**ОТЧЕТ**

**по лабораторной работе**

|  |  |
| --- | --- |
| *Дисциплина* | Теория языков программирования и вычислительных процессов (ТЯПиВП) |
| *Номер работы* | 2 |
| *Тема работы* | Организация параллельных вычислений на сети ЭВМ |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Принял:** | |  | **Выполнил (-и):** | | | |
|  | А.Г.Чефранов, профессор каф. МОП ЭВМ |  |  | *Фамилия И.О.* | *Группа* | *Оценка* |
|  | Заветная В.Т. | гр. КТбо4-1 |  |
|  | Яшенков А.В. | гр. КТбо4-1 |  |

**ОГЛАВЛЕНИЕ**

[1 ЦЕЛЬ РАБОТЫ 3](#_Toc117323076)

[2 ХОД РАБОТЫ 4](#_Toc117323077)

[2.1 Вычисление обратной матрицы 4](#_Toc117323078)

[2.2 Эксперименты 5](#_Toc117323079)

[3 ВЫВОД 7](#_Toc117323080)

[4 ПРИЛОЖЕНИЕ А 8](#_Toc117323081)

# ЦЕЛЬ РАБОТЫ

Реализовать вычисление умножение матриц с использованием сети ЭВМ и без неё, сравнить скорость работы полученных методов и проанализировать скорость вычислений.

# ХОД РАБОТЫ

## Вычисление обратной матрицы

Пусть даны две прямоугольные матрицы и размерности и соответственно:

Тогда матрица размерностью :

в которой:

называется её ***произведением***.

В данной работе подсчёт произведения элементов матриц будет происходить параллельно, то есть цикл умножения будет разделяться на несколько независимых процессорных потоков. Все мы можем использовать 12 потоков (использовался процессор с 6 ядрами и 12 потоками).

В процессе работы было создано 2 консольных приложения (клиент и сервер) на языке C++ для платформы GNU/Linux. Сервер получает данные о размерах исходных матриц из параметров командной строки, номер порта, который нужно прослушивать; разделяет каждую матрицу на 2 части и отправляет 1-ю часть на клиент; вычисляет произведение 2-й части матриц в однопоточном режиме и получает данные от клиента; далее сервер объединяет полученные результаты путём сложения элементов с одинаковыми индексами; выводит затраченное время в миллисекундах; вычисляет произведение целых матриц в однопоточном режиме, и также выводит затраченное время в миллисекундах.

Для проведения экспериментов был запущены несколько тестов.

Пример результатов работы теста (рис. 1).

Изображение выглядит как текст

Автоматически созданное описание

Рис. 1 – результаты вычисления умножения матриц.

## Эксперименты

Проведём ряд экспериментов, чтобы определить влияние вычислений в сети ЭВМ на время вычисления обратной матрицы.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  | 2978 | 2307 |
|  |  | 6400 | 5246 |
|  |  | 202 | 91 |
|  |  | 19 | 4 |
|  |  | 7 | 0 |
|  |  | 3 | 0 |

– количество строк и столбцов в первой матрице в формате . Где N – количество строк, а M – количество столбцов.

– количество строк и столбцов во второй матрице в формате . Где N – количество строк, а M – количество столбцов.

– время работы программы в однопоточном режиме в миллисекундах.

– время работы программы в параллельном режиме в миллисекундах.

**ВЫВОД**

В данной лабораторной работе мы реализовали вычисления умножения матриц с использованием сети ЭВМ и без неё. Проведя ряд экспериментов, мы можем видеть, что уже на количество строк 1-й матрицы или столбцов 2-й матрицы, тем больше времени тратится на вычисление. Также для проведения экспериментов были использованы 2 компьютера, а передача данных велась по сети Wi-Fi. Исходя из полученных результатов видно, что распределять вычисления вычисление умножения матриц всего лишь на 2 компьютера является нецелесообразной идей, так как большее время тратится на передачу данных.

# ПРИЛОЖЕНИЕ А

В приложении представлен текст исходной программы:

**matrix.h:**

#ifndef MAT\_H

#define MAT\_H

#include <iostream>

#include <omp.h>

namespace lab1

{

  class matrix

  {

  public:

    matrix() = default;

    matrix(int rowsCount, int colsCount)

        : data\_(new int \*[rowsCount]),

          rows\_(rowsCount),

          cols\_(colsCount)

    {

      for (int i = 0; i < rows\_; i++)

      {

        data\_[i] = new int[cols\_];

      }

    }

    matrix(const matrix &mat)

        : data\_(new int \*[mat.rows\_]),

          rows\_(mat.rows\_),

          cols\_(mat.cols\_)

    {

      for (int i = 0; i < rows\_; i++)

      {

        data\_[i] = new int[cols\_];

      }

      for (int i = 0; i < rows\_; i++)

      {

        for (int j = 0; j < cols\_; j++)

        {

          data\_[i][j] = mat.data\_[i][j];

        }

      }

    }

    matrix(matrix &&mat)

        : data\_(mat.data\_),

          rows\_(mat.rows\_),

          cols\_(mat.cols\_)

    {

      mat.data\_ = nullptr;

      mat.rows\_ = 0;

      mat.cols\_ = 0;

    }

    ~matrix()

    {

      if (data\_)

      {

        for (int i = 0; i < rows\_; i++)

        {

          if (data\_[i])

          {

            delete[] data\_[i];

          }

        }

      }

    }

    matrix &operator=(const matrix &mat) = delete;

    matrix &operator=(matrix &&mat) = delete;

    inline int \*operator[](int rowIndex)

    {

      if (0 <= rowIndex && rowIndex < rows\_)

      {

        return data\_[rowIndex];

      }

      return nullptr;

    }

    inline const int \*operator[](int rowIndex) const

    {

      if (0 <= rowIndex && rowIndex < rows\_)

      {

        return data\_[rowIndex];

      }

      return nullptr;

    }

    inline operator bool()

    {

      if (!data\_)

      {

        return false;

      }

      for (int i = 0; i < rows\_; i++)

      {

        if (!data\_[i])

        {

          return false;

        }

      }

      return true;

    }

    inline int \*\*data()

    {

      return data\_;

    }

    inline const int \*const \*data() const

    {

      return data\_;

    }

    inline int rows() const

    {

      return rows\_;

    }

    inline int cols()

    {

      return cols\_;

    }

    matrix operator\*(const matrix &mat)

    {

      if (cols\_ != mat.rows\_)

      {

        return matrix();

      }

      matrix result(rows\_, mat.cols\_);

      for (int i = 0; i < rows\_; i++)

      {

        for (int j = 0; j < mat.cols\_; j++)

        {

          result[i][j] = 0;

          for (int k = 0; k < cols\_; k++)

          {

            result[i][j] += (data\_[i][k] \* mat.data\_[k][j]);

          }

        }

      }

      return result;

    }

    friend std::ostream &operator<<(std::ostream &os, const matrix &mat)

    {

      if (!mat.data\_)

      {

        return os;

      }

      for (int i = 0; i < mat.rows\_; i++)

      {

        for (int j = 0; j < mat.cols\_; j++)

        {

          if (!mat.data\_[i])

          {

            return os;

          }

          os << mat.data\_[i][j] << ' ';

        }

        os << std::endl;

      }

      return os;

    }

    static matrix parallel\_multiplication(const matrix &matrix1, const matrix &matrix2, int thread\_num)

    {

      matrix result(matrix1.rows\_, matrix2.cols\_);

      int i, j, k;

      omp\_set\_num\_threads(thread\_num);

      #pragma omp parallel for shared(matrix1, matrix2, result) private(i, j, k) collapse(2)

      for (i = 0; i < matrix1.rows\_; i++)

      {

        for (j = 0; j < matrix2.cols\_; j++)

        {

          result.data\_[i][j] = 0;

          for (k = 0; k < matrix1.cols\_; k++)

          {

            result.data\_[i][j] += (matrix1.data\_[i][k] \* matrix2.data\_[k][j]);

          }

        }

      }

      return result;

    }

  private:

    int \*\*data\_ = nullptr;

    int rows\_{0};

    int cols\_{0};

  };

}

#endif

**main.cpp (клиент):**

#include <iostream>

#include <boost/asio.hpp>

#include <boost/stacktrace.hpp>

#include <thread>

#include <chrono>

#include <cstdarg>

#include <vector>

#include "matrix.h"

using namespace boost::asio;

using ip::tcp;

using std::cout;

using std::endl;

using std::string;

static int to\_int(const std::string &str)

{

  std::istringstream sin(str);

  int res;

  sin >> res;

  return res;

}

std::vector<lab2::matrix> read\_matrixes(tcp::socket& socket)

{

  boost::system::error\_code error;

  boost::asio::streambuf receive\_buffer;

  int bytes\_cnt;

  boost::asio::read(socket, boost::asio::buffer(&bytes\_cnt, sizeof(int)), error);

  std::vector<int> buffer(bytes\_cnt);

  boost::asio::read(socket, boost::asio::buffer(buffer.data(), bytes\_cnt \* sizeof(int)), error);

  auto buf\_iter = buffer.begin();

  std::vector<lab2::matrix> result;

  for (int matrix\_cnt = 0; matrix\_cnt < 2; matrix\_cnt++)

  {

    int rows = \*buf\_iter++;

    int cols = \*buf\_iter++;

    lab2::matrix matrix(rows, cols);

    for (int i = 0; i < rows; i++)

    {

      for (int j = 0; j < cols; j++)

      {

        matrix[i][j] = \*buf\_iter++;

      }

    }

    result.push\_back(std::move(matrix));

  }

  return result;

}

std::vector<int> matrix\_to\_vector(lab2::matrix& matrix)

{

  std::vector<int> mat\_v(matrix.rows() \* matrix.cols() + 3);

  auto mat\_v\_iter = mat\_v.begin();

  \*mat\_v\_iter++ = mat\_v.size() - 1;

  \*mat\_v\_iter++ = matrix.rows();

  \*mat\_v\_iter++ = matrix.cols();

  for (int i = 0; i < matrix.rows(); i++)

  {

    for (int j = 0; j < matrix.cols(); j++)

    {

      \*mat\_v\_iter++ = matrix[i][j];

    }

  }

  return mat\_v;

}

void write\_matrix(tcp::socket &socket, lab2::matrix& matrix)

{

  auto tx\_buf = matrix\_to\_vector(matrix);

  boost::system::error\_code error;

  boost::asio::write(socket, boost::asio::buffer(tx\_buf.data(), tx\_buf.size() \* sizeof(int)), error);

}

int main()

{

  boost::asio::io\_service io\_service;

  // socket creation

  tcp::socket socket(io\_service);

  // connection

  bool connected = false;

  int  connecte\_time = 0;

  int  time\_to\_connect = 10;

  for (uint32\_t i = 0; i < time\_to\_connect && !connected; ++i)

  {

    try

    {

      socket.connect(tcp::endpoint(boost::asio::ip::address::from\_string("127.0.0.1"), 1234));

      connected = true;

    }

    catch (const std::exception &e)

    {

      std::cout << "Trying to connect... Attempt: {" << i << "}" << std::endl;

      std::this\_thread::sleep\_for(std::chrono::seconds(1));

    }

  }

  if (!connected)

  {

    std::cout << "\nClien couldn`t connect to server!" << std::endl;

    exit(1);

  }

  auto mat\_v = read\_matrixes(socket);

  auto result = mat\_v[0] \* mat\_v[1];

  write\_matrix(socket, result);

  return 0;

}

**main.cpp (сервер):**

#include <iostream>

#include <boost/asio.hpp>

#include <vector>

#include <sstream>

#include <fstream>

#include <cstdarg>

#include "matrix.h"

using namespace boost::asio;

using ip::tcp;

using std::cout;

using std::endl;

using std::string;

static int to\_int(const std::string &str)

{

  std::istringstream sin(str);

  int res;

  sin >> res;

  return res;

}

static std::vector<lab2::matrix> matrix\_generator(int argc, char \*\*argv)

{

  std::vector<lab2::matrix> result;

  if (1 == argc)

  {

    return result;

  }

  for (int i = 1; i < argc; ++i)

  {

    if (std::string(argv[i]) == "--matrix")

    {

      int rows = to\_int(argv[++i]);

      int cols = to\_int(argv[++i]);

      lab2::matrix mat(rows, cols);

      for (int i = 0; i < rows; i++)

      {

        for (int j = 0; j < cols; j++)

        {

          mat[i][j] = rand() % 11;

        }

      }

      // std::cout << mat << std::endl

      //           << std::endl;

      result.push\_back(std::move(mat));

    }

  }

  return result;

}

void write\_matrixes(tcp::socket &socket, std::vector<lab2::matrix>& mat\_v)

{

  int tx\_buf\_size = mat\_v[0].cols() \* mat\_v[0].rows() / 2 +

                    mat\_v[1].cols() \* mat\_v[1].rows() / 2 + 5;

  std::vector<int> tx\_buf(tx\_buf\_size);

  auto tx\_buf\_iter = tx\_buf.begin();

  \*tx\_buf\_iter++ = tx\_buf.size() - 1;

  for (int mat\_cnt = 0; mat\_cnt < mat\_v.size(); mat\_cnt++)

  {

    int rows;

    int cols;

    if (!mat\_cnt)

    {

      \*tx\_buf\_iter++ = rows = mat\_v[mat\_cnt].rows();

      \*tx\_buf\_iter++ = cols = mat\_v[mat\_cnt].cols() / 2;

    }

    else

    {

      \*tx\_buf\_iter++ = rows = mat\_v[mat\_cnt].rows() / 2;

      \*tx\_buf\_iter++ = cols = mat\_v[mat\_cnt].cols();

    }

    for (int i = 0; i < rows; i++)

    {

      for (int j = 0; j < cols; j++)

      {

        \*tx\_buf\_iter++ = mat\_v[mat\_cnt][i][j];

      }

    }

  }

  boost::system::error\_code error;

  boost::asio::write(socket, boost::asio::buffer(tx\_buf.data(), tx\_buf.size() \* sizeof(int)), error);

}

lab2::matrix read\_matrix(tcp::socket& socket)

{

  boost::system::error\_code error;

  boost::asio::streambuf receive\_buffer;

  int bytes\_cnt;

  boost::asio::read(socket, boost::asio::buffer(&bytes\_cnt, sizeof(int)), error);

  std::vector<int> buffer(bytes\_cnt);

  boost::asio::read(socket, boost::asio::buffer(buffer.data(), bytes\_cnt \* sizeof(int)), error);

  auto buf\_iter = buffer.begin();

  int rows = \*buf\_iter++;

  int cols = \*buf\_iter++;

  lab2::matrix matrix(rows, cols);

  for (int i = 0; i < rows; i++)

  {

    for (int j = 0; j < cols; j++)

    {

      matrix[i][j] = \*buf\_iter++;

    }

  }

  return matrix;

}

int main(int argc, char \*\*argv)

{

  srand(time(NULL));

  int port;

  std::ofstream fout("result.txt", std::ios::app);

  if (1 == argc)

  {

    std::cerr << "Input params wasn`t specified!" << std::endl;

    exit(1);

  }

  for (int i = 0; i < argc; i++)

  {

    if (std::string(argv[i]) == "--port")

    {

      port = to\_int(argv[++i]);

    }

  }

  auto mat\_v = matrix\_generator(argc, argv);

  if (mat\_v.size() != 2)

  {

    std::cerr << "The required number of matrices wasn`t specified!" << std::endl;

    exit(1);

  }

  // auto result = mat\_v[0] \* mat\_v[1];

  boost::asio::io\_service io\_service;

  // listen for new connection

  tcp::acceptor acceptor\_(io\_service, tcp::endpoint(tcp::v4(), 1234));

  // socket creation

  tcp::socket socket\_(io\_service);

  // waiting for connection

  // std::cout << "Wainting for connection..."

            // << std::endl;

  acceptor\_.accept(socket\_);

  auto begin = std::chrono::steady\_clock::now();

  write\_matrixes(socket\_, mat\_v);

  auto matrix = read\_matrix(socket\_);

  lab2::matrix mat1(mat\_v[0].rows(), mat\_v[0].cols() / 2 ), mat2(mat\_v[1].rows() / 2 , mat\_v[1].cols());

  for (int i = 0 ; i < mat\_v[0].rows(); i++)

  {

    for (int j = mat\_v[0].cols() / 2; j < mat\_v[0].cols(); j++)

    {

      mat1[i][j - mat\_v[0].cols() / 2] = mat\_v[0][i][j];

    }

  }

  for (int i = mat\_v[1].rows() / 2; i < mat\_v[1].rows(); i++)

  {

    for (int j = 0; j < mat\_v[1].cols(); j++)

    {

      mat2[i - mat\_v[1].rows() / 2][j] = mat\_v[1][i][j];

    }

  }

  lab2::matrix matrix1 = mat1 \* mat2;

  // std::cout << matrix1 << std::endl;

  // std::cout << std::endl;

  for (int i = 0; i < matrix1.rows(); i++)

  {

    for (int j = 0; j < matrix1.cols(); j++)

    {

      matrix[i][j] += matrix1[i][j];

    }

  }

  auto end = std::chrono::steady\_clock::now();

  auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

  printf("[P]: time -> %ldms\n", elapsed\_ms.count());

  fout << "[P]: time -> " << elapsed\_ms.count() << "ms\n";

  begin = std::chrono::steady\_clock::now();

  mat\_v[0] \* mat\_v[1];

  end = std::chrono::steady\_clock::now();

  elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

  printf("[NP]: time -> %ldms\n", elapsed\_ms.count());

  fout << "[NP]: time -> " << elapsed\_ms.count() << "ms\n\n";

  return 0;

}