] = 0;
  for ( j = 0; j < n; j++ ) {
    c[i] += A[i][j]*b[j]
  }
}
```

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

При дальнейшем изложении материала для снижения сложности и упрощения получаемых соотношений будем предполагать, что матрица A является квадратной, т.е. m=n.

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

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

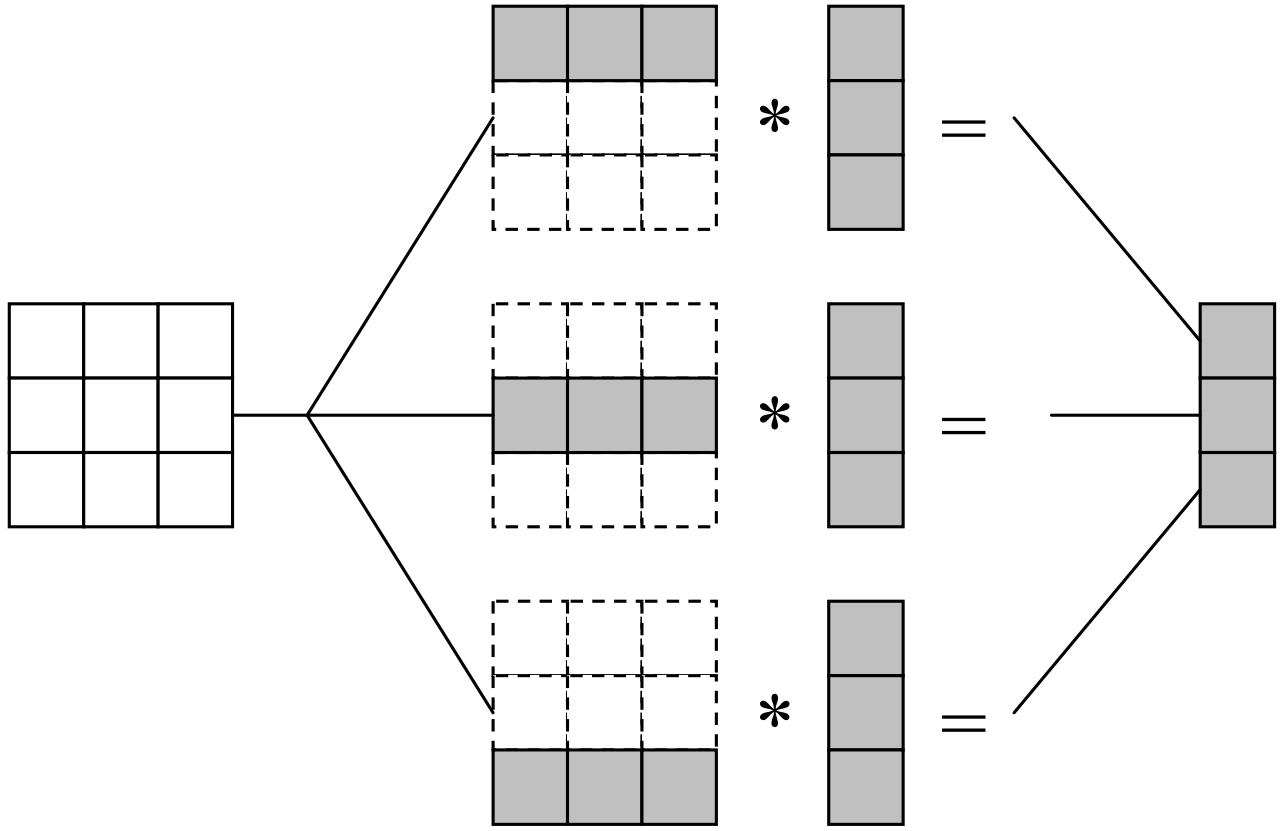


- **Базовая подзадача** - операция скалярного умножения одной строки матрицы на вектор

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

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

- Выделение информационных зависимостей

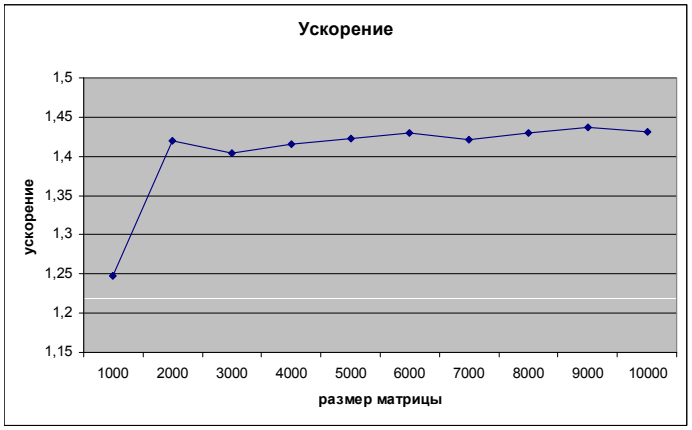


Алгоритм 1: ленточная схема (разбиение матрицы по строкам)…

- **Масштабирование и распределение подзадач по вычислительным элементам:**
  - Если число вычислительных элементов (процессоров и/или ядер) ***p*** меньше числа базовых подзадач ***m*** ***(p<m)***, базовые подзадачи могут быть укрупнены с тем, чтобы каждый вычислительный элемент 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***

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

- **Анализ эффективности:…**
  - Алгоритм обладает хорошей «локализацией вычислений», накладные расходы на организацию параллельности практически отсутствуют. Следовательно, можно ожидать линейного ускорения
  - Ускорение, которое демонстрирует алгоритм, далеко от линейного:



В чем причина недостаточного ускорения?

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

- **Анализ эффективности:…**
  - Вычислительная сложность алгоритма – $O(n^2)$
  - Объем обрабатываемых данных - $O(n^2)$

Время решения задачи складывается из времени вычислений и времени, которое тратится на чтение необходимых для вычислений данных из оперативной памяти в кэш.

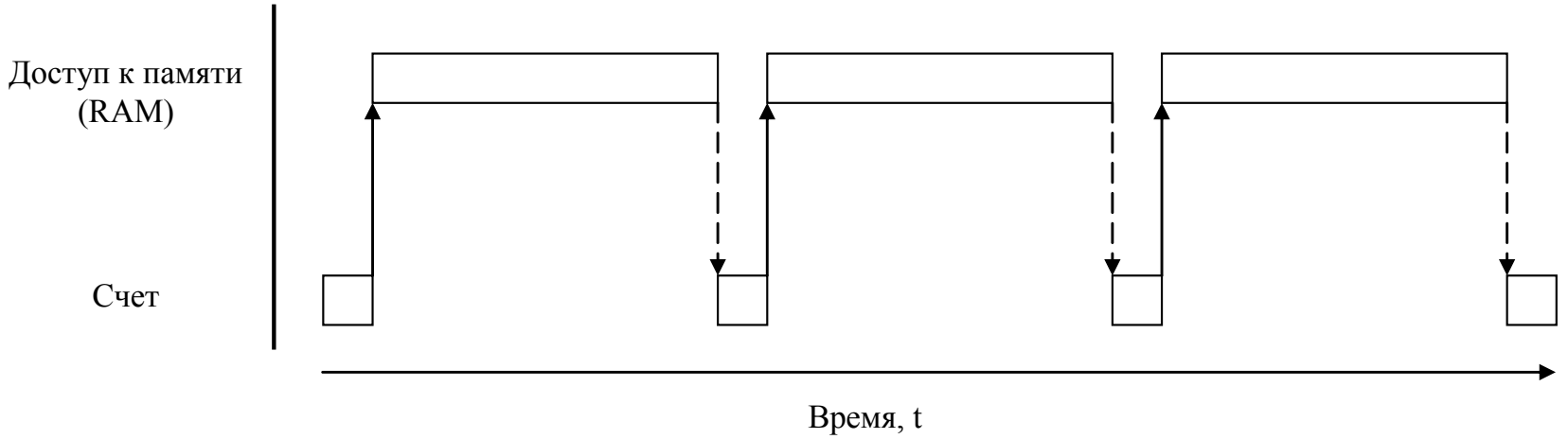
Время, необходимое на чтение данных, может быть сопоставимо или в несколько раз превосходить время счета.

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

- Анализ эффективности:…

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

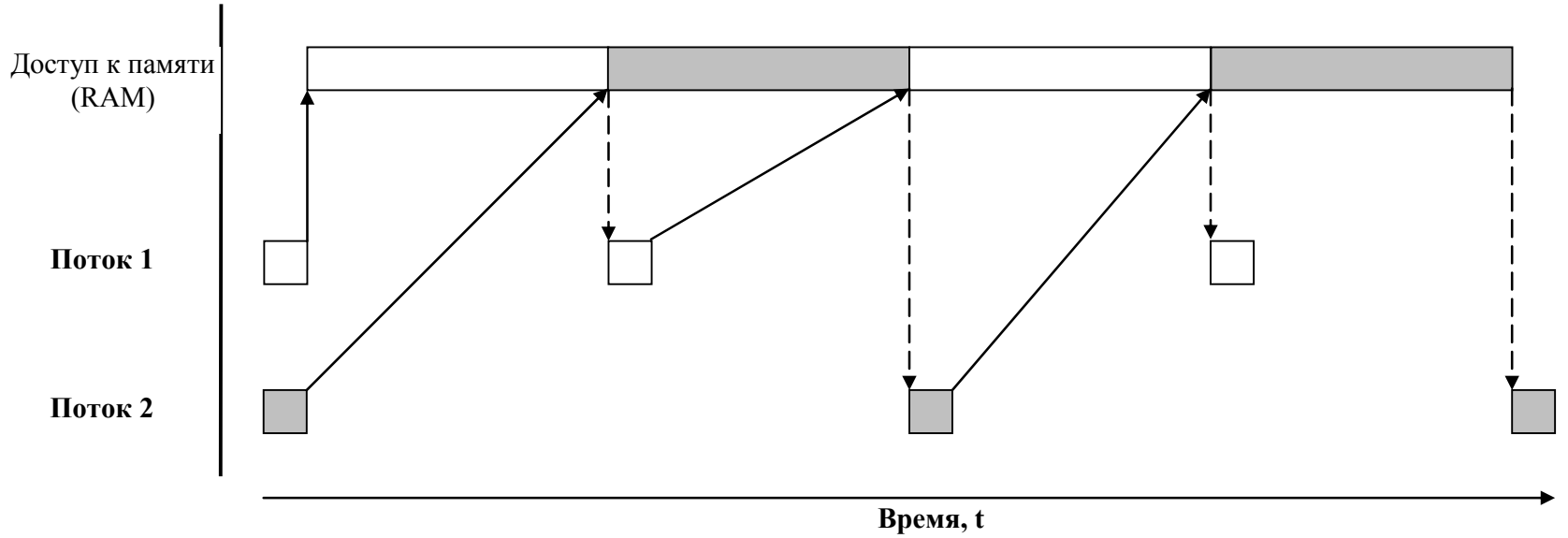
- **Анализ эффективности:…**
  - Процесс выполнения последовательного алгоритма умножения матрицы на вектор может быть представлен в виде диаграммы:



  - *Замечание*: В современных процессорах на аппаратном уровне часто реализуются алгоритмы предсказания потребности данных для обработки и часть данных может перемещаться в кэш заранее Тем самым обеспечивается совмещение обработки и подкачки данных)

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

- **Анализ эффективности:…**
  - Процесс выполнения параллельного алгоритма на многоядерных процессорах Intel может быть представлен при помощи диаграммы (ядра разделяют общий канал доступа к памяти):



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

- **Анализ эффективности:**
  - Время выполнения параллельного алгоритма может быть оценено при помощи следующего соотношения:

$$T_{mem} \le T_p \le T_{calc} + T_{mem}$$

  - Время вычислений: $T_{calc} = \frac{n(2n-1)}{p} \cdot \tau$
  - Время доступа к оперативной памяти: $T_{mem} = \frac{8 \cdot n^2}{\beta}$
  - Итоговое соотношение:

$$\frac{8 \cdot n^2}{\beta} \le T_p \le \frac{n(2n-1)}{p} \cdot \tau + \frac{8 \cdot n^2}{\beta}$$

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

- **Программная реализация:**
  - Функция **ParallelResultCalculation** – для реализации параллельного алгоритма умножения матрицы на вектор достаточно добавить одну директиву *parallel for* в последовательную реализацию*:*

```
// Function for calculating matrix-vector multiplication
void ParallelResultCalculation(double* pMatrix,
  double* pVector, double* pResult, int Size) {
  int i, j;  // Loop variables
#pragma omp paralell for private (j)
  for (i=0; i<Size; i++) {
    for (j=0; j<Size; j++)
      pResult[i] += pMatrix[i*Size+j]*pVector[j];
  }
}
```

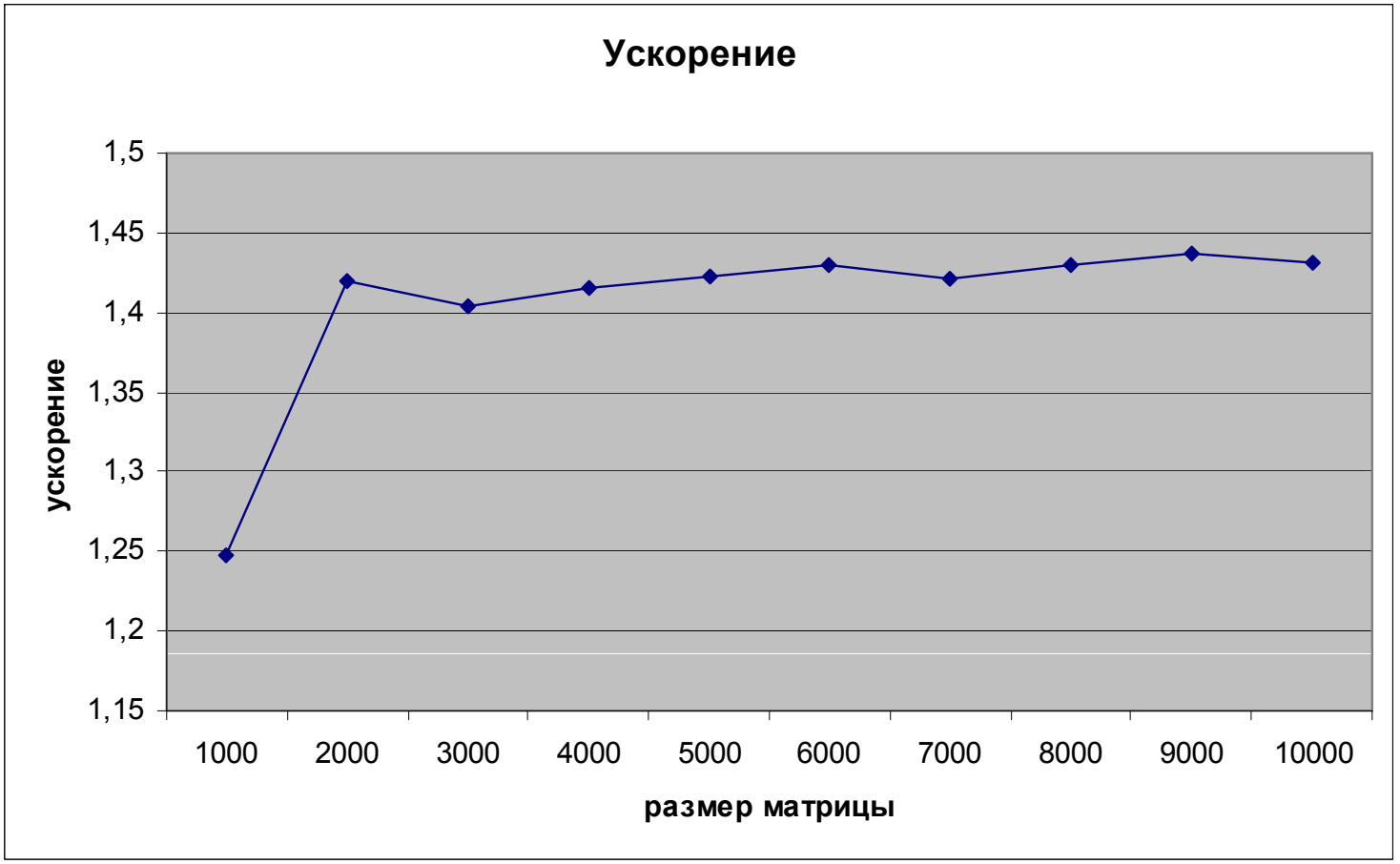
Алгоритм 1: ленточная схема (разбиение матрицы по строкам)…

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

Размер матрицы	Последовательный алгоритм	Параллельный алгоритм	
		Время	Ускорение
1000	0,0031	0,0024	1,2475
2000	0,0107	0,0075	1,4197
3000	0,0232	0,0165	1,4044
4000	0,0408	0,0288	1,4149
5000	0,0632	0,0445	1,4221
6000	0,0908	0,0635	1,4300
7000	0,1232	0,0867	1,4209
8000	0,1611	0,1127	1,4297
9000	0,2036	0,1416	1,4374
10000	0,2508	0,1753	1,4311

Алгоритм 1: ленточная схема (разбиение матрицы по строкам)

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



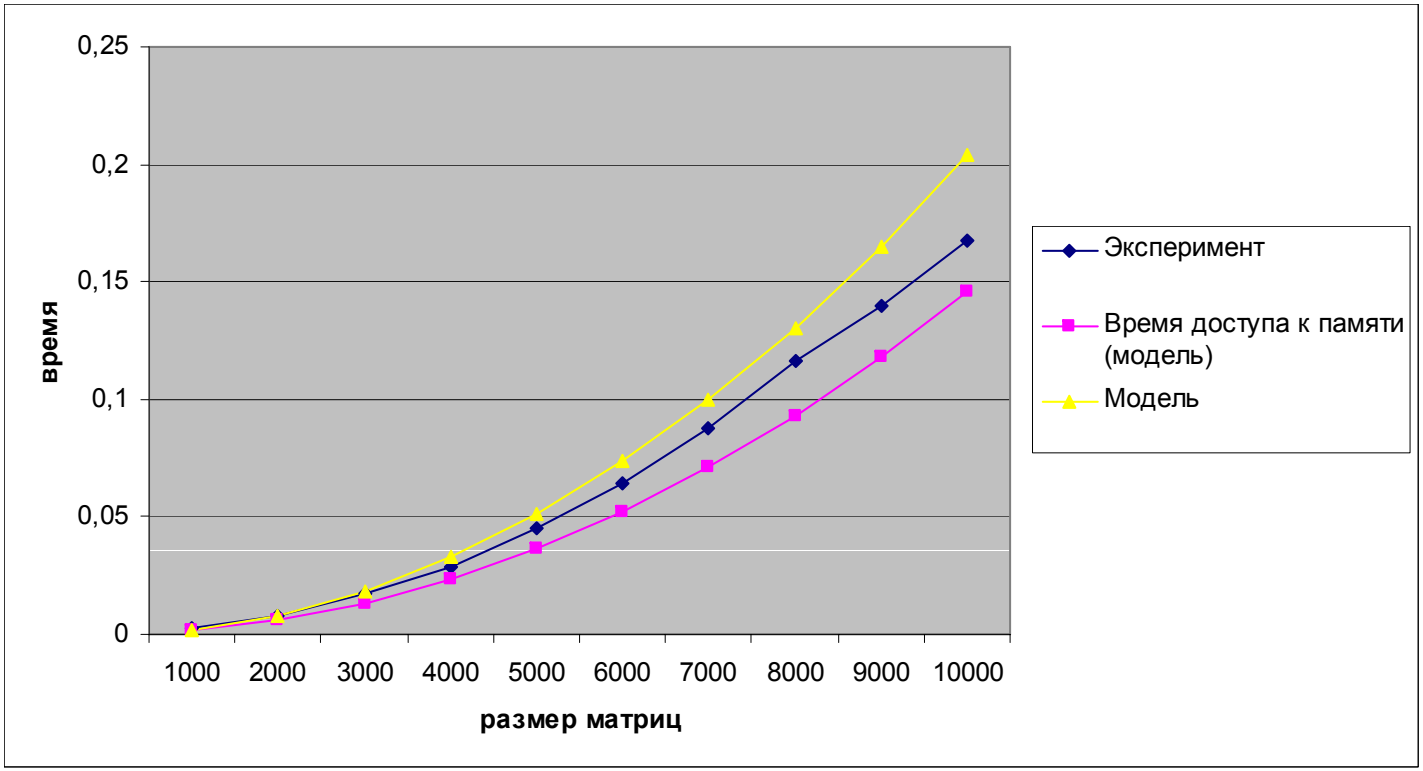
Алгоритм 1: ленточная схема (разбиение матрицы по строкам)…

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

Размер матриц	Эксперимент	Время счета (модель)	Время доступа к памяти (модель)	Общее время (модель)
1000	0,0031	5,86E-04	0,0015	0,0020
2000	0,0107	2,34E-03	0,0058	0,0082
3000	0,0232	5,27E-03	0,0131	0,0184
4000	0,0408	9,37E-03	0,0233	0,0327
5000	0,0632	1,46E-02	0,0364	0,0510
6000	0,0908	2,11E-02	0,0524	0,0735
7000	0,1232	2,87E-02	0,0713	0,0999
8000	0,1611	3,75E-02	0,0931	0,1306
9000	0,2036	4,75E-02	0,1178	0,1653
10000	0,2508	5,86E-02	0,1455	0,2041

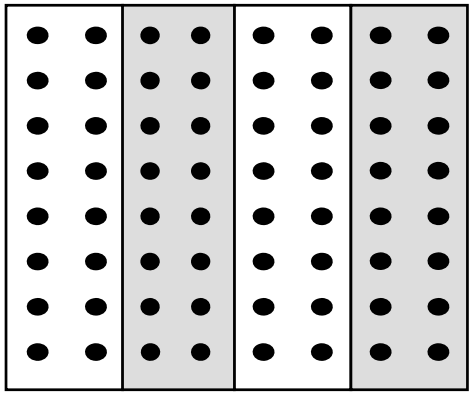
Алгоритм 1: ленточная схема (разбиение матрицы по строкам)

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

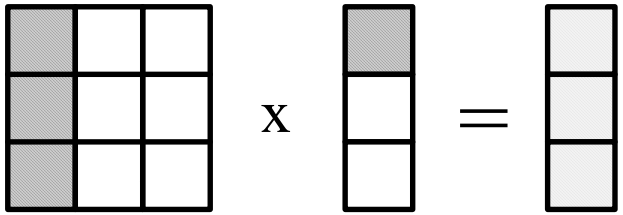


Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)…

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



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

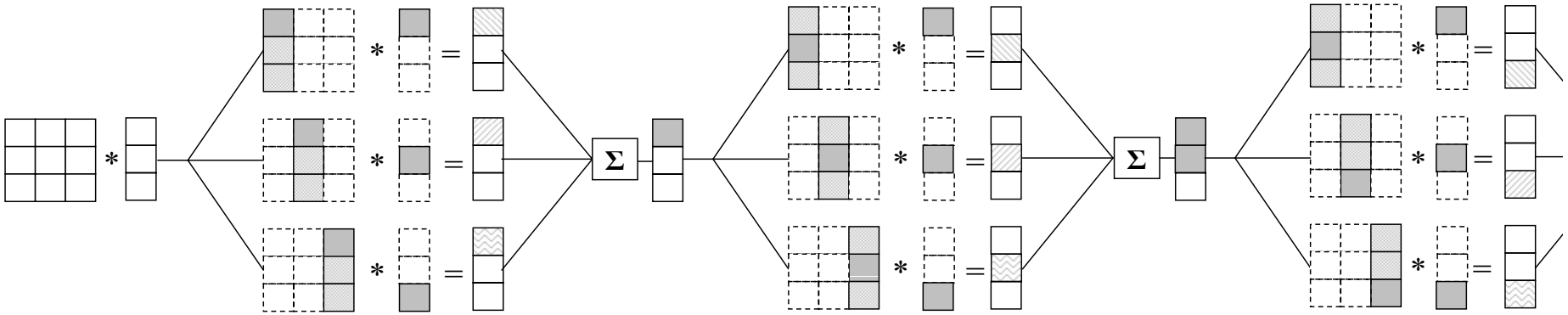


Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)…

- **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* необходимо просуммировать вектора *c'(i)*, которые были получены каждой подзадачей

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

❑ Схема информационного взаимодействия:



– Параллельные области создаются для вычисления каждого отдельного элемента результирующего вектора
– Программа представляется в виде набора последовательных (*однопотоковых*) и параллельных (*многопотоковых*) участков. Подобный принцип организации параллелизма получил название «*вилочного*» (*fork-join*) или *пульсирующего параллелизма*.

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

- **Масштабирование и распределение подзадач по вычислительным элементам:**
  - В случае, когда количество столбцов матрицы ***n*** превышает число вычислительных элементов ***p***, базовые подзадачи можно укрупнить, объединив в рамках одной подзадачи несколько соседних столбцов (в этом случае исходная матрица ***A*** разбивается на ряд вертикальных полос)

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

- **Анализ эффективности:…**
  - Выполняется больше арифметических операций:
    - На каждой итерации алгоритма после вычисления каждым потоком своей частичной суммы необходимо выполнить редукцию полученных результатов. Сложность выполнения операции редукции – $log_2 p$.
    - После выполнения редукции данных, необходимо запомнить результат в очередной элемент результирующего вектора.
  - Время выполнения вычислений:

$$T_{calc} = \left(\frac{n \cdot (2n-1)}{p} + n \cdot \log_2 p + n \right) \cdot \tau$$

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

- **Анализ эффективности:…**
  - На каждой итерации алгоритма организуется параллельная секция для вычисления каждого элемента результирующего вектора, следовательно, вызываются специализированные функции библиотеки OpenMP:
    - Для выполнения каждой такой функции необходимо определенное время,
    - В процессе выполнения эти функции используют данные, которые должны быть загружены в кэш процессора.
  - Величину накладных расходов на организацию параллельности на каждой итерации алгоритма обозначим δ

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

- **Анализ эффективности:**
  - Полное время выполнения параллельного алгоритма может быть вычислено по формуле:

$$T_p = \left(\frac{n \cdot (2n-1)}{p} + n \cdot \log_2 p + n \right) \cdot \tau + \frac{8 \cdot n^2}{\beta} + n \cdot \delta$$

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

- **Программная реализация:…**
  - Для реализации описанного подхода распараллелим внутренний цикл умножения матрицы на вектор:

```
// Function for calculating matrix-vector multiplication
void ParallelResultCalculation(double* pMatrix, double* pVector,
  double* pResult, int Size) {
  int i, j;  // Loop variables
  for (i=0; i<Size; i++) {
#pragma omp paralell for
    for (j=0; j<Size; j++) // Внимание – см. приводимые далее
                           // пояснения
      pResult[i] += pMatrix[i*Size+j]*pVector[j];
  }
}
```

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

- **Программная реализация:…**
  - Функция *TestResult* служит для проверки правильности выполнения умножения
    - выполняет умножение исходной матрицы на вектор при помощи последовательного алгоритма,
    - сравнивает поэлементно вектора, полученные в результате выполнения последовательного и параллельного алгоритмов

```
void TestResult(double* pMatrix, double* pVector, double* pResult,
  int Size);
```

  - Результаты экспериментов показывают, что разработанная функция параллельного умножения матрицы на вектор не всегда дает верный результат.

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

- **Программная реализация:…**
  - Реализация взаимоисключения доступа в общим данным с использованием *замков*:

```
void ParallelResultCalculation(double* pMatrix, double* pVector,
  double* pResult, int Size) {
  omp_lock_t MyLock;
  omp_init_lock(&MyLock);
  for (int i=0; i<Size; i++)
#pragma omp paralell for
    for (int j=0; j<Size; j++) {
      omp_set_lock(&MyLock);
      pResult[i] += pMatrix[i*Size+j]*pVector[j];
      omp_unset_lock(&MyLock);
    }
  omp_destroy_lock(&MyLock);
}
```

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

- **Программная реализация:…**
  - Регулирование доступа к элементу вектора *pResult[i]* привело к *сериализации* вычислений - потоки многопотокового участка могут выполняться только последовательно (!)
  - В OpenMP обеспечивается эффективное выполнение операции редукции при распараллеливании циклов (директивы *reduction*)

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

- **Программная реализация:**
  - Использование редукции:

```
// Function for calculating matrix-vector multiplication
void ParallelResultCalculation(double* pMatrix, double* pVector,
  double* pResult, int Size) {
  int i, j;  // Loop variables
  double IterGlobalSum = 0;
  for (i=0; i<Size; i++) {
    IterGlobalSum = 0;
#pragma omp parallel for reduction(+:IterGlobalSum)
    for (j=0; j<Size; j++)
      IterGlobalSum += pMatrix[i*Size+j]*pVector[j];
    pResult[i] = IterGlobalSum;
  }
}
```

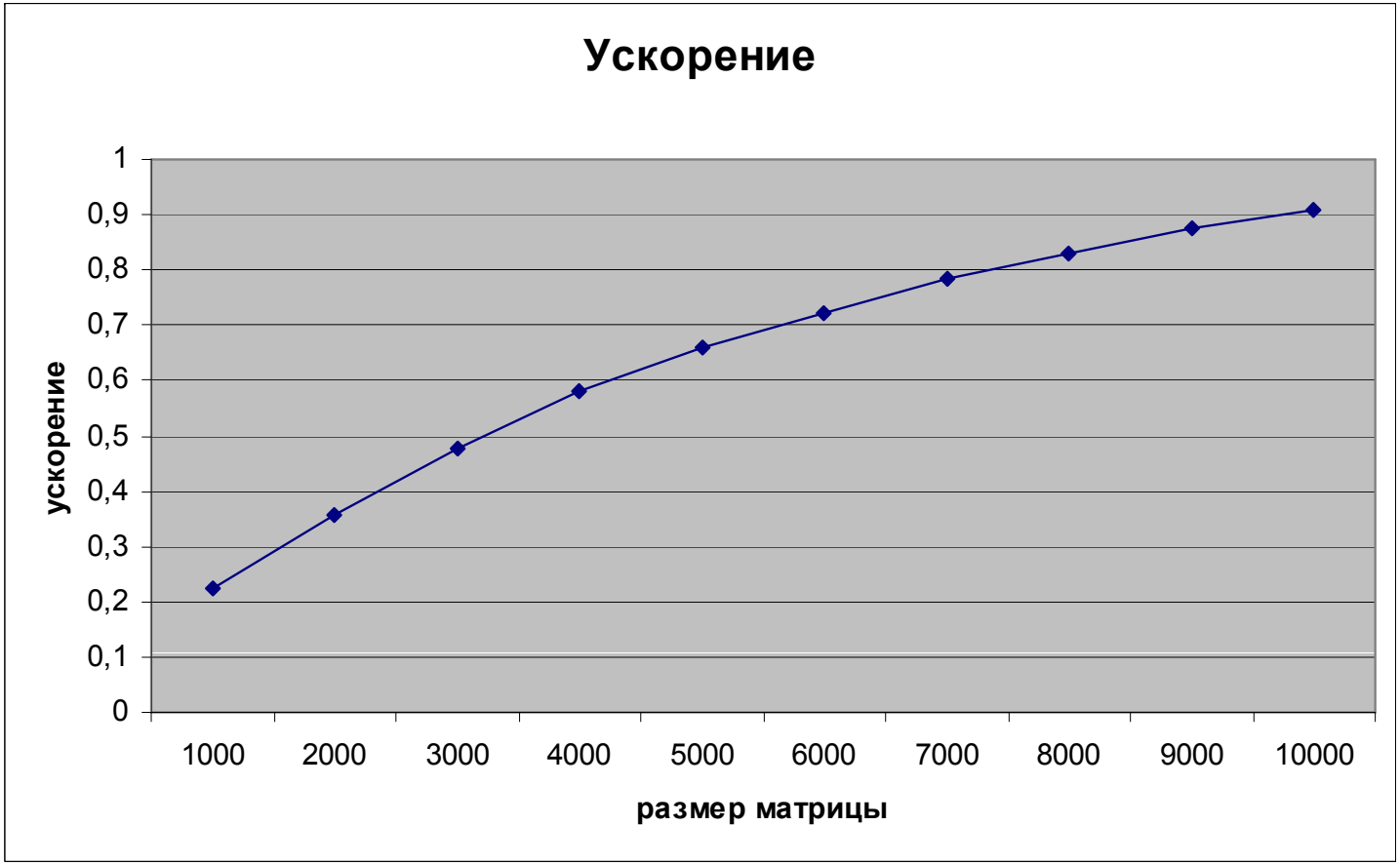
Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)…

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

Размер матрицы	Последовательный алгоритм	Параллельный алгоритм	
		Время	Ускорение
1000	0,0031	0,0136	0,2249
2000	0,0107	0,0298	0,3582
3000	0,0232	0,0484	0,4784
4000	0,0408	0,0704	0,5795
5000	0,0632	0,0959	0,6598
6000	0,0908	0,1255	0,7237
7000	0,1232	0,1574	0,7825
8000	0,1611	0,1939	0,8305
9000	0,2036	0,2326	0,8753
10000	0,2508	0,2764	0,9074

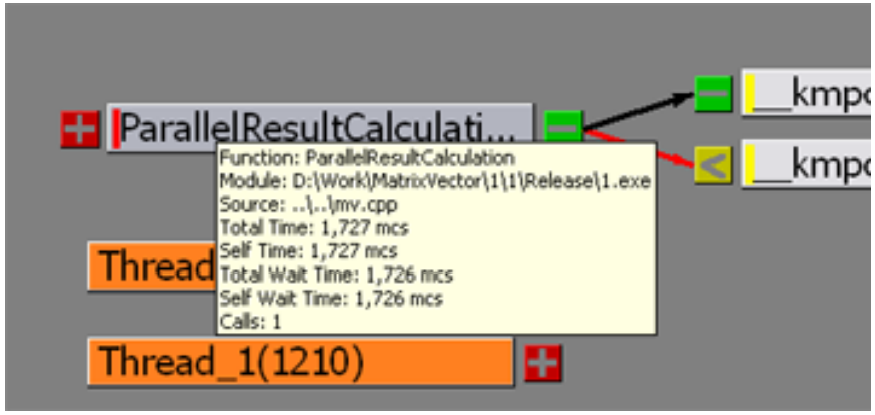
Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)…

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

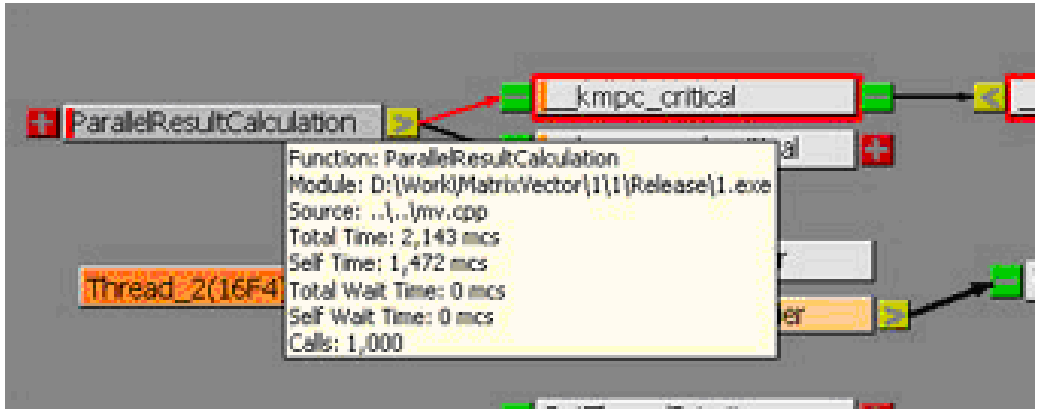


Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)…

- **Результаты вычислительных экспериментов:…**
  - Результаты профилирования при помощи инструмента *Call Graph (граф вызова функций)* профилировщика Intel VTune Performance Analyzer:



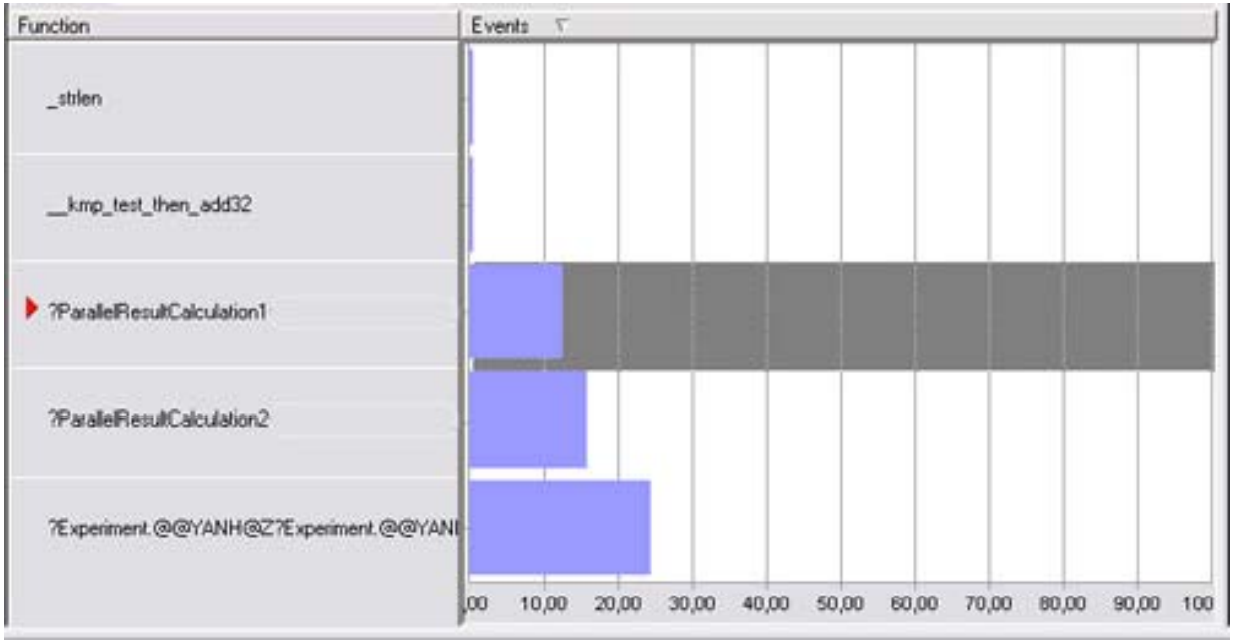
Результаты профилирования алгоритма, основанного на ленточном горизонтальном разделении данных



Результаты профилирования алгоритма, основанного на ленточном вертикальном разделении данных

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

- **Результаты вычислительных экспериментов:…**
  - Результаты профилирования при помощи инструмента *Event Sampling* – соотношение количества выделенных кеш-линий первого уровня для первого и второго алгоритмов



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

- **Результаты вычислительных экспериментов:…**
  - Оценим накладные расходы на организацию параллельности на каждой итерации алгоритма - подготовим для этого еще один вариант параллельного алгоритма умножения матрицы на вектор при разделении матрицы по столбцам, в котором:
    - Потоки самостоятельно определяют (используя идентификатор потока) блоки элементов матрицы и вектора для обработки и параллельно выполняют умножение своих столбцов матрицы на элементы вектора; результат умножения сохраняется в общем массиве *AllResults,*
    - После выполнения умножения элементы вектора *AllResults* суммируются для получения элементов результирующего вектора

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

- Результаты вычислительных экспериментов:…

```
void ParallelResultCalculation(double* pMatrix, double* pVector,
  double* pResult, double* AllResults, int Size) {
  int ThreadNum;
#pragma omp parallel shared (ThreadNum)
{
    ThreadNum = omp_get_num_threads();
    int ThreadID = omp_get_thread_num();
    int BlockSize = Size/ThreadNum;
    double IterResult;
    for (int i=0; i<Size; i++) {
      IterResult = 0;
      for (int j=0; j<BlockSize; j++)
        IterResult += pMatrix[i*Size+j+ThreadID*BlockSize] *
          pVector[j+ThreadID*BlockSize];
      AllResults[Size*ThreadID+i] = IterResult;
    }
  }
  for (int i=0; i<Size; i++)
    for (int j=0; j<ThreadNum; j++)
      pResult[i] += AllResults[j*Size+i];
}
```

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

- **Результаты вычислительных экспериментов:…**
  - Для того, чтобы оценить величину накладных расходов δ на организацию параллельности:
    - из времени выполнения исходного алгоритма вычтем время выполнения вновь разработанного алгоритма,
    - полученную разницу поделим на количество параллельных секций.
  - Эксперименты показывают, что величина δ равна 10 микросекунд ($1 \cdot 10^{-5}$ с).

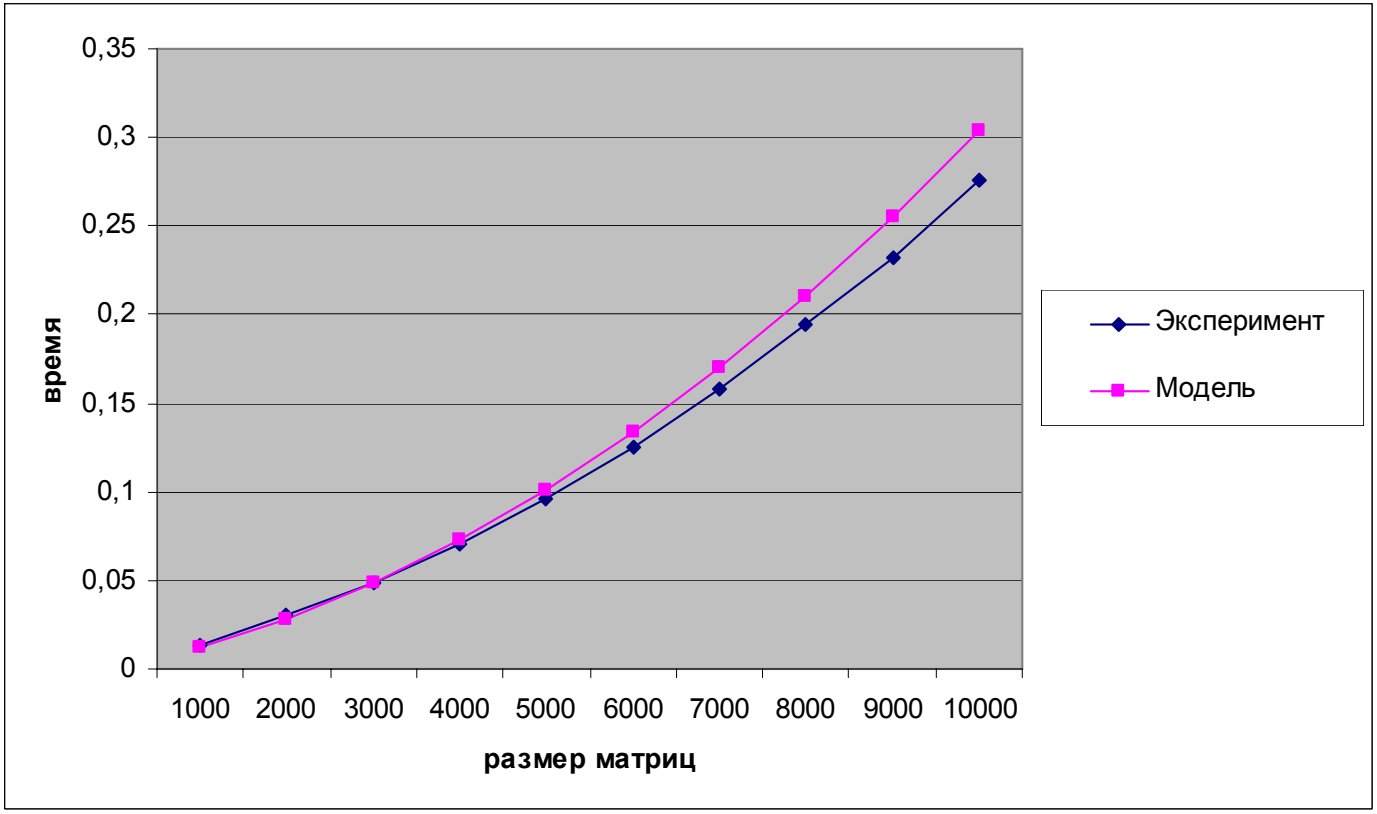
Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)…

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

Размер матриц	Эксперимент	Модель
1000	0,0136	0,0120
2000	0,0298	0,0282
3000	0,0484	0,0484
4000	0,0704	0,0727
5000	0,0959	0,1010
6000	0,1255	0,1335
7000	0,1574	0,1700
8000	0,1939	0,2106
9000	0,2326	0,2553
10000	0,2764	0,3041

Алгоритм 2: ленточная схема (разбиение матрицы по столбцам)

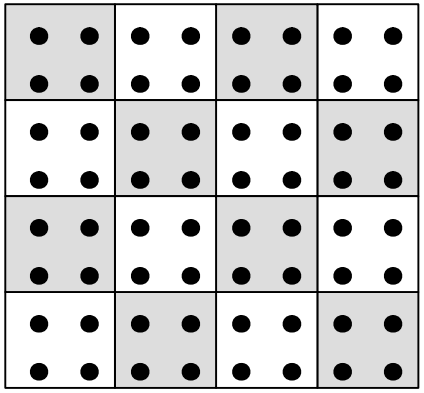
- **Результаты вычислительных экспериментов:**
  - Сравнение теоретических оценок и экспериментальных данных:



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)***,

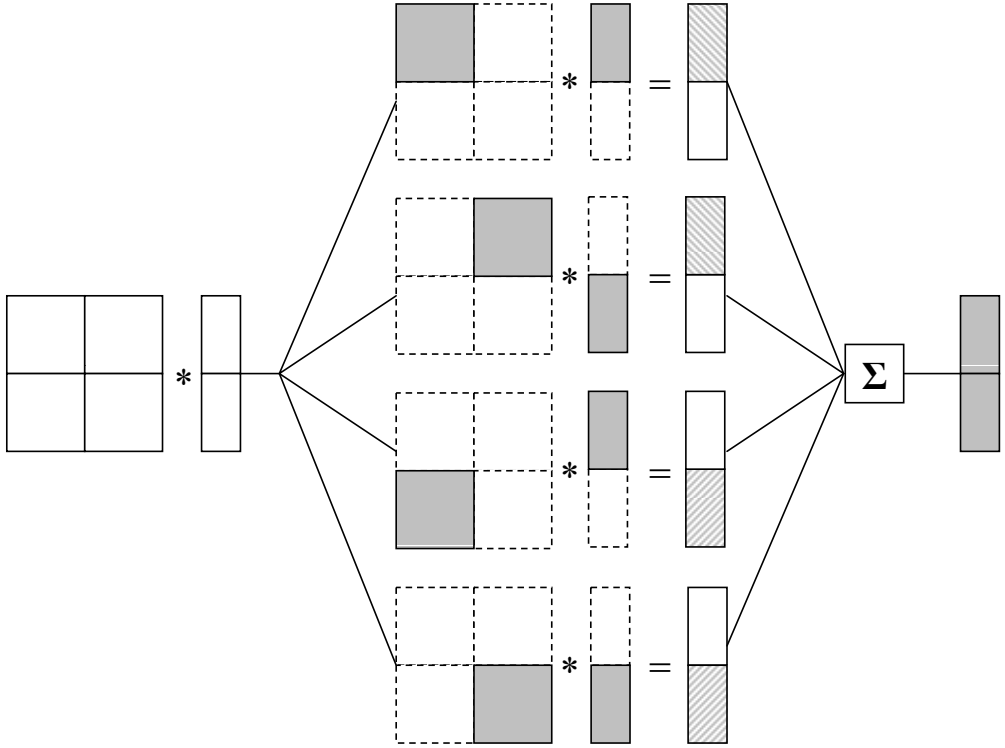
$$c'_v(i,j) = \sum_{u=0}^{l-1} a_{i_v j_u} b_{j_u},\ i_v = ik+v, 0 \le v < k, k = m/s,$$
$$j_u = jl+u, 0 \le u \le l, l = n/q$$

Алгоритм 3: блочная схема…

- **Базовая подзадача:**
  - Для снижения сложности и упрощения получаемых соотношений будем полагать, что число блоков в матрице A совпадает по горизонтали и по вертикали, т.е. $s=q$,
  - Для эффективного выполнения параллельного алгоритма умножения матрицы на вектор, основанного на блочном разделении данных, целесообразно выделить число параллельных потоков совпадающим с количеством блоков матрицы A, т.е. такое количество потоков, которое является полным квадратом q^2,
  - Число вычислительных элементов и количество потоков может различаться:
    - для обозначения числа потоков в дальнейшем будем использовать переменную π, т.е. $\pi = q^2$.

Алгоритм 3: блочная схема…

- **Схема информационного взаимодействия:**
  - В случае, когда для выполнения алгоритма создано 4 потока



Алгоритм 3: блочная схема…

- **Масштабирование и распределение подзадач по вычислительным элементам:**
  - Размер блоков матрицы ***A*** может быть подобран таким образом, чтобы общее количество базовых подзадач совпадало с числом параллельных потоков π, $\pi = q \cdot q$
  - Такой способ определения размера блоков приводит к тому, что объем вычислений в каждой подзадаче является равным и, тем самым, достигается полная балансировка вычислительной нагрузки между потоками

Алгоритм 3: блочная схема…

- **Анализ эффективности:…**
  - Первый вариант:
    - Каждый поток параллельной программы определяет блок матрицы и вектора
    - Умножение блоков выполняется потоками параллельно.
    - Элементы результирующего вектора вычисляются при помощи редукции полученных частичных результатов
  - Дополнительные вычислительные операции:
    - После того, как каждый поток выполнил умножение блока матрицы на блок вектора, необходимо сложить вектора частичных результатов для получения результирующего вектора. Таким образом, время вычислений можно определить по формуле:

$$T_{calc} = \left(\frac{n \cdot (2n-1)}{p} + n \cdot \pi \right) \cdot \tau$$

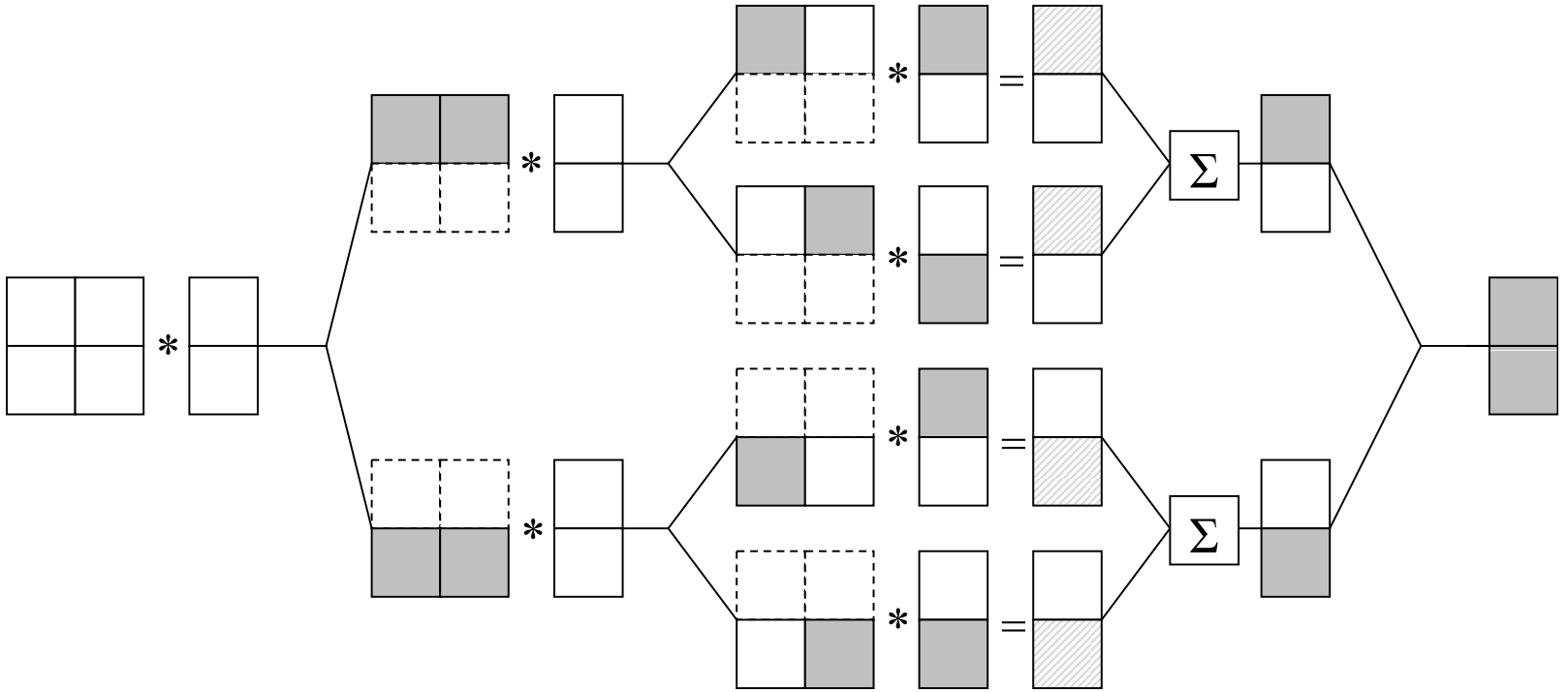
Алгоритм 3: блочная схема…

- **Анализ эффективности:…**
  - Первый вариант:
    - Для оценки полного времени выполнения параллельного алгоритма можно использовать все положения, использованные при выводе необходимых аналитических соотношений для параллельного алгоритма при ленточном горизонтальном разделении данных:

$$\frac{8 \cdot n^2}{\beta} \le T_p \le \left(\frac{n \cdot (2n-1)}{p} + n \cdot \pi \right) \cdot \tau + \frac{8 \cdot n^2}{\beta}$$

Алгоритм 3: блочная схема…

- **Анализ эффективности:…**
  - Второй вариант:
    - Использование вложенных параллельных секций



Алгоритм 3: блочная схема…

- **Анализ эффективности:…**
  - Второй вариант:
    - Дополнительные вычислительные операции:
      - После вычисления частичного результата каждым параллельным потоком «внутренней» параллельной секции необходимо выполнить редукцию и записать полученный результат в соответствующий элемент результирующего вектора
      - Для выполнения задачи используется π параллельных потоков, $\pi=q\cdot q$, и их число может отличаться от количества имеющихся вычислительных элементов. С учетом данного обстоятельства можно определить, что время выполнения вычислений ограничено сверху величиной

$$T_{calc} = \left(\frac{n \cdot (2n-1)}{p} + n \cdot q \right) \cdot \tau$$

Алгоритм 3: блочная схема…

- **Анализ эффективности:…**
  - Второй вариант:
    - Дополнительные вычислительные операции:
      - В случае же, когда количества вычислительных элементов совпадает с числом потоков, т.е. $p=\pi$, оценка времени вычислений может быть получена более точно:

$$T_{calc} = \left(\frac{n \cdot (2n-1)}{p} + \frac{n}{q} \cdot \log_2 q + \frac{n}{q} \right) \cdot \tau$$

Алгоритм 3: блочная схема…

- **Анализ эффективности:**
  - Второй вариант:
    - Для организации параллельных вычислений используется механизм вложенного параллелизма:
      - «внутренние» параллельные секции создаются и закрываются,
      - дополнительное время тратится на синхронизацию и выполнение операции редукции.
    - Величина накладных расходов δ была оценена ранее
    - Время выполнения параллельного алгоритма может быть вычислено по формуле:

$$T_p = \left(\frac{n \cdot (2n-1)}{p} + \frac{n}{p} \cdot \log_2 q + \frac{n}{q} \right) \cdot \tau + \frac{8 \cdot n^2}{\beta} + \frac{n}{q} \cdot \delta$$

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

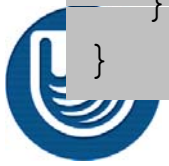
- **Программная реализация:…**
  - Первый вариант:
    - Для определения количества параллельных потоков используется функция *omp_set_num_threads*
    - Номер потока сохраняется в переменной *ThreadID*, блоки матрицы и вектора для обработки в потоке определяются в зависимости от значения переменной *ThreadID*
    - Для накопления результатов умножения блоков матрицы и вектора в каждом потоке используется локальная область памяти *pThreadResult* - для получения элемента результирующего вектора необходимо просуммировать элементы векторов *pThreadResult* со всех потоков
    - Для обеспечения контроля доступа к разделяемому ресурсу *pResult* используется механизм критических секций

Программа

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

- **Программная реализация:**
  - Второй вариант – использование вложенного параллелизма и механизма редукции

```
// Function for matrix-vector multiplication
void ResultCalculation(double* pMatrix, double* pVector,
  double* pResult, int Size) {
  int NestedThreadsNum = 2;
  omp_set_num_threads(NestedThreadsNum);
  omp_set_nested(true);
#pragma omp parallel for
  for (int i=0; i<Size; i++) {
    double ThreadResult = 0;
#pragma omp parallel for reduction(+:ThreadResult)
    for (int j=0; j<Size; j++)
      ThreadResult += pMatrix[i*Size+j]*pVector[j];
    pResult[i] = ThreadResult;
  }
}
```



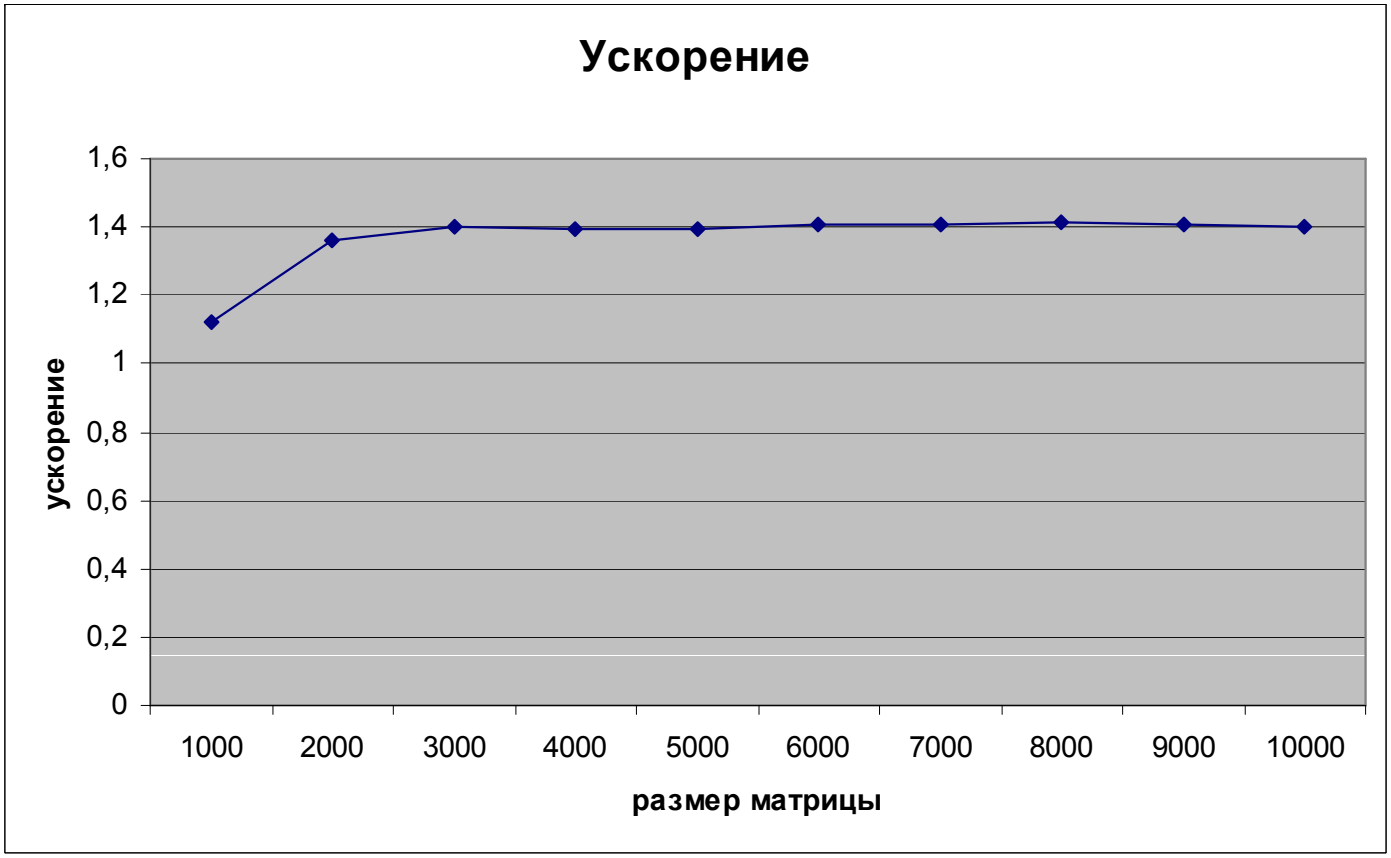
Алгоритм 3: блочная схема…

- **Результаты вычислительных экспериментов:…**
  - Первый вариант реализации – ускорение вычислений:…

Размер матрицы	Последовательный алгоритм	Параллельный алгоритм	
		Время	Ускорение
1000	0,0031	0,0027	1,1220
2000	0,0107	0,0078	1,3640
3000	0,0232	0,0165	1,4035
4000	0,0408	0,0292	1,3948
5000	0,0632	0,0454	1,3938
6000	0,0908	0,0646	1,4051
7000	0,1232	0,0877	1,4047
8000	0,1611	0,1141	1,4115
9000	0,2036	0,1446	1,4078
10000	0,2508	0,1787	1,4032

Алгоритм 3: блочная схема

- **Результаты вычислительных экспериментов:…**
  - Первый вариант реализации - ускорение вычислений:



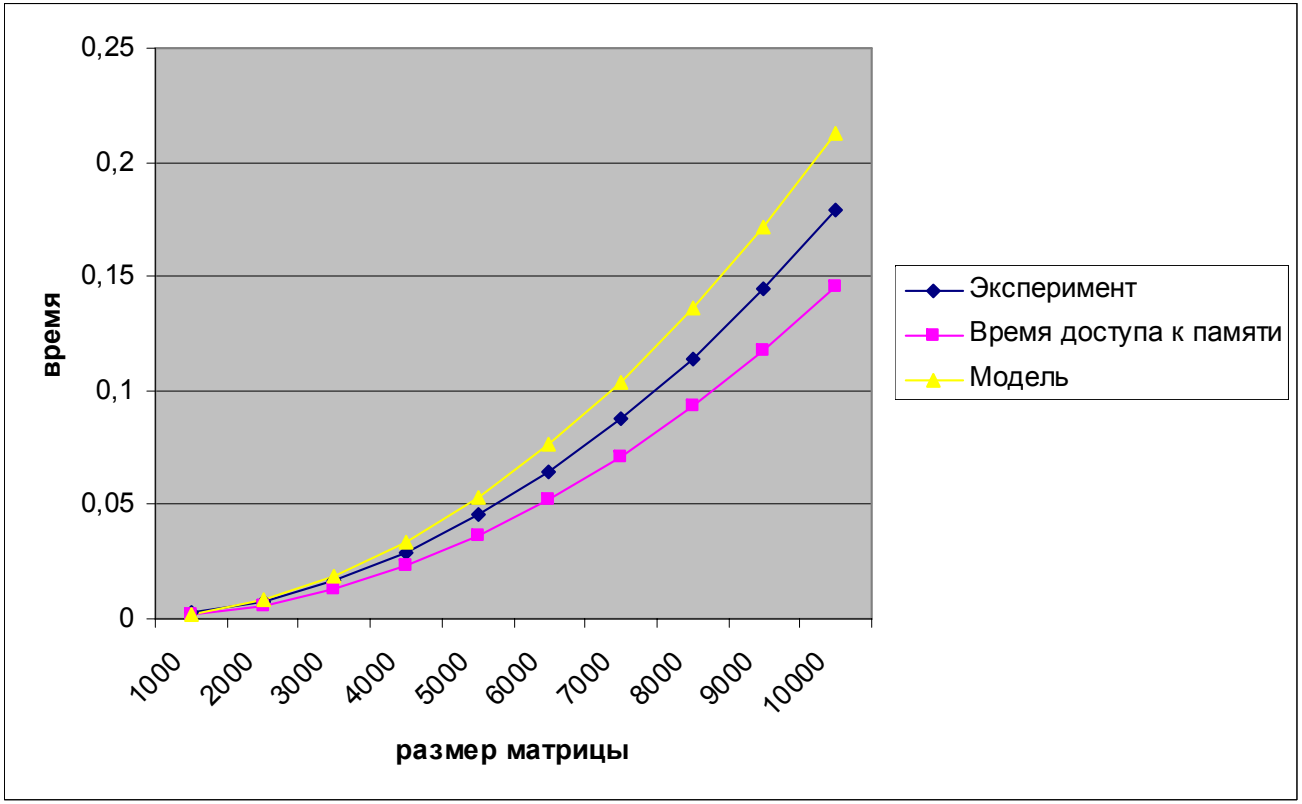
Алгоритм 3: блочная схема…

- **Результаты вычислительных экспериментов:...**
  - Первый вариант реализации – сравнение теоретических оценок и экспериментальных данных:…

Размер матриц	Эксперимент	Время счета (модель)	Время доступа к памяти (модель)	Общее время (модель)
1000	0,0027	0,0006	0,0015	0,0021
2000	0,0078	0,0023	0,0058	0,0085
3000	0,0165	0,0053	0,0131	0,0191
4000	0,0292	0,0094	0,0233	0,0340
5000	0,0454	0,0147	0,0364	0,0531
6000	0,0646	0,0211	0,0524	0,0764
7000	0,0877	0,0287	0,0713	0,1040
8000	0,1141	0,0375	0,0931	0,1358
9000	0,1446	0,0475	0,1178	0,1719
10000	0,1787	0,0586	0,1455	0,2122

Алгоритм 3: блочная схема…

- **Результаты вычислительных экспериментов:…**
  - Первый вариант реализации – сравнение теоретических оценок и экспериментальных данных:



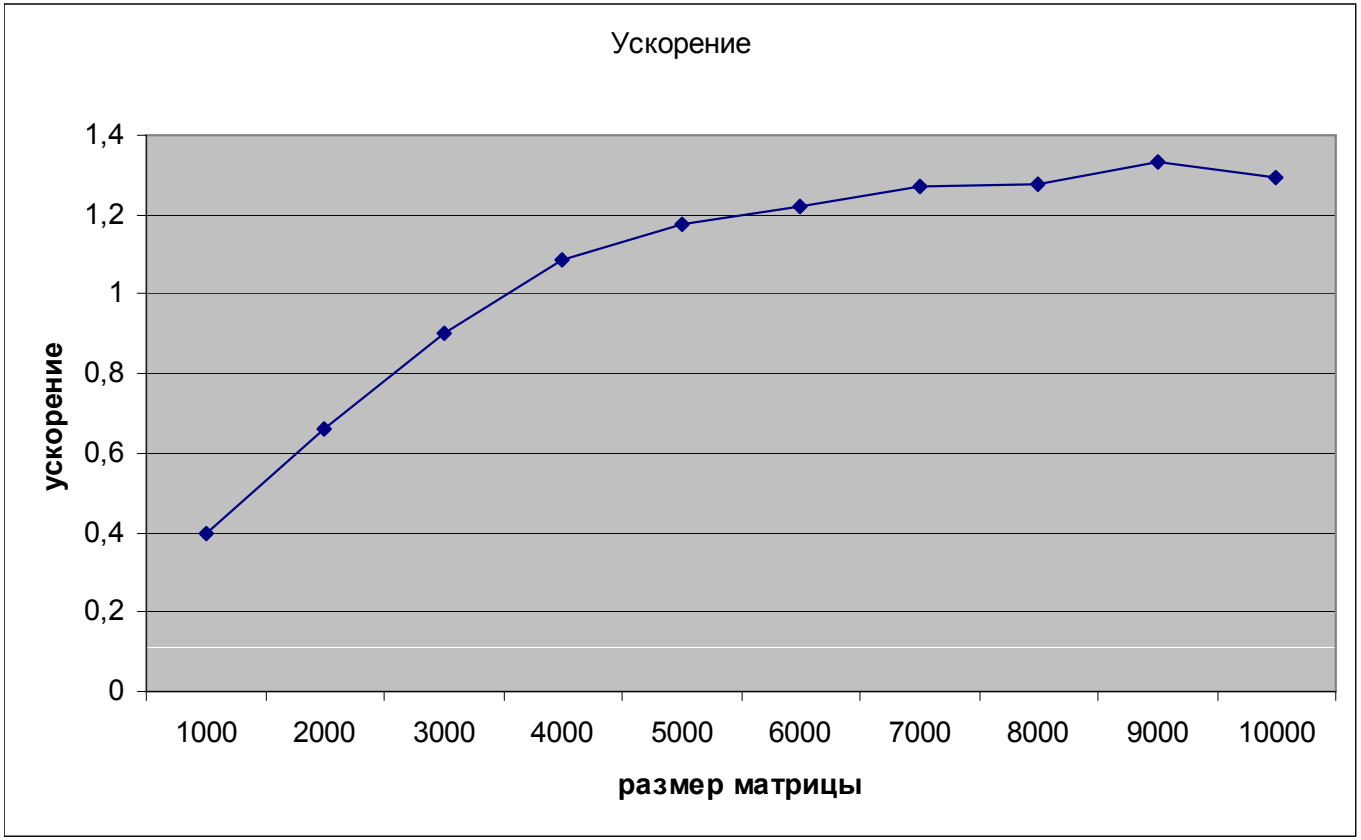
Алгоритм 3: блочная схема…

- **Результаты вычислительных экспериментов:…**
  - Второй вариант реализации – ускорение вычислений:…

Размер матрицы	Последовательный алгоритм	Параллельный алгоритм	
		Время	Ускорение
1000	0,0031	0,0077	0,3961
2000	0,0107	0,0161	0,6617
3000	0,0232	0,0257	0,9020
4000	0,0408	0,0374	1,0890
5000	0,0632	0,0539	1,1737
6000	0,0908	0,0742	1,2234
7000	0,1232	0,0968	1,2729
8000	0,1611	0,1260	1,2785
9000	0,2036	0,1528	1,3321
10000	0,2508	0,1937	1,2947

Алгоритм 3: блочная схема…

- **Результаты вычислительных экспериментов:…**
  - Второй вариант реализации - ускорение вычислений:



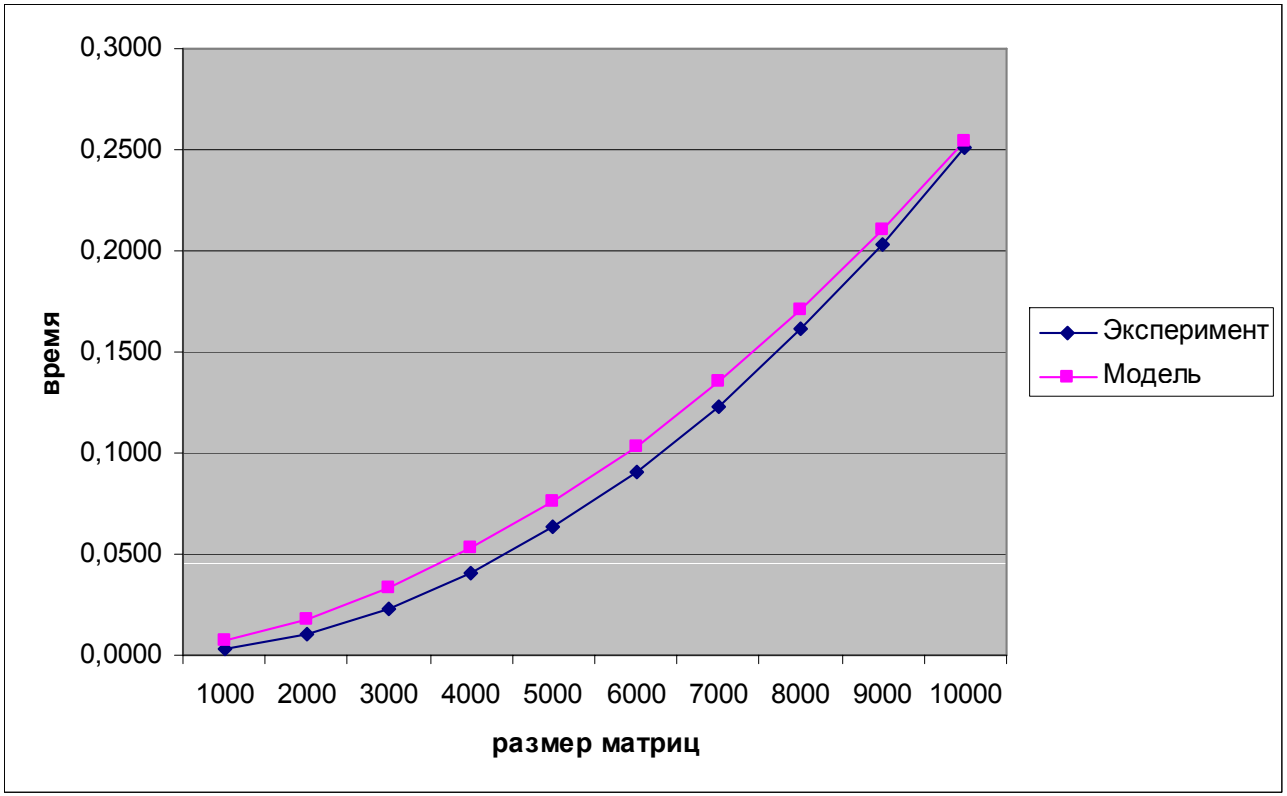
Алгоритм 3: блочная схема…

- **Результаты вычислительных экспериментов:…**
  - Второй вариант реализации – сравнение теоретических оценок и экспериментальных данных:…

Размер матриц	Эксперимент	Модель
1000	0,0031	0,0070
2000	0,0107	0,0182
3000	0,0232	0,0334
4000	0,0408	0,0526
5000	0,0632	0,0760
6000	0,0908	0,1035
7000	0,1232	0,1350
8000	0,1611	0,1706
9000	0,2036	0,2103
10000	0,2508	0,2541

Алгоритм 3: блочная схема…

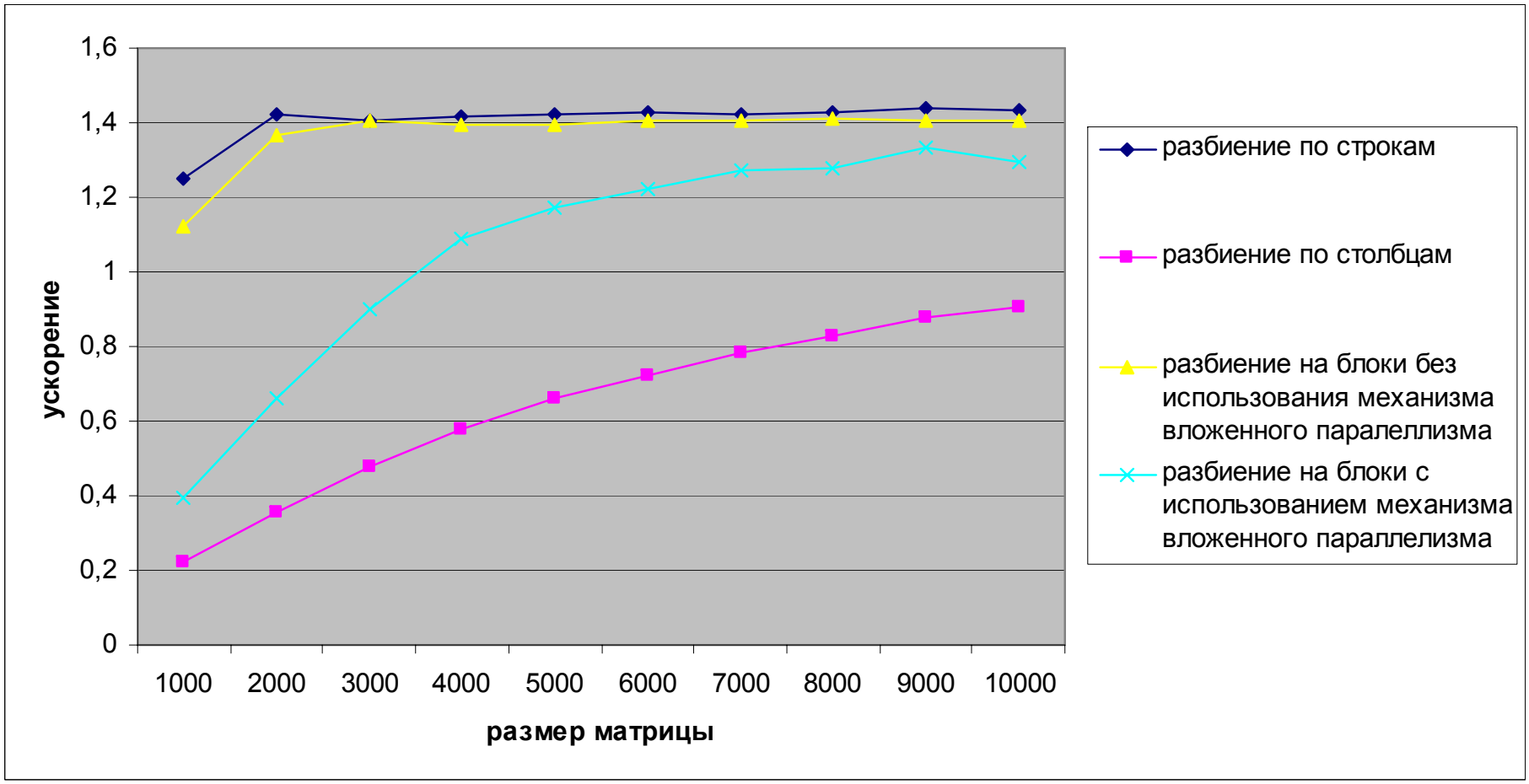
- **Результаты вычислительных экспериментов:**
  - Второй вариант реализации – сравнение теоретических оценок и экспериментальных данных:



Заключение

- Рассмотрены возможные схемы разделения данных между процессорами многопроцессорной вычислительной системы для параллельного выполнения матричных вычислений:
  - Ленточная схема (горизонтальное и вертикальное разбиение),
  - Блочная схема
- Описаны три возможных параллельных реализации одной из наиболее часто используемых матричных операций - умножения матрицы на вектор:
  - Алгоритм 1 – ленточное горизонтальное разбиение данных,
  - Алгоритм 2 – ленточное вертикальное разбиение данных,
  - Алгоритм 3 – блочное разбиение данных
- Теоретические оценки позволяют достаточно точно определить показатели эффективности параллельных вычислений

Сравнение алгоритмов



Вопросы для обсуждения

- Какой из алгоритмов обладает наилучшими показателями ускорения и эффективности?
- Может ли использование циклической схемы разделения данных повлиять на время работы каждого из представленных алгоритмов?

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.

Следующая тема

- **Параллельные методы матричного умножения**

Авторский коллектив

О проекте