Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Образовательный комплекс

Введение в методы параллельного программирования

Лабораторная работа 3.

Параллельные методы решения систем линейных уравнений

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

Содержание

Упражнения:

- Определение задачи решения системы линейных уравнений
- Изучение последовательного алгоритма Гаусса решения систем линейных уравнений
- Реализация последовательного алгоритма Гаусса
- Разработка параллельного алгоритма Гаусса
- Реализация параллельного алгоритма Гаусса решения систем линейных уравнений

Упражнение 1: Определение задачи решения системы линейных уравнений…

Линейное уравнение с *n* неизвестными

$$a_0 x_0 + a_1 x_1 + \ldots + a_{n-1} x_{n-1} = b$$

Множество *n* линейных уравнений называется *системой линейных уравнений* или *линейной системой*

$$\begin{aligned}
a_{0,0} x_0 \quad & + a_{0,1} x_1 \quad && + \ldots + a_{0,n-1} x_{n-1} \quad && = b_0 \\
a_{1,0} x_0 \quad & + a_{1,1} x_1 \quad && + \ldots + a_{1,n-1} x_{n-1} \quad && = b_1 \\
& \ldots \\
a_{n-1,0} x_0 \quad & + a_{n-1,1} x_1 \quad && + \ldots + a_{n-1,n-1} x_{n-1} \quad && = b_{n-1}
\end{aligned}$$

В матричной форме:

$$Ax = b$$

Упражнение 1: Определение задачи решения системы линейных уравнений

- Под *задачей решения системы линейных уравнений* для заданных матрицы *A* и вектора *b* понимается нахождение значения вектора неизвестных *x*, при котором выполняются все уравнения системы.

Упражнение 2: Изучение последовательного алгоритма Гаусса решения систем линейных уравнений…

- *Основная идея метода* - приведение матрицы *A* посредством эквивалентных преобразований к треугольному виду, после чего значения искомых неизвестных могут быть получены непосредственно в явном виде
- *Эквивалентные преобразования*:
  - Умножение любого из уравнений на ненулевую константу,
  - Перестановка уравнений,
  - Прибавление к уравнению любого другого уравнения системы.

Упражнение 2: Изучение последовательного алгоритма Гаусса решения систем линейных уравнений…

На первом этапе – *прямой ход* метода Гаусса – исходная система линейный уравнений при помощи последовательного исключения неизвестных приводится к верхнему треугольному виду

$$Ux = c, \qquad U = \begin{pmatrix} u_{0,0} & u_{0,1} & \dots & u_{0,n-1} \\ 0 & u_{1,1} & \dots & u_{1,n-1} \\ & & \dots & \\ 0 & 0 & \dots & u_{n-1,n-1} \end{pmatrix}$$

На *обратном ходе* метода Гаусса (второй этап алгоритма) осуществляется определение значений неизвестных

Упражнение 2: Изучение последовательного алгоритма Гаусса решения систем линейных уравнений…

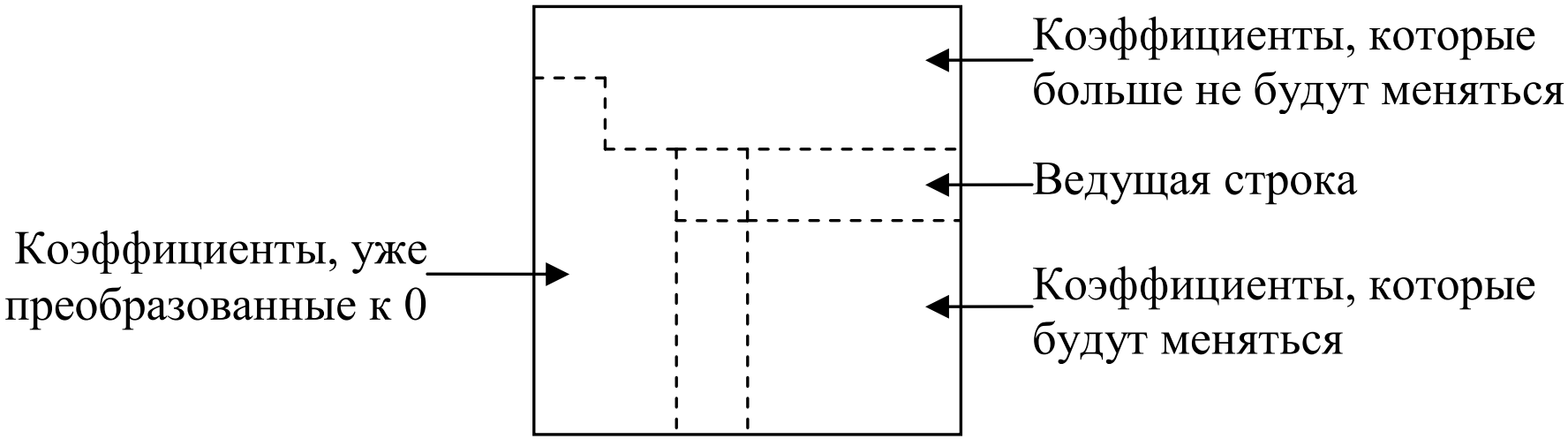
❑ Прямой ход

На итерации i, $0 \le i < n-1$, метода производится исключение неизвестной i для всех уравнений с номерами k, больших i (т.е. $i < k \le n-1$,). Для этого из этих уравнений осуществляется вычитание строки i, умноженной на константу (a_{ki}/a_{ii}) с тем, чтобы результирующий коэффициент при неизвестной x_i в строках оказался нулевым.
Все необходимые вычисления определяются при помощи соотношений:

$$a'_{kj} = a_{kj} - (a_{ki} / a_{ii}) \cdot a_{ij},$$
$$b'_{k} = b_{k} - (a_{ki} / a_{ii}) \cdot b_{i},$$
$$i \le j \le n-1, i < k \le n-1, 0 \le i < n-1$$

Упражнение 2: Изучение последовательного алгоритма Гаусса решения систем линейных уравнений…

- Общая схема состояния данных на *i*-ой итерации прямого хода алгоритма Гаусса



Упражнение 2: Изучение последовательного алгоритма Гаусса решения систем линейных уравнений

❑ Обратный ход

После приведения матрицы коэффициентов к верхнему треугольному виду становится возможным определение значений неизвестных:

- Из последнего уравнения преобразованной системы может быть вычислено значение переменной x_{n-1},
- Из предпоследнего уравнения становится возможным определение переменной x_{n-2} и т.д.

В общем виде, выполняемые вычисления при обратном ходе метода Гаусса могут быть представлены при помощи соотношений:

$$x_{n-1} = b_{n-1} / a_{n-1,n-1},$$

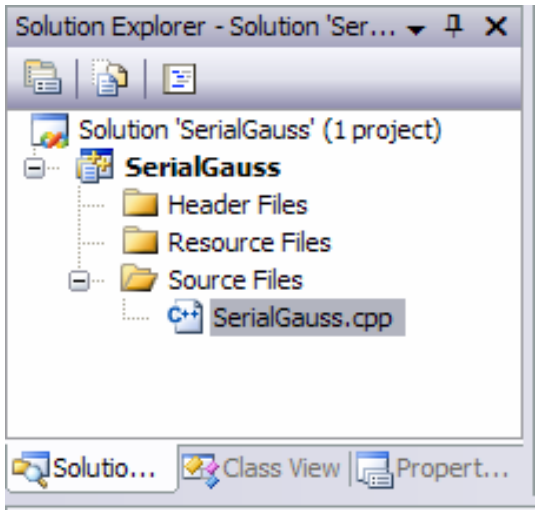
$$x_i = (b_i - \sum_{j=i+1}^{n-1} a_{ij} x_j) / a_{ii}, \quad i = n-2, n-3, \ldots, 0$$

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Поэтапная разработка последовательного алгоритма:**
  - Задание 1 – Открытие проекта **SerialGauss**
  - Задание 2 – Ввод размеров матрицы и вектора
  - Задание 3 – Ввод данных
  - Задание 4 – Завершение процесса вычислений
  - Задание 5 – Реализация прямого хода метода Гаусса
  - Задание 6 – Выполнение обратного хода алгоритма Гаусса
  - Задание 7 – Проведение вычислительных экспериментов

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 1** – Открытие проекта **SerialGauss:…**
  - Запустите приложение Microsoft Visual Studio 2005
  - Откройте проект **SerialGauss.sln**, расположенный в папке **C:\MsLabs\SerialGauss**
  - При помощи окна **Solution Explorer** откройте файл **SerialGauss.cpp**



Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 1** – Открытие проекта **SerialGauss:…**
  - Переменные, которые будут использоваться в программе:

```
double* pMatrix; // The matrix of linear system
double* pVector; // The right parts of the linear system
double* pResult; // The result vector
int Size;     // Sizes of the initial matrix and the vector
```

  - Вывод начального сообщения и ожидание нажатия любой клавиши перед завершением выполнения приложения:

```
printf("Serial Gauss algorithm for solving linear systems");
getch();
```

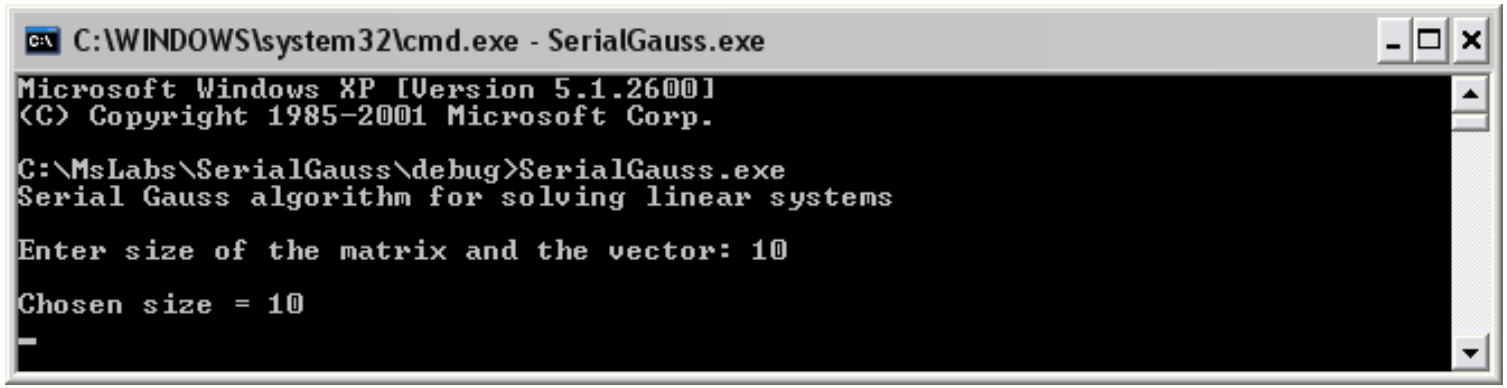
Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 2** – Ввод размеров матрицы и вектора:…
  - Функция *ProcessInitialization* для задания исходных данных:
    - Определяет размера матрицы и векторов,
    - Выделяет память для исходных матрицы *pMatrix* и вектора *pVector*, и вектора-результата *pResult*,
    - Определяет значения элементов исходных объектов

```
void ProcessInitialization (double* &pMatrix,
  double* &pVector, double* &pResult, int &Size);
где
-pMatrix – матрица линейной системы
-pVector – правая часть линейной системы
-pResult – искомый вектор неизвестных
-Size    – порядок матрицы и векторов
```

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 2** – Ввод размеров матрицы и вектора:
  - Реализуйте ввод размеров (задание значения переменной *Size*) матрицы и вектора, включая обработку возможных ошибочных ситуаций, внутри функции *ProcessInitialization*,
  - Добавьте вызов функции *ProcessInitialization* в главную функцию (функцию *main*) приложения,
  - Скомпилируйте и запустите приложение. Убедитесь в том, что размер объектов задается корректно



Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 3** – Ввод данных:…
  - Выделение памяти:

```
// Function for memory allocation and data initialization
void ProcessInitialization (double* &pMatrix,
  double* &pVector, double* &pResult, int &Size) {
  // Setting the size of the matrix and the vector
  <…>

 // Memory allocation
  pMatrix = new double [Size*Size];
  pVector = new double [Size];
  pResult = new double [Size];
}
```

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 3** – Ввод данных:…
  - Реализуйте функцию *DummyDataInitialization*, которая задавала бы значения элементов исходных по шаблону:

$$pMatrix = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \quad pVector = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

  - Интерфейс функции *DummyDataInitialization*:

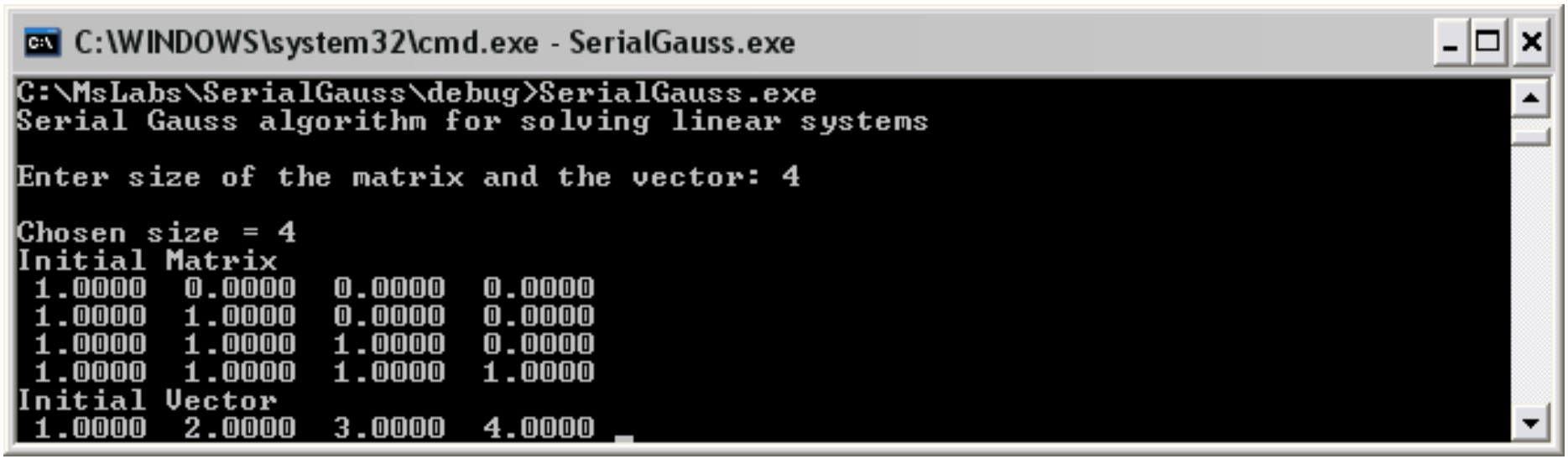
```
// Function for simple initialization of the matrix
// and the vector elements
void DummyDataInitialization (double* pMatrix,
  double* pVector, int Size);
```

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 3** – Ввод данных:…
  - Реализуйте функцию *DummyDataInitialization,*
  - Добавьте вызов функции *DummyDataInitialization* в функцию инициализации процесса вычислений *ProcessInitialization* для задания значений исходных матриц,
  - В главной функции приложения после вызова функции *ProcessInitialization* распечатайте исходную матрицу и вектор правых частей, используйте для печати функции форматированного вывода матрицы *PrintMatrix* и форматированного вывода вектора *PrintVector*, разработанные в лабораторной работе 1,
  - Скомпилируйте и запустите приложение. Убедитесь в том, что ввод данных осуществляется корректно.

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 3** – Ввод данных:



Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 4** – Завершение процесса вычислений:…
  - Функция *ProcessTermination* для завершения процесса вычислений:
    - Освобождает память, выделенную в ходе выполнения программы

```
// Function for computational process termination
void ProcessTermination (double* pMatrix, double* pVector,
  double* pResult) ,
где
pMatrix – матрица линейной системы
pVector – правая часть линейной системы
pResult – искомый вектор неизвестных
```

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 4** – Завершение процесса вычислений:
  - Реализуйте функцию *ProcessTermination*,
  - Добавьте вызов этой функции в главную функцию приложения,
  - Скомпилируйте и запустите приложение. Убедитесь, что программа выполняется корректно.

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 5** – Реализация прямого хода метода Гаусса:…
  - Для решения системы линейных уравнений при помощи последовательного алгоритма Гаусса реализуем функцию *SerialResultCalculation*:

```
// Function for the execution of Gauss algorithm
void SerialResultCalculation (double* pMatrix,
  double* pVector, double* pResult, int Size) {
  // Gaussian elimination
  SerialGaussianElimination (pMatrix, pVector, Size);
  // Back substitution
  SerialBackSubstitution (pMatrix, pVector, pResult, Size);
}
```

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 5** – Реализация прямого хода метода Гаусса:…
  - *pSerialPivotPos* - массив для запоминания порядка выбора ведущих строк
  - *pSerialPivotIter* – массив, в каждом *i*-ом элементе которого хранится номер итерации, на которой строка с номером *i* выбиралась в качестве ведущей.

```
int* pSerialPivotPos;  // The number of pivot rows selected
                       // at the iterations
int* pSerialPivotIter; // The iterations, at which the rows
                       // were pivots
```

  - Добавьте в код функции *SerialResultCalculation* операторы для выделения памяти для массивов *pSerialPivotPos* и *pSerialPivotIter,* присвойте элементам массива начальные значения.

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 5** – Реализация прямого хода метода Гаусса:…
  - На каждой итерации прямого хода метода Гаусса нужно определить ведущую строку в соответствие с **методом главных элементов**
  - Для выбора ведущей строки реализуйте функцию *FindPivotRow:*

```
// Finding the pivot row
int FindPivotRow(double* pMatrix, int Size, int Iter),
где
-pMatrix - матрица системы линейных уравнений,
-Size - размер матрицы,
-Iter – номер текущей итерации.
```

Упражнение 3: Реализация последовательного алгоритма Гаусса…

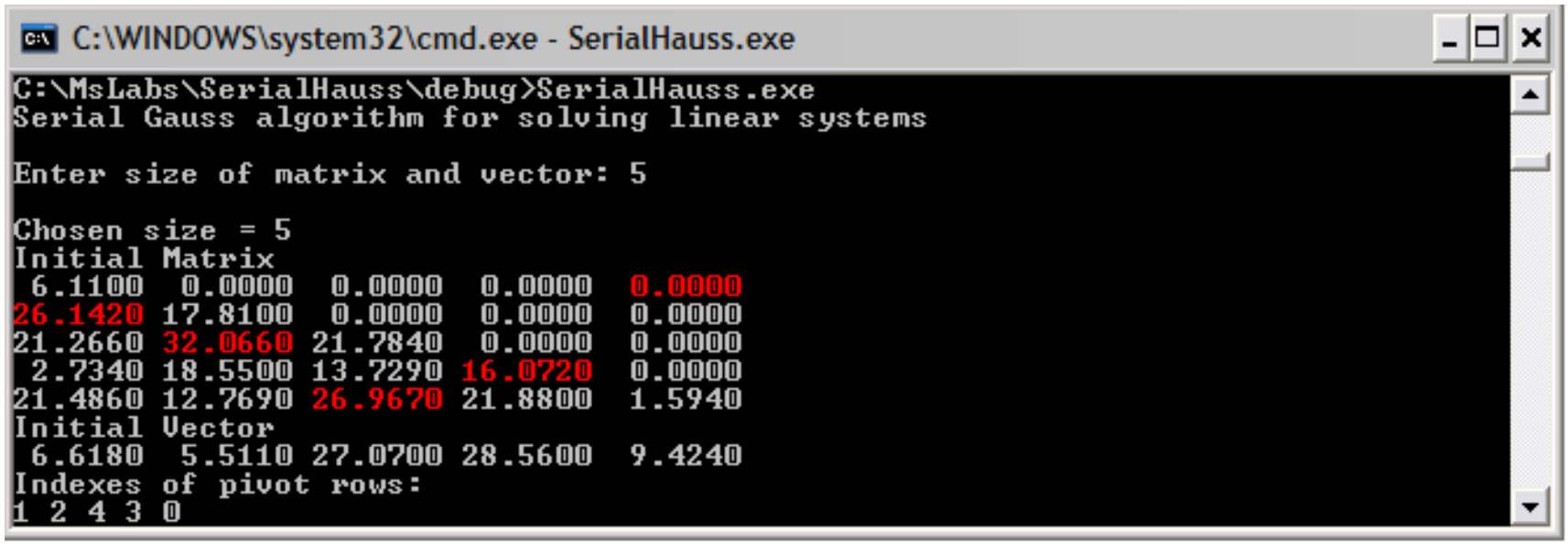
- **Задание 5** – Реализация прямого хода метода Гаусса:…
  - Функция, выполняющая прямой ход метода Гаусса:

```
// Gaussian elimination
void SerialGaussianElimination (double* pMatrix,
  double* pVector, int Size)
```

  - Добавьте вызов функции *FindPivotRow* в тело функции, *SerialGaussianElimination,* на каждой итерации присваивайте найденные значения элементам массивов *pSerialPivotPos* и *pSerialPivotIter*,
  - Распечатайте номера выбираемых ведущих строк для проверки правильности вычислений,
  - Закомментируйте вызов функции *SerialBackSubstitution.*

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 5** – Реализация прямого хода метода Гаусса:…
  - Добавьте вызов функции *SerialResultCalculation* в основную функцию приложения,
  - Скомпилируйте и запустите приложение, убедитесь в том, что ведущие строки выбираются верно:



Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 5** – Реализация прямого хода метода Гаусса:…
  - После выбора ведущей строки производится вычитание этой строки из всех строк, которые не были выбраны в качестве ведущих на предыдущих итерациях,
  - Для выполнения этих действий реализуем функцию *SerialColumnElimination*:

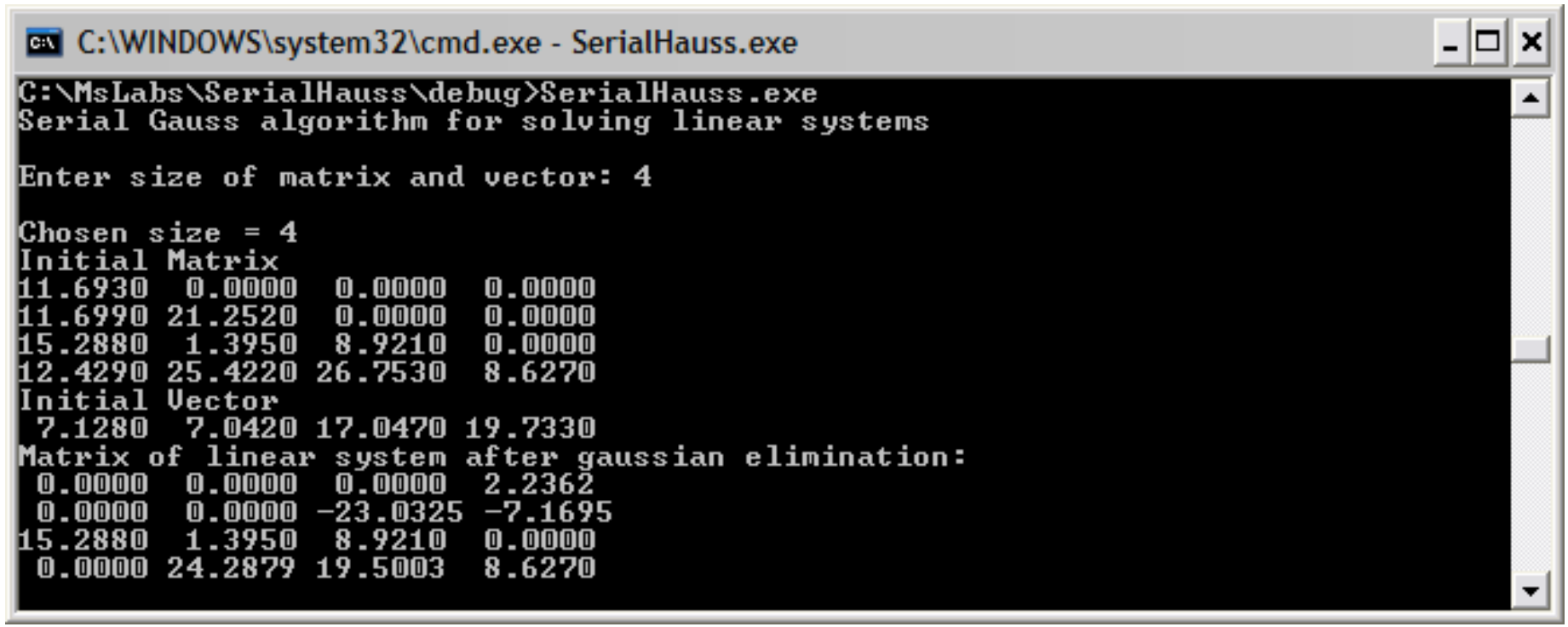
```
// Column elimination
void SerialColumnElimination (double* pMatrix,
  double* pVector, int Pivot, int Iter, int Size), где
-pMatrix – матрица линейной системы,
-pVector – вектор правых частей,
-Pivot – индекс ведущей строки,
-Iter – номер текущей итерации,
-Size – порядок системы уравнений.
```

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 5** – Реализация прямого хода метода Гаусса:…
  - Реализуйте функцию *SerialColumnElimination*,
  - Реализуйте вызов функции *SerialColumnElimination* на каждой итерации прямого хода метода Гаусса,
  - После выполнения всех итераций прямого хода распечатайте матрицу линейной системы при помощи функции *PrintMatrix*,
  - Скомпилируйте и запустите приложение, убедитесь в том, что после выполнения прямого хода метода Гаусса, матрица приводится к верхне-треугольному виду (с точностью до перестановки строк).

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 5** – Реализация прямого хода метода Гаусса:



Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 6** – Выполнение обратного хода алгоритма Гаусса:…
  - Реализуйте функцию *SerialBackSubstitution*, выполняющую обратный ход метода Гаусса в соответствии с алгоритмом, изложенным в упражнении 2:

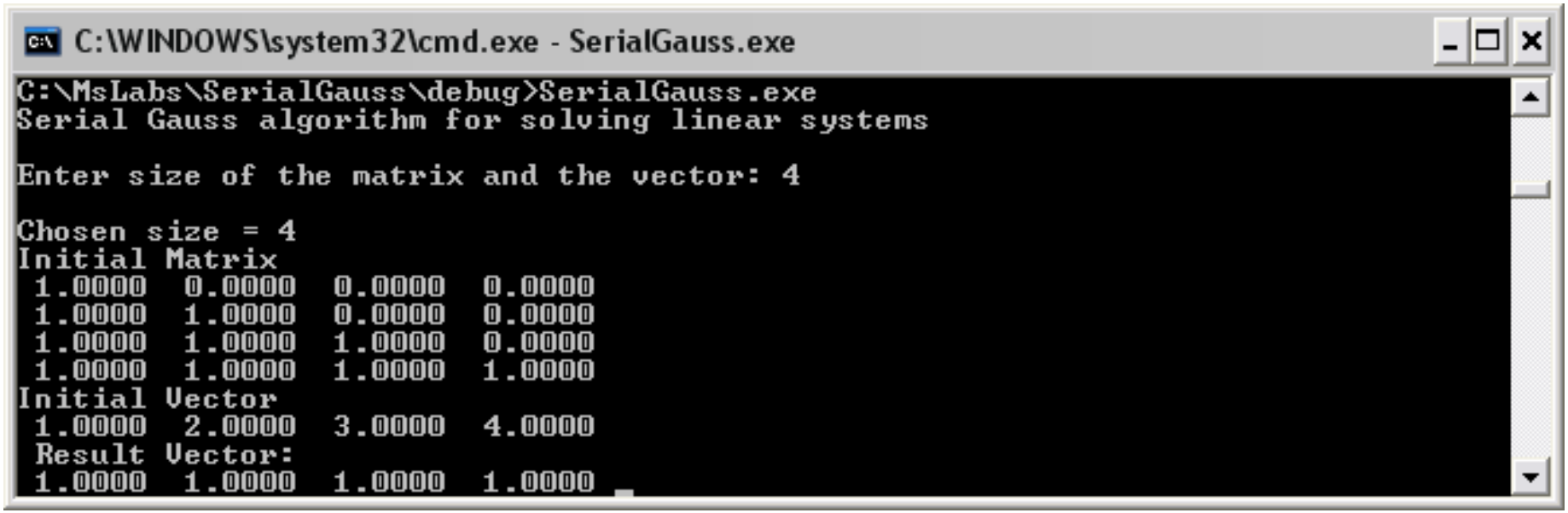
```
// Back substution
void SerialBackSubstitution (double* pMatrix,
  double* pVector, double* pResult, int Size), где
-pMatrix – матрица линейной системы,
-pVector – вектор правых частей,
-pResult – искомый вектор неизвестных,
-Size – порядок системы уравнений.
```

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 6** – Выполнение обратного хода алгоритма Гаусса:…
  - Добавьте вызов функции *SerialBackSubstitution* в функцию *SerialResultCalculation*,
  - Распечатайте матрицу линейной системы и результирующий вектор после выполнения метода Гаусса (если элементы исходной системы были заданы при помощи функции *DummyDataInitialization*, то все элементы результирующего вектора должны быть равны 1),
  - Скомпилируйте и запустите приложение. Убедитесь в его работоспособности.

Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 6** – Выполнение обратного хода алгоритма Гаусса:



Упражнение 3: Реализация последовательного алгоритма Гаусса…

- **Задание 7** – Проведение вычислительных экспериментов:…
  - Функция для задания значений элементов матрицы линейной системы и вектора правых частей при помощи датчика случайных чисел (чтобы гарантировать существование решения, генерируется нижне-треугольная матрица системы линейных уравнений):

```
// Function for random initialization of the matrix elements
void RandomDataInitialization (double* pMatrix,
  double* pVector, int Size);
```

  - Замените вызов функции *DummyDataInitialization* на *RandomDataInitialization* в функции *ProcessInitialization*,
  - Добавьте вычисление и вывод времени выполнения последовательного алгоритма Гаусса,
  - Протестируйте работоспособность приложения

Упражнение 3: Реализация последовательного алгоритма Гаусса

- **Задание 7** – Проведение вычислительных экспериментов:
  - Измерьте времена работы последовательного алгоритма Гаусса решения линейных систем при различных размерах матрицы,
  - Рассчитайте теоретическое время выполнения алгоритма,
  - Занесите результаты измерений и вычислений в таблицу.

Упражнение 4: Разработка параллельного алгоритма Гаусса…

- **Определение подзадач**
  - Все вычисления сводятся к однотипным вычислительным операциям над строками матрицы коэффициентов системы линейных уравнений,
  - Следовательно, в основу параллельной реализации алгоритма Гаусса может быть положен принцип распараллеливания по данным,
  - В качестве *базовой подзадачи* примем все вычисления, связанные с обработкой одной строки матрицы *A* и соответствующего элемента вектора *b*.

Упражнение 4: Разработка параллельного алгоритма Гаусса…

- **Выделение информационных зависимостей…**
  - Каждая итерация *i* выполнения **прямого хода** алгоритма Гаусса включает:
    - **Выбор ведущей строки**, для выполнения которого подзадачи с номерами *k, k>i,* должны обменяться своими элементами при исключаемой переменной *xi* для нахождения максимального по абсолютной величине значения. Строка, которой принадлежит выбранное значение, выбирается в качестве *ведущей строки* для выполняемой итерации метода,
    - **Рассылку** выбранной ведущей строки матрицы *A* и соответствующего элемента вектора *b* всем подзадачам с номерами *k, k>i*,
    - **Вычитание** строк для всех подзадачи *k (k>i)*, обеспечивая тем самым исключение соответствующей неизвестной *xi*.

Упражнение 4: Разработка параллельного алгоритма Гаусса…

- **Выделение информационных зависимостей**
  - При выполнении **обратного хода** метода Гаусса подзадачи выполняют необходимые вычисления для нахождения значения неизвестных:
    - Как только какая-либо подзадача *i*, $0 \le i < n-1$, определяет значение своей переменной *xi*, это значение должно быть разослано всем подзадачам с номерами *k*, $k < i$,
    - Далее подзадачи подставляют полученное значение новой неизвестной в линейное уравнение, представленное строкой матрицы *A*, расположенной в данной подзадаче, и выполняют корректировку значений для элементов вектора *b*.

Упражнение 4: Разработка параллельного алгоритма Гаусса…

- **Масштабирование и распределение подзадач по процессорам…**
  - В случае, когда размер матрицы, описывающей систему линейных уравнений, оказывается большим, чем число доступных процессоров (т.е., $n>p$), базовые подзадачи можно укрупнить, объединив в рамках одной подзадачи несколько строк матрицы.

Упражнение 4: Разработка параллельного алгоритма Гаусса

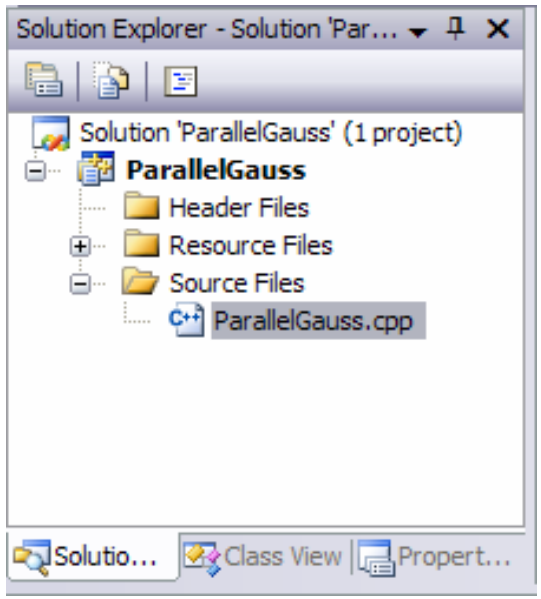
- **Масштабирование и распределение подзадач по процессорам**
  - Основным видом информационного взаимодействия подзадач является операция передачи данных от одного процессора всем процессорам вычислительной системы,
  - Как результат, для эффективной реализации требуемых информационных взаимодействий между базовыми подзадачами, топология сети передачи данных должны иметь структуру *гиперкуба* или *полного графа*.

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Поэтапная разработка параллельного алгоритма:**
  - Задание 1 – Открытие проекта **ParallelGauss**
  - Задание 2 – Определение размеров объектов и ввод исходных данных
  - Задание 3 – Завершение процесса вычислений
  - Задание 4 – Распределение данных между процессами
  - Задание 5 – Параллельное выполнение прямого хода алгоритма Гаусса
  - Задание 7 – Параллельное выполнение обратного хода алгоритма Гаусса
  - Задание 8 – Сбор результатов
  - Задание 9 – Проверка правильности работы программы
  - Задание 10 –Проведение вычислительных экспериментов

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 1** – Открытие проекта **ParallelGauss:…**
  - Запустите приложение Microsoft Visual Studio 2005
  - Откройте проект **ParallelGauss.sln**, расположенный в папке **C:\MsLabs\ParallelGauss**
  - При помощи окна **Solution Explorer** откройте файл **ParallelGauss.cpp**



Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 1** – Открытие проекта **ParallelGauss**:
  - Проект содержит функции:
    - *DummyDataInitialization* - начальное задание исходных данных,
    - *RandomDataInitialization* - задание исходных данных при помощи датчика случайных чисел,
    - *SerialResultCalculation* - последовательный алгоритм Гаусса (включая все функции, выполнение которых необходимо для выполнения последовательного алгоритма Гаусса),
    - *PrintMatrix, PrintVector* – форматированная печать матрицы и ветора
  - В главной функции приложения объявлены переменные *ProcNum*, *ProcRank*, *pMatrix*, *pVector*, *pResult*, *Size*,
  - Проинициализирована среда выполнения MPI-программы, определены количество процессов *ProcNum* и ранг каждого процесса *ProcRank,*
  - Скомпилируйте и запустите приложение. Убедитесь, что на экран выводится приветствие:

```
"Parallel Gauss algorithm for solving linear systems"
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 2** – Определение размеров объектов и ввод исходных данных:…

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 2** – Определение размеров объектов и ввод исходных данных:…
  - Переменные, необходимые для выполнения параллельного алгоритма Гаусса решения линейных систем:

```
double *pProcRows;   // The rows of matrix A on the process
double *pProcVector; // The elements of vector b
                     // on the process
double *pProcResult; // The elements of vector x
                     // on the process
int     RowNum;      // The Number of  the matrix rows on
                     // the  current process
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 2** – Определение размеров объектов и ввод исходных данных:…
  - Функция *ProcessInitialization:*
    - Организует диалог с пользователем на одном из процессов для того, чтобы определить размер системы *Size*, рассылает значение переменной *Size* на все процессы,
    - Вычисляет число строк, которые будут обрабатываться каждым процессом *RowNum*,
    - Выделяет память для хранения исходной системы на нулевом процессе и для хранения строк системы на всех процессах,
    - Определяет значения элементов матрицы системы и вектора правых частей.

```
void ProcessInitialization(double* &pMatrix, double* &pVector,
  double* &pResult, double* &pProcRows,double* &pProcVector,
  double* &pProcResult, int &Size, int &RowNum)
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 2** – Определение размеров объектов и ввод исходных данных:…
  - Реализуйте функцию *ProcessInitialization*, для определения числа строк, которые будут обрабатываться каждым конкретным процессом используйте механизм, изложенный в лабораторной работе 1,
  - Добавьте вызов функции инициализации процесса вычислений в основную функцию параллельного приложения,
  - Для контроля правильности выполнения этого этапа, распечатайте исходную систему линейных уравнений на нулевом процессе, используя функции *PrintMatrix* и *PrintVector*,
  - Скомпилируйте и запустите приложение. Убедитесь в его работоспособности.

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 2** – Определение размеров объектов и ввод исходных данных:

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Admin>cd c:\MsLabs\ParallelGauss\Debug

C:\MsLabs\ParallelGauss\Debug>mpiexec -n 4 ParallelGauss.exe
Parallel Gauss algorithm for solving linear systems

Enter size of the initial objects: 7
Initial matrix
 1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 1.0000  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 1.0000  1.0000  1.0000  0.0000  0.0000  0.0000  0.0000
 1.0000  1.0000  1.0000  1.0000  0.0000  0.0000  0.0000
 1.0000  1.0000  1.0000  1.0000  1.0000  0.0000  0.0000
 1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  0.0000
 1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
Initial vector
 1.0000  2.0000  3.0000  4.0000  5.0000  6.0000  7.0000
C:\MsLabs\ParallelGauss\Debug>
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 3** – Завершение процесса вычислений:…
  - Функция *ProcessTermination*:
    - Корректное завершение работы приложения,
    - Освобождение памяти, выделенной динамически в ходе выполнения приложения

```
// Function for computational process termination
void ProcessTermination (double* pMatrix, double* pVector,
  double* pResult, double* pProcRows, double* pProcVector,
  double* pProcResult)
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 3** – Завершение процесса вычислений:
  - Реализуйте функцию *ProcessTermination,*
  - Добавьте вызов функции *ProcessTermination* в функцию *main,*
  - Протестируйте работоспособность приложения

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 4** – Распределение данных между процессами:…
  - Исходные система линейных уравнений (матрица *pMatrix* и вектор правых частей *pVector*) на ведущем процессе,
  - система линейных уравнений должна быть распределена между процессами горизонтальными полосами (разделена на непрерывные последовательности строк)
  - Для разделения данных нужно использовать функцию *MPI_Scatterv*,
  - Для распределения данных реализуйте функцию *DataDistribution*:

```
// Data distribution among the processes
void DataDistribution(double* pMatrix, double* pProcRows,
  double* pVector,double* pProcVector, int Size, int RowNum)
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 4** – Распределение данных между процессами:…
  - Для контроля правильности выполнения этапа распределения данных предназначена функция *TestDistribution*:
    - Сначала распечатывается исходная системы линейных уравнений на нулевом процессе,
    - Далее распечатываются полосы системы с каждого из процессов параллельного приложения

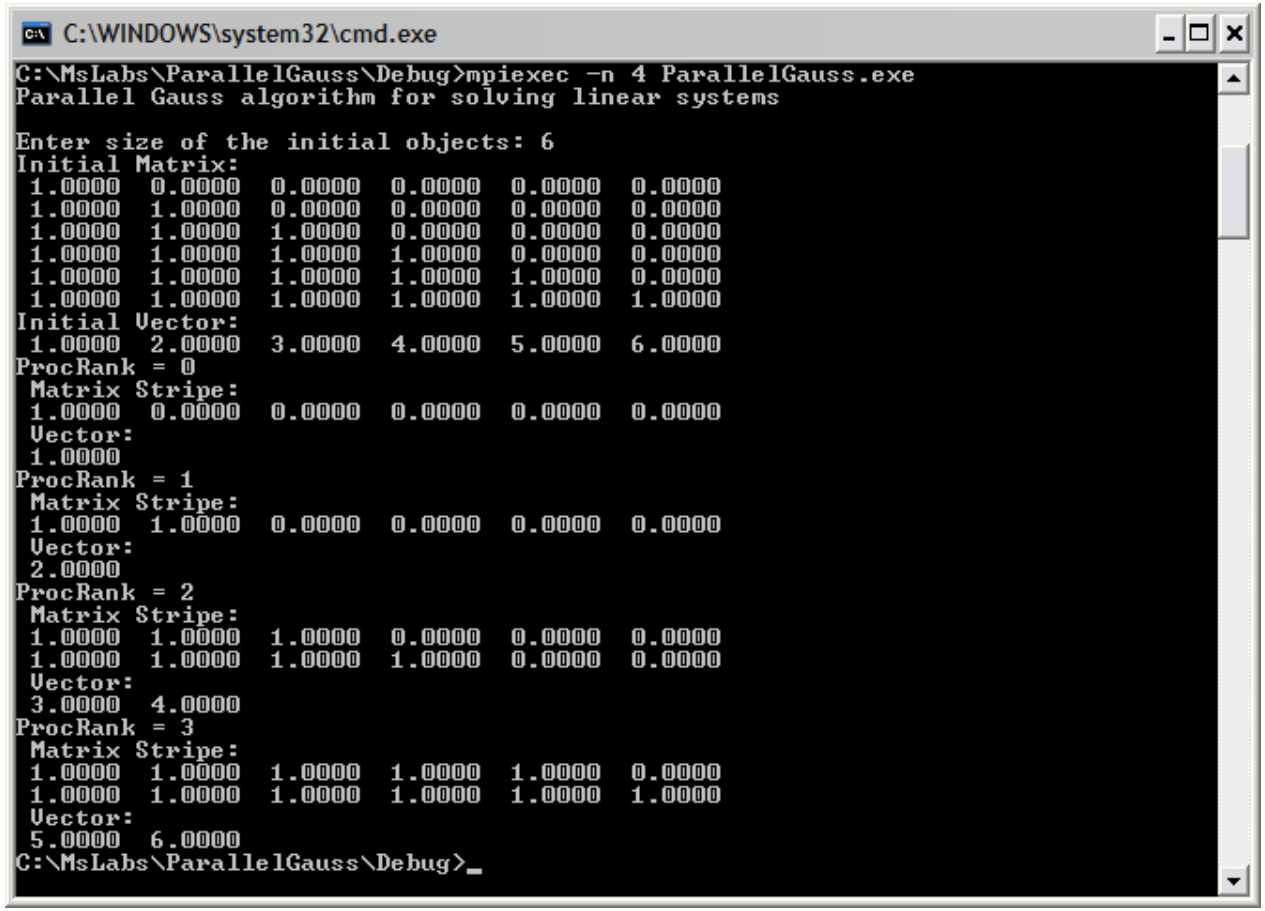
```
// Function for testing the data distribution
void TestDistribution (double* pMatrix, double* pVector,
  double* pProcRows, double* pProcVector, int Size,
  int RowNum);
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 4** – Распределение данных между процессами:…
  - Реализуйте функции *DataDistribution* и *TestDistribution*,
  - Добавьте вызов функции распределения данных в основную функцию параллельного приложения,
  - Проверьте правильность распределения данных при помощи функции *TestDistribution*,
  - Скомпилируйте и запустите приложение, убедитесь в его работоспособности.

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 4** – Распределение данных между процессами:



Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 5** – Параллельное выполнение прямого хода алгоритма Гаусса:…
  - Функция *ParallelResultCalculation*:
    - Параллельное выполнение прямого хода метода Гаусса,
    - Параллельное выполнение обратного хода метода Гаусса.

```
// Function for execution of the parallel Gauss algorithm
void ParallelResultCalculation(double* pProcRows,
  double* pProcVector, double* pProcResult, int Size,
  int RowNum), где
-pProcRows – полоса матрицы линейной системы,
-pProcVector – блок вектора правых частей,
-pProcResult – блок вектора неизвестных,
-Size – порядок линейной системы,
-RowNum – число строк, которые обрабатывает процесс.
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 5** – Параллельное выполнение прямого хода алгоритма Гаусса:…

```
// Function for execution of the parallel Gauss algorithm
void ParallelResultCalculation (double* pProcRows,
  double* pProcVector, double* pProcResult, int Size,
  int RowNum) {
  // Gaussian elimination
  ParallelGaussianElimination (pProcRows, pProcVector, Size,
    RowNum);
  // Back substitution
  ParallelBackSubstitution (pProcRows, pProcVector,
    pProcResult, Size, RowNum);
}
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 5** – Параллельное выполнение прямого хода алгоритма Гаусса:…
  - Необходимые переменные:
    - Массив *pParallelPivotPos* - элементы определяют номера строк матрицы, выбираемых в качестве ведущих, по итерациям прямого хода метода Гаусса. (Массив *pParallelPivotPos* является глобальным и любое его изменение в одном из процессов требует выполнения операции рассылки измененных данных всем остальным процессам программы)
    - Массив *pProcPivotIter* - элементы определяют номера итераций прямого хода метода Гаусса, на которых строки процесса использовались в качестве ведущих. (Массив *pProcPivotIter* является локальным для каждого процесса)

```
int *pParallelPivotPos; /* The number of rows selected as
  the pivot ones */
int *pProcPivotIter;    /* The number of iterations, at
  which the processor rows were used as the pivot ones */
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 5** – Параллельное выполнение прямого хода алгоритма Гаусса:…
  - В функции *ParallelResultCalculation* до начала выполнения прямого хода метода Гаусса выделите память для хранения массивов *pParallelPivotPos* и *pProcPivotIter*, присвойте элементам массивов начальные значения,
  - После вызова функции, выполняющей обратный ход, освободите выделенную память.

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 5** – Параллельное выполнение прямого хода алгоритма Гаусса:…
  - Функция *ParallelGaussianElimination* приводит матрицу линейной системы к верхне-треугольному виду с помощью эквивалентных преобразований:

```
// Gaussian elimination
void ParallelGaussianElimination (double* pProcRows,
  double* pProcVector, int Size, int RowNum),
где
-pProcRows – полоса матрицы линейной системы,
-pProcVector – бллок вектора правых частей,
-Size – порядок линейной системы,
-RowNum – число строк, которые обрабатывает данный процесс.
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 5** – Параллельное выполнение прямого хода алгоритма Гаусса:…
  - Функция *ParallelGaussianElimination*:
    - Каждый процесс выбирает локальную ведущую строчку из полосы системы, которую он обрабатывает,
    - При помощи функции обобщенной редукции *MPI_Allreduce* среди локальных ведущих строк выбирается глобальная ведущая строка,
    - Эта строка рассылается на все процессы при помощи функции *MPI_Bcast*,
    - Все процессы производят вычитание ведущей строки из строк, подлежащих обработке.

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 5** – Параллельное выполнение прямого хода алгоритма Гаусса:…
  - Необходимо выбрать максимальный среди полученных на каждом процессе ведущих элементов и определить, на каком процессе он располагается. Для выполнения таких действий в библиотеке MPI предназначена функция *MPI_Allreduce* :

```
int MPI_Allreduce(void *sendbuf, void *recvbuf, int count,
  MPI_Datatype type,MPI_Op op,MPI_Comm comm), где
- sendbuf- буфер памяти с отправляемым сообщением,
- recvbuf– буфер памяти для результирующего сообщения,
- count  - количество элементов в сообщениях,
- type   – тип элементов сообщений,
- op     - операция, которая должна быть выполнена над данными,
- comm   - коммуникатор, в рамках которого выполняется операция.
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 5** – Параллельное выполнение прямого хода алгоритма Гаусса:…
  - Для выполнения вычитания полученной глобальной ведущей строки из всех строк, которые обрабатывает данный процесс, предназначена функция *ParallelEliminateColumns*:

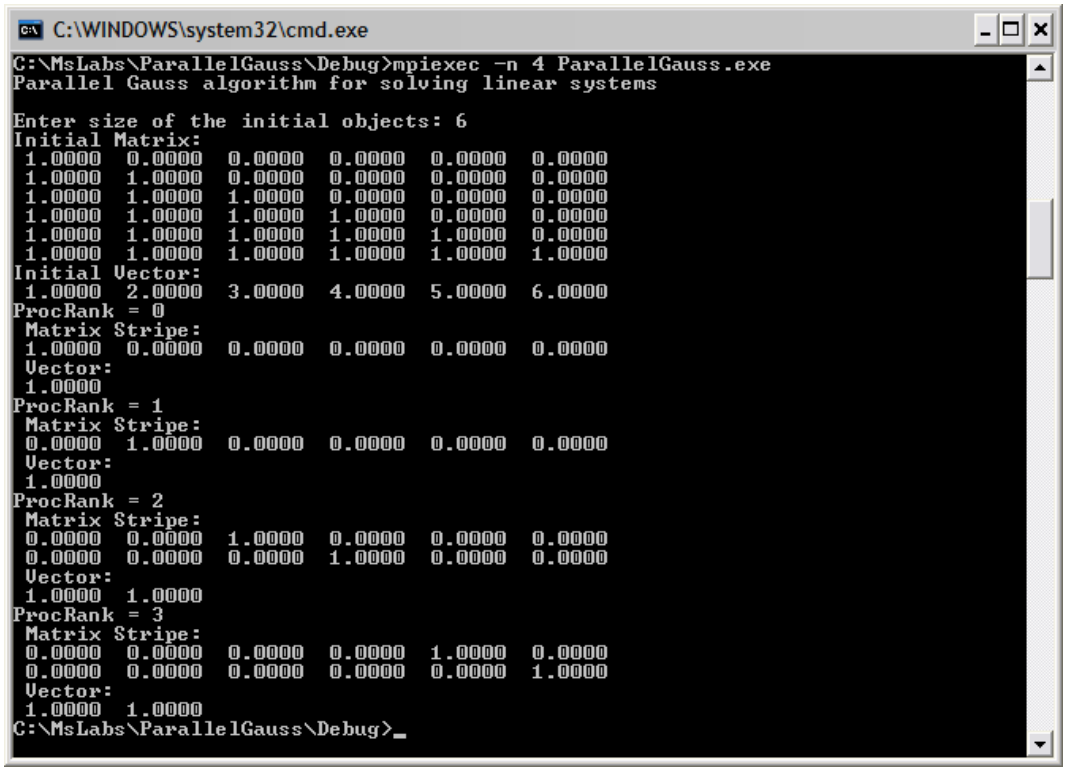
```
// Fuction for column elimination
void ParallelEliminateColumns (double* pProcRows,
  double* pProcVector, double* pPivotRow, int Size,
  int RowNum, int Iter), где
-pProcRows – полоса матрицы линейной системы,
-pProcVector – блок вектора правых частей,
-pPivotRow – ведущая строка,
-Size – порядок линейной системы,
-RowNum – количество строк, которые обрабатывает данный процесс,
-Iter – номер итерации.
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 5** – Параллельное выполнение прямого хода алгоритма Гаусса:…
  - Реализуйте функции *ParallelEliminateColumns* и *ParallelGaussianElinimation*,
  - Добавьте вызов функции *ParallelGaussianElimination* в тело функции *ParallelResultCalculation*,
  - Добавьте вызов функции *ParallelResultCalculation* в основную функцию параллельного приложения,
  - Для проверки правильности выполнения прямого хода метода Гаусса используйте функцию *TestDistribution*.

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 5** – Параллельное выполнение прямого хода алгоритма Гаусса:



– Скомпилируйте и запустите приложение. Убедитесь в его работоспособности:
  - Если прямой ход метода Гаусса выполняется верно, то после выполнения соответствующей функции матрица линейной системы должна быть приведена к верхне-треугольному виду (с точностью до перестановки строк):

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 7** – Параллельное выполнение обратного хода алгоритма Гаусса:…
  - Процессы выполняют необходимые вычисления для нахождения значения неизвестных,
  - Как только какой-либо процесс определяет значение своей переменной, это значение должно быть разослано всем процессам для того, чтобы они могли подставить полученное значение новой неизвестной и выполнить корректировку значений для элементов вектора правых частей.

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 7** – Параллельное выполнение обратного хода алгоритма Гаусса:…
  - На каждой итерации обратного хода необходимо определить строку, из которой можно вычислить значение очередного элемента результирующего вектора:
    - Номер этой строки хранится в массиве *pParallelPivotIter*,
    - По номеру необходимо определить номер процесса, на котором эта строка хранится, и номер этой строки в полосе *pProcRows* этого процесса.
  - Для нахождения очередной строки реализуйте функцию *FindBackPivotRow*.

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 7** – Параллельное выполнение обратного хода алгоритма Гаусса:…

```
// Function for finding the pivot row of the back
substitution
void FindBackPivotRow (int RowIndex, int Size,
  int &IterProcRank, int &IterPivotPos), где
-RowIndex – номер строки, для которой определяется расположение,
-Size – порядок линейной системы,
-IterProcRank – ранг того процесса, на котором располагается строка,
-IterPivotPos - номер этой строки в буфере pProcRows.
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 7** – Параллельное выполнение обратного хода алгоритма Гаусса:…
  - Для выполнения обратного хода метода Гаусса предназначена функция *ParallelBackSubstitution*:

```
// Back substitution
void ParallelBackSubstitution (double* pProcRows,
  double* pProcVector, double* pProcResult, int Size,
  int RowNum), где
-pProcRows - полоса матрицы линейной системы,
-pProcVector – блок вектора правых частей,
-pProcResult – блок вектора неизвестных,
-Size – порядок линейной системы,
-RowNum – количество строк, которые обрабатывает данный процесс.
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 7** – Параллельное выполнение обратного хода алгоритма Гаусса:
  - Реализуйте функции *FindBackPivotRow* и *ParallelBackSubstitution*,
  - Добавьте вызов функции, выполняющей обратный ход метода Гаусса, в тело функции *ParallelResultCalculation*,
  - Распечатайте блоки результирующего вектора,
  - Скомпилируйте и запустите приложение. Убедитесь в том, что параллельный алгоритм Гаусса решения линейных систем работает корректно.

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 8** – Сбор результатов:…
  - Необходимо собрать результирующий вектор *pResult* на ведущем процессе параллельного приложения из блоков *pProcResult*, расположенных на каждом процессе,
  - Для выполнения сбора воспользуйтесь функцией *MPI_Gatherv*.
  - Для сбора данных предназначена функция *ResultCollection*:

```
// Function for gathering the result vector
void ResultCollection(double* pProcResult, double* pResult)
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 8** – Сбор результатов:
  - Реализуйте функцию ResultCollection,
  - Добавьте вызов этой функции в тело главной функции параллельного приложения,
  - После выполнения сбора результирующего вектора, распечатайте этот вектор на нулевом процессе.
  - Скомпилируйте и запустите приложение. Убедитесь в его работоспособности.

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 9** – Проверка правильности работы программы:…
  - Функция *TestResult* сравнивает результаты работы последовательного и параллельного алгоритмов Гаусса путем поэлементного сравнения полученных векторов

```
// Function for testing the result
void TestResult(double* pMatrix, double* pVector,
  double* pResult, int Size)
где
-pMatrix – исходная матрица линейной системы,
-pVector – исходный вектор правых частей,
-pResult – вектор неизвестных, полученный при помощи
           параллельного алгоритма.
-Size    – порядок линейной системы.
```

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 9** – Проверка правильности работы программы:…
  - Результатом работы функции *TestResult* является печать диагностического сообщения,
  - Используя эту функцию, можно проверять результат работы параллельного алгоритма независимо от того, насколько велики исходные матрицы и при любых значениях исходных данных.

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений…

- **Задание 9** – Проверка правильности работы программы:
  - Замените вызов функции *DummyDataIniialization* на *RandomDataInitialization* внутри функции *ProcessInitialization*,
  - Добавьте вызов функции *TestResult* в функцию *main*,
  - Удалите функции отладочной печати,
  - Скомпилируйте и запустите приложение. Убедитесь в том, что реализованный параллельный алгоритм Гаусса решения систем линейных уравнений работает правильно.

Упражнение 5: Реализация параллельного алгоритма Гаусса решения систем линейных уравнений

- **Задание 10** – Проведение вычислительных экспериментов:
  - Добавьте вычисление и вывод времени выполнения параллельного алгоритма Гаусса,
  - Протестируйте работоспособность приложения,
  - Проведите вычислительные эксперименты,
  - Измерьте времена работы алгоритма Гаусса при различных количествах исходных данных и различном числе процессов,
  - Определите получаемое ускорение,
  - Вычислите теоретическое время работы параллельного алгоритма,
  - Заполните таблицу результатов вычислений

Заключение

- Рассмотрен параллельный алгоритм Гаусса решения систем линейных уравнений, основанный на разделении линейной системы на горизонтальные полосы
- Разработаны приложения, реализующие последовательный и параллельный алгоритм Гаусса
- Проведены вычислительные эксперименты, проведено сравнение последовательного и параллельного алгоритмов Гаусса решения систем линейных уравнений

Темы заданий для самостоятельной работы

- Изучите метод сопряженных градиентов решения систем линейных уравнений. Выполните реализацию последовательного и параллельного вариантов этого метода.

Литература

- **Березин И.С., Жидков И.П. (1966).** Методы вычислений. – М.:Наука.
- **Bertsekas, D.P., Tsitsiklis, J.N.** (1989). Parallel and distributed Computation. Numerical Methods. - Prentice Hall, Englewood Cliffs, New Jersey.
- **Blackford,** L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J. J., Hammarling, S., Henry, G., Petitet, A., Stanley, D. Walker, R.C. Whaley, K. (1997). Scalapack Users' Guide (Software, Environments, Tools). Soc for Industrial & Applied Math.
- **Dongarra, J.J., Duff, L.S., Sorensen, D.C., Vorst, H.A.V. (1999).** Numerical Linear Algebra for High Performance Computers (Software, Environments, Tools). Soc for Industrial & Applied Math/
- **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 сортировки**

Авторский коллектив

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

Гришагин В.А., доцент, к.ф.м.н.

Абросимова О.Н., ассистент (раздел 10)

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

Лабутин Д.Ю., ассистент (система ПараЛаб)

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. (раздел 11, лабораторные работы 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.