|  |  |
| --- | --- |
| |  | | --- | | МИНОБРНАУКИ РОССИИ  Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |

Институт Информационных технологий

Кафедра МОСИТ

**ПРАКТИЧЕСКАЯ РАБОТА №3**

по дисциплине «Параллельное программирование»

**Студент группы** ИКБО-11-17 Алиев Ю. А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись студента)*

**Руководитель работы** Сыромятников В. П. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись руководителя)*

Москва, 2020

Оглавление

[**Цель работы** 2](#_Toc54385753)

[**Постановка задачи** 2](#_Toc54385754)

[**Теоретическое введение** 2](#_Toc54385755)

[**Описание алгоритмов, используемых для решения задачи** 3](#_Toc54385756)

[**Текст исходного кода программы** 7](#_Toc54385757)

[**Тестирование** 9](#_Toc54385758)

[**Контрольные прогоны** 9](#_Toc54385759)

[**Выводы** 13](#_Toc54385760)

[**Список используемых информационных источников** 15](#_Toc54385761)

**Практическая работа №3**

Параллельное программирование с использованием расширенных средств технологии OpenMP (редукции, атомарных операций, критических секций, замков).

# **Цель работы**

* Отработка принципов распараллеливания программ;
* Реализация многопоточной программы с использованием различных средств синхронизации;
* Анализ эффективности параллельной программы в зависимости от объемов входных данных и разных количествах потоков.

# **Постановка задачи**

Составить программу последовательного и параллельного умножения двух матриц A[n,n] и В[n,n], используя горизонтальный (rowwize) алгоритм разбивки матрицы А по столбцам. Провести тестирование программы (в последовательном и параллельном исполнениях) на массивах размером n = 3, сформированных вводом с клавиатуры. Рабочие матрицы А и В, сформировать используя генератор псевдослучайных чисел. Провести контрольные прогоны программы для размеров n = 500, 600, 700, 800, 900, 1000 в последовательном и параллельном исполнении с количеством потоков p = 2, 4, 8, 16, 32. Полученные результаты свести в сводную таблицу. Построить графики изменения ускорения умножения в зависимости от размеров матриц. Построить графики изменения ускорения параллельных вычислений с разным количеством потоков. Вычислить показатели эффективности и стоимости параллельной реализации программы. Провести анализ полученных результатов. Сделать выводы о проделанной работе, основанные на эмпирических данных.

# **Теоретическое введение**

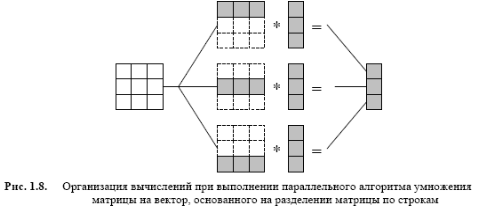
Вертикальный алгоритм относиться к ленточным алгоритмам умножения матриц. В данных алгоритмах матрицы разбиваются на

непрерывные последовательности строк или столбцов (полосы). В простейшем случае полосой может служить отдельная строка или столбец. В рассматриваемых ниже алгоритмах каждый процесс используется для вычисления одной строки результирующего произведения матриц AB. В этом случае процесс должен иметь доступ к соответствующей строке матрицы A и всей матрице B. Поскольку одновременное хранение всей матрицы B во всех процессах параллельного приложения требует чрезмерных затрат памяти, вычисления организуются таким образом, чтобы в каждый момент времени процессы содержали лишь часть элементов матрицы B (один столбец или одну строку), а доступ к остальной части обеспечивался бы при помощи передачи сообщений.

# **Описание алгоритмов, используемых для решения задачи**

Последовательный алгоритм представляется тремя вложенными циклами и ориентирован на последовательное вычисление столбцов результирующей матрицы С, здесь можно выполнить циклы параллельно, так как результаты не зависят друг от друга.

Пусть A и B – матрицы, которые необходимо перемножить, C – матрица, содержащая результат умножения. Начальные значения всех элементов матрицы C равны нулю. Алгоритм представляет собой итерационную процедуру. На каждой итерации алгоритма один столбец матрицы A умножается на один столбец матрицы B. При выполнении итерации проводится поэлементное умножение столбцов, что приводит к получению столбца частичных результатов для матрицы С. Пример работы алгоритма представлен на рисунке 1.



*Рисунок 1 Организация вычислений при выполнении параллельного алгоритма на примере умножения матрицы на вектор*

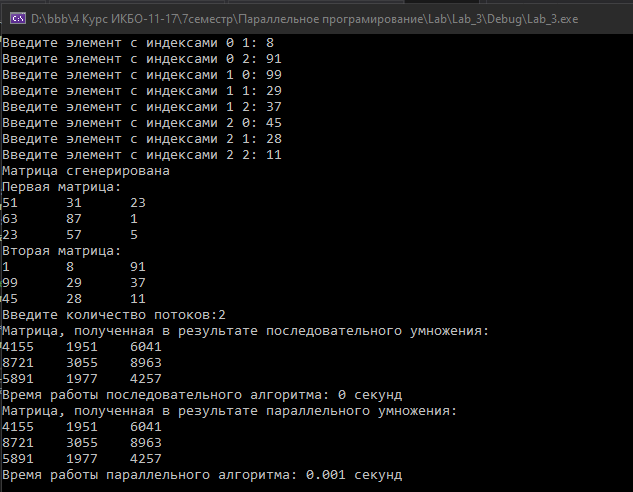
# **Текст исходного кода программы**

|  |
| --- |
| // Lab\_3.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.  //  #include <iostream>  #include <omp.h>  #include <Windows.h>  #include <time.h>  #include <ctime>  // Генерация рандомных чисел  int get\_random\_number(int min, int max)  {  static const double fraction = 1.0 / (static\_cast<double>(RAND\_MAX) + 1.0);  // Равномерно распределяем рандомное число в нашем диапазоне  return static\_cast<int>(rand() \* fraction \* (max - min + 1) + min);  }  // Функция создания двумерной матрицы  int\*\* array\_generator(unsigned int dim)  {  int\*\* ptrary = new int\* [dim];  for (int i = 0; i < dim; i++) {  ptrary[i] = new int[dim];  }  return ptrary;  }  // Функция удаления двумерной матрицы  void array\_destroyer(int\*\* ary, unsigned int dim1)  {  for (int i = 0; i < dim1; i++) {  delete[] ary[i];  }  delete[] ary;  }  // Функция заполнения матрицы нулями  int\*\* matrix\_generate\_empty(int\*\* matrix, int n) {  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  matrix[i][j] = 0;  }  }  return matrix;  }  // Заполнение матрицы вручную  int\*\* fill\_matrix(int\*\* matrix, int n)  {  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  std::cout << "Введите элемент с индексами " << i << " " << j << ": ";  std::cin >> matrix[i][j];  }  }  std::cout << "Матрица сгенерирована\n";  return matrix;  }  // Функция вывода матрицы на экран  int\*\* matrix\_output(int\*\* matrix, int n)  {  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  std::cout << matrix[i][j] << "\t";  }  std::cout << std::endl;  }  return matrix;  }  // Функция заполнения матрицы с помощью генератора случайных чисел  int\*\* matrix\_generate(int\*\* matrix, int n) {  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  matrix[i][j] = get\_random\_number(0, 200);  }  }  std::cout << "Матрица сгенерирована\n";  return matrix;  }  // Процедура последовательного умножения двух матриц  void sequential\_matrix\_multiplication(int\*\* matrix\_one, int\*\* matrix\_two, int n)  {  int\*\* matrix\_result = array\_generator(n);  matrix\_generate\_empty(matrix\_result, n);  int start = clock();  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  for (int k = 0; k < n; k++) {  matrix\_result[i][j] += matrix\_one[i][k] \* matrix\_two[k][j];  }  }  }  int stop = clock();  //std::cout << "Матрица, полученная в результате последовательного умножения: \n";  //matrix\_output(matrix\_result, n);  std::cout << "Время работы последовательного алгоритма: " << ((double)stop - (double)start) / CLOCKS\_PER\_SEC << " секунд" << std::endl;  array\_destroyer(matrix\_result, n);  }  // Процедура параллельного умножения двух матриц  void parallel\_matrix\_multiplication(int\*\* matrix\_one, int\*\* matrix\_two, int n, int thread)  {  int\*\* matrix\_result = array\_generator(n);  matrix\_generate\_empty(matrix\_result, n);  int start = clock();  omp\_set\_num\_threads(thread);  #pragma omp parallel for  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  for (int k = 0; k < n; k++) {  matrix\_result[i][j] += matrix\_one[i][k] \* matrix\_two[k][j];  }  }  }  int stop = clock();  //std::cout << "Матрица, полученная в результате параллельного умножения: \n";  //matrix\_output(matrix\_result, n);  std::cout << "Время работы параллельного алгоритма: " << ((double)stop - (double)start) / CLOCKS\_PER\_SEC << " секунд" << std::endl;  array\_destroyer(matrix\_result, n);  }  void menu()  {  std::cout << "Меню программы:\n";  std::cout << "1 - Ручное заполнение\n";  std::cout << "2 - Заполнение случайными числами\n";  std::cout << "0 - Завершение программы\n";  }  int main()  {  // Настройка кодировки в консоли  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  int command; // Выбор раздела меню  int number\_matrix; // Выбор размера матрицы  int threads; // Выбор количества поток  int\*\* matrix\_one = 0;  int\*\* matrix\_two = 0;  srand(time(NULL));  menu();  do  {  std::cout << "\nВведите команду: ";  std::cin >> command;  switch (command)  {  case 1:    do  {  std::cout << "Введите размерность матрицы: ";  std::cin >> number\_matrix;    if (number\_matrix > 0)  {  break;  }  else  {  std::cout << "Проверьте входные данные! Размерность должна быть положительна!\n";  }  } while (true);    matrix\_one = array\_generator(number\_matrix);  matrix\_two = array\_generator(number\_matrix);  std::cout << "Заполнение первой матрицы" << std::endl;  fill\_matrix(matrix\_one, number\_matrix);  std::cout << "Заполнение второй матрицы" << std::endl;  fill\_matrix(matrix\_two, number\_matrix);  std::cout << "Первая матрица:\n";  matrix\_output(matrix\_one, number\_matrix);  std::cout << "Вторая матрица:\n";  matrix\_output(matrix\_two, number\_matrix);  do  {  std::cout << "Введите количество потоков:";  std::cin >> threads;  if (threads > 0)  {  break;  }  else  {  std::cout << "Проверьте входные данные! Размерность должна быть положительна!\n";  }  } while (true);  // Последовательное умножение  sequential\_matrix\_multiplication(matrix\_one, matrix\_two, number\_matrix);  // Распараллеливание умножения матриц  parallel\_matrix\_multiplication(matrix\_one, matrix\_two, number\_matrix, threads);  array\_destroyer(matrix\_one, number\_matrix);  array\_destroyer(matrix\_two, number\_matrix);  break;  case 2:  do  {  std::cout << "Введите размерность матрицы: ";  std::cin >> number\_matrix;  if (number\_matrix > 0)  {  break;  }  else  {  std::cout << "Проверьте входные данные! Размерность должна быть положительна!\n";  }  } while (true);  matrix\_one = array\_generator(number\_matrix);  matrix\_two = array\_generator(number\_matrix);  matrix\_generate(matrix\_one, number\_matrix);  matrix\_generate(matrix\_two, number\_matrix);  do  {  std::cout << "Введите количество потоков:";  std::cin >> threads;  if (threads > 0)  {  break;  }  else  {  std::cout << "Проверьте входные данные! Размерность должна быть положительна!\n";  }  } while (true);  // Последовательное умножение  sequential\_matrix\_multiplication(matrix\_one, matrix\_two, number\_matrix);  // Распараллеливание умножения матриц  parallel\_matrix\_multiplication(matrix\_one, matrix\_two, number\_matrix, threads);  array\_destroyer(matrix\_one, number\_matrix);  array\_destroyer(matrix\_two, number\_matrix);  break;  case 0:  // Завершение программы  std::cout << "Программа завершена";  break;  default:  std::cout << "Неверно введена комманда\n";  break;  }  } while (command);  } |

Листинг файла main.cpp. Исходный код программы.

# **Тестирование**

Тестирование программы происходит при размерности матрицы n = 3, при количестве потоков p = 2. Результат выполнения программы представлен на рисунке 6.



*Рисунок 2. Результаты тестирования*.

# **Контрольные прогоны**

На основе выполненных тестов были составлены сводные таблицы, на основе которых составлены графики, указанные в основной задаче.

*Таблица 1. Результаты вертикального алгоритма (columnwize) методов без распараллеливания*

|  |  |  |
| --- | --- | --- |
| № тестирования | Размер матрицы,  n | Время выполнения работы программы, сек. |
| 1 | 500 | 0.741 |
| 2 | 600 | 1.416 |
| 3 | 700 | 2.255 |
| 4 | 800 | 3.934 |
| 5 | 900 | 5.97 |
| 6 | 1000 | 9.418 |

*Таблица 2 Результаты работы вертикального алгоритма (columnwize) при количестве потоков p = 2*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № тестирования | Размер матрицы, n | Время выполнения программы, сек | Ускорение (Sn) | Эффективность | Стоимость |
| 1 | 500 | 0.373 | 1.986 | 0.993 | 0.746 |
| 2 | 600 | 0.767 | 1.846 | 0.923 | 1.534 |
| 3 | 700 | 1.273 | 1.771 | 0.885 | 2.546 |
| 4 | 800 | 2.007 | 1.976 | 0.98 | 4.014 |
| 5 | 900 | 3.1 | 1.925 | 0.962 | 6.2 |
| 6 | 1000 | 4.766 | 1.976 | 0.988 | 9.532 |

*Таблица 3. Результаты работы вертикального алгоритма (columnwize) при количестве потоков p = 4*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № тестирования | Размер матрицы, n | Время выполнения программы, сек | Ускорение (Sn) | Эффективность | Стоимость |
| 1 | 500 | 0.199 | 3.723 | 0.93 | 0.796 |
| 2 | 600 | 0.372 | 3.806 | 0.951 | 1.488 |
| 3 | 700 | 0.579 | 3.894 | 0.973 | 2.316 |
| 4 | 800 | 0.986 | 3.989 | 0.997 | 3.944 |
| 5 | 900 | 1.535 | 3.889 | 0.972 | 6.14 |
| 6 | 1000 | 2.495 | 3.774 | 0.943 | 9.98 |

*Таблица 4. Результаты работы вертикального алгоритма (columnwize) при количестве потоков p = 8*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № тестирования | Размер матрицы, n | Время выполнения программы, сек | Ускорение (Sn) | Эффективность | Стоимость |
| 1 | 500 | 0.212 | 3.495 | 0.436 | 1.696 |
| 2 | 600 | 0.326 | 4.343 | 0.542 | 2.608 |
| 3 | 700 | 0.538 | 4.191 | 0.523 | 4.304 |
| 4 | 800 | 0.818 | 4.809 | 0.601 | 6.544 |
| 5 | 900 | 1.238 | 4.822 | 0.602 | 9.904 |
| 6 | 1000 | 2.148 | 4.384 | 0.548 | 17.184 |

*Таблица 5. Результаты работы вертикального алгоритма (columnwize) при количестве потоков p = 16*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № тестирования | Размер матрицы, n | Время выполнения программы, сек | Ускорение (Sn) | Эффективность | Стоимость |
| 1 | 500 | 0.247 | 3 | 0.185 | 3.952 |
| 2 | 600 | 0.319 | 4.43 | 0.277 | 5.104 |
| 3 | 700 | 0.547 | 4.122 | 0.257 | 8.752 |
| 4 | 800 | 0.824 | 2.156 | 0.134 | 29.184 |
| 5 | 900 | 1.208 | 4.94 | 0.308 | 19.328 |
| 6 | 1000 | 2.267 | 2.69 | 0.168 | 56 |

*Таблица 6. Результаты вертикального алгоритма (columnwize) при количестве потоков p = 32*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № тестирования | Размер матрицы, n | Время выполнения программы, сек | Ускорение (Sn) | Эффективность | Стоимость |
| 1 | 500 | 0.187 | 3.962 | 0.123 | 5.984 |
| 2 | 600 | 0.347 | 4.080 | 0.127 | 11.104 |
| 3 | 700 | 0.542 | 4.16 | 0.13 | 17.344 |
| 4 | 800 | 0.839 | 4.688 | 0.146 | 26.848 |
| 5 | 900 | 1.235 | 4.834 | 0.151 | 39.52 |
| 6 | 1000 | 2.238 | 4.208 | 0.131 | 71.616 |

*Рисунок 3. Зависимости ускорения при различных потоках от размерности матрицы*

На основе данного рисунка можно сделать вывод о том, что ускорение происходит на небольших величинах больше всего при выполнении алгоритма на 32 потока.

*Рисунок 4. Зависимости времени выполнения при различных потоках от количества элементов матрицы*

На основе данного рисунка можно сделать вывод о том, что самое быстрое выполнение происходит при использовании 32-х поточной программой.

*Рисунок 5. Зависимости эффективности выполнения при различных потоках от количества элементов матрицы*

На основе данного рисунка можно сделать вывод о том, самая высокая эффективность достигается при 4 поточном вычислений значений. Худший вариант – использование 32 потоков.

*Рисунок 6. Зависимости стоимости выполнения при различных потоках от количества элементов матрицы*

На основе данного рисунка можно сделать вывод о том, самая высокая стоимость выполнения достигается при использовании 32 потоков, самая маленькая – при использовании 2 потоков.

# **Выводы**

В результате выполнения лабораторной работы был реализован вертикальный (columnwize) алгоритм разбивки матрицы. Были построены графики времени работы программы двух разных алгоритмов, графики зависимости ускорения от размерности и количества потоков, были рассчитаны эффективности распараллеливания и построены наглядные графики.

1. Максимальное ускорение для n = 500 достигается при p = 32;
2. Максимальное ускорение для n = 600 достигается при p = 16;
3. Максимальное ускорение для n = 700 достигается при p = 8;
4. Максимальное ускорение для n = 800 достигается при p = 8;
5. Максимальное ускорение для n = 900 достигается при p = 16;
6. Максимальное ускорение для n = 1000 достигается при p = 8;
7. Для всех размерностей наибольшая эффективность достигается при p = 4;

# **Список используемых информационных источников**

1. Сыромятников В. П. Курс лекций по дисциплине «Параллельное программирование». – РТУ МИРЭА, 2020-2021 г.

2. Руководство по использованию модуля OpenMp [Электронный ресурс]. – Режим доступа: <http://ccfit.nsu.ru/arom/data/openmp.pdf> (дата обращения 10.11.2020)

3. Официальные характеристики процессора Intel Core i7 8700k [Электронный ресурс]. – Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/80815/intel-core-i5-4590-processor-6m-cache-up-to-3-70-ghz.html> (дата обращения 12.11.2020)

4. Официальный сайт дистрибутива MVS Code [Электронный ресурс]. – Режим доступа: <https://code.visualstudio.com> (дата обращения 17.11.2020)

5. Параллельные методы умножения матрицы на вектор [Электронный ресурс]. – Режим доступа: [www.hpcc.unn.ru](http://www.hpcc.unn.ru) (дата обращения 17.11.2020)