**ОТЧЕТ**

**по лабораторной работе**

|  |  |
| --- | --- |
| *Дисциплина* | Теория языков программирования и вычислительных процессов (ТЯПиВП) |
| *Номер работы* | 1 |
| *Тема работы* | Исследование влияния псевдопараллелизма на производительность |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 параллелизма и без него, сравнить скорость работы полученных методов и проанализировать влияние параллелизма на 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 консольное приложение на языке C++ для платформы GNU/Linux, которое получает данные о размерах исходных матриц из параметров командной строки, вычисляет произведение матриц в однопоточном режиме, выводит затраченное время в миллисекундах, затем вычисляет произведение матриц в параллельном режиме, и также выводит затраченное время в миллисекундах.

Для проведения экспериментов был написан скрипт на языке *bash*, который запускает полученное приложение и перенаправляется стандартные вывод приложения в файл результатов.

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 параллелизма на время вычисления обратной матрицы. График зависимости времени работы от количества потоков для матриц и представлен на рис. 2. Полные результаты работы теста представлены в ПРИЛОЖЕНИИ Б**.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  | 4608 | 4874 | 1 |
|  |  | 4687 | 2548 | 2 |
|  |  | 4701 | 1751 | 3 |
|  |  | 4521 | 1302 | 4 |
|  |  | 4731 | 1076 | 5 |
|  |  | 4675 | 871 | 6 |
|  |  | 4523 | 1089 | 7 |
|  |  | 4846 | 1019 | 8 |
|  |  | 4960 | 934 | 9 |
|  |  | 4634 | 854 | 10 |
|  |  | 4789 | 828 | 11 |
|  |  | 4935 | 887 | 12 |

– 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.

– количество потоков.

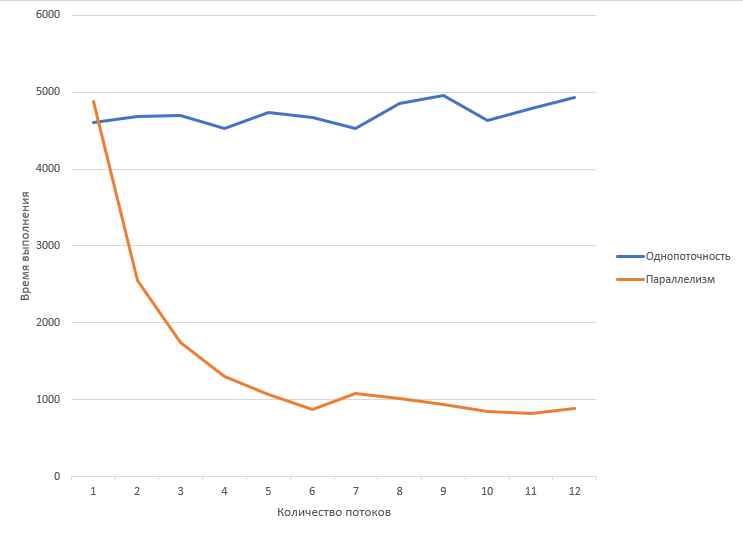


Рис. 2 - График зависимости времени работы от количества потоков

**ВЫВОД**

В данной лабораторной работе мы реализовали вычисления умножения матриц с использованием параллелизма и без него. Проведя ряд экспериментов, мы можем видеть, что уже на 2-х потоках произведение матриц вычисляется заметно быстрее, нежели в однопоточном режиме, начиная с количества потоков равном 6 производительность практически остается на одинаковом уровне, но всё ещё в разы быстрее, чем однопоточный режим.

# 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:**

#include <iostream>

#include <sstream>

#include <string>

#include <vector>

#include <cstdlib>

#include <omp.h>

#include <chrono>

#include <fstream>

#include "matrix.h"

using namespace std;

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

{

  std::istringstream sin(str);

  int res;

  sin >> res;

  return res;

}

static std::vector<lab1::matrix> matrix\_generator(int argc, char \*\*argv)

{

  std::vector<lab1::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]);

      lab1::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 printMatrix(int \*\*matrix, int n, int m)

{

  for(int i = 0; i < n; i++)

  {

    for(int j = 0; j < m; j++)

    {

        std::cout << matrix[i][j] << ' ';

    }

    std::cout << std::endl;

  }

}

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

{

  srand(time(NULL));

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

  std::ofstream fout;

  std::string filename;

  int thread\_num;

  for (int i = 0; i < argc; i++)

  {

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

    {

      fout.open(argv[++i], std::ios::app);

      filename = argv[i];

    }

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

    {

      thread\_num = atoi(argv[++i]);

    }

  }

  // Не паралелльно вычисление

  {

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

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

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

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

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

  }

  // Паралелльное вычисление

  {

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

    auto res1 = lab1::matrix::parallel\_multiplication(mat\_v[0], mat\_v[1], thread\_num);

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

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

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

  }

  return 0;

}

# ПРИЛОЖЕНИЕ Б

Discription:

[P] - parallel calculating.

[NP] - not parallel calculating.

Test 1: Same size, different thread count.

Test 1.0:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 1

[NP]: time -> 4608ms

[P]: time -> 4874ms

Test 1.1:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 2

[NP]: time -> 4687ms

[P]: time -> 2548ms

Test 1.2:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 3

[NP]: time -> 4701ms

[P]: time -> 1751ms

Test 1.3:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 4

[NP]: time -> 4521ms

[P]: time -> 1302ms

Test 1.4:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 5

[NP]: time -> 4731ms

[P]: time -> 1076ms

Test 1.5:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 6

[NP]: time -> 4675ms

[P]: time -> 871ms

Test 1.6:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 7

[NP]: time -> 4523ms

[P]: time -> 1089ms

Test 1.7:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 8

[NP]: time -> 4846ms

[P]: time -> 1019ms

Test 1.8:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 9

[NP]: time -> 4960ms

[P]: time -> 934ms

Test 1.9:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 10

[NP]: time -> 4634ms

[P]: time -> 854ms

Test 1.10:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 11

[NP]: time -> 4789ms

[P]: time -> 828ms

Test 1.11:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 12

[NP]: time -> 4935ms

[P]: time -> 887ms

Test 2: Same thread number [2], different size.

Test 2.0:

Programm arguments: --matrix 1 2 --matrix 2 1 --threads 2

[NP]: time -> 0ms

[P]: time -> 0ms

Test 2.1:

Programm arguments: --matrix 2 5 --matrix 5 2 --threads 2

[NP]: time -> 0ms

[P]: time -> 0ms

Test 2.2:

Programm arguments: --matrix 10 5 --matrix 5 10 --threads 2

[NP]: time -> 0ms

[P]: time -> 0ms

Test 2.3:

Programm arguments: --matrix 10 10 --matrix 10 10 --threads 2

[NP]: time -> 0ms

[P]: time -> 0ms

Test 2.4:

Programm arguments: --matrix 10 500 --matrix 500 10 --threads 2

[NP]: time -> 0ms

[P]: time -> 0ms

Test 2.5:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 2

[NP]: time -> 4681ms

[P]: time -> 2592ms

Test 2.6:

Programm arguments: --matrix 200 500 --matrix 500 200 --threads 2

[NP]: time -> 93ms

[P]: time -> 48ms

Test 2.7:

Programm arguments: --matrix 190 50 --matrix 50 190 --threads 2

[NP]: time -> 8ms

[P]: time -> 6ms

Test 2.8:

Programm arguments: --matrix 10000 5 --matrix 5 10000 --threads 2

[NP]: time -> 3011ms

[P]: time -> 1514ms

Test 2.9:

Programm arguments: --matrix 1000 1000 --matrix 1000 1000 --threads 2

[NP]: time -> 5004ms

[P]: time -> 2773ms

Test 2.10:

Programm arguments: --matrix 5 5000 --matrix 5000 5 --threads 2

[NP]: time -> 0ms

[P]: time -> 0ms

Test 2.11:

Programm arguments: --matrix 1500 500 --matrix 500 1500 --threads 2

[NP]: time -> 5353ms

[P]: time -> 3097ms

Test 3: Different size, different thread number.

Test 3.0:

Programm arguments: --matrix 1 2 --matrix 2 1 --threads 1

[NP]: time -> 0ms

[P]: time -> 0ms

Test 3.1:

Programm arguments: --matrix 2 5 --matrix 5 2 --threads 2

[NP]: time -> 0ms

[P]: time -> 0ms

Test 3.2:

Programm arguments: --matrix 10 5 --matrix 5 10 --threads 3

[NP]: time -> 0ms

[P]: time -> 0ms

Test 3.3:

Programm arguments: --matrix 10 10 --matrix 10 10 --threads 4

[NP]: time -> 0ms

[P]: time -> 0ms

Test 3.4:

Programm arguments: --matrix 10 500 --matrix 500 10 --threads 5

[NP]: time -> 0ms

[P]: time -> 0ms

Test 3.5:

Programm arguments: --matrix 1000 500 --matrix 500 2000 --threads 6

[NP]: time -> 4555ms

[P]: time -> 900ms

Test 3.6:

Programm arguments: --matrix 200 500 --matrix 500 200 --threads 7

[NP]: time -> 90ms

[P]: time -> 25ms

Test 3.7:

Programm arguments: --matrix 190 50 --matrix 50 190 --threads 8

[NP]: time -> 9ms

[P]: time -> 2ms

Test 3.8:

Programm arguments: --matrix 10000 5 --matrix 5 10000 --threads 9

[NP]: time -> 2953ms

[P]: time -> 575ms

Test 3.9:

Programm arguments: --matrix 1000 1000 --matrix 1000 1000 --threads 10

[NP]: time -> 4621ms

[P]: time -> 854ms

Test 3.10:

Programm arguments: --matrix 5 5000 --matrix 5000 5 --threads 11

[NP]: time -> 0ms

[P]: time -> 1ms

Test 3.11:

Programm arguments: --matrix 1500 500 --matrix 500 1500 --threads 12

[NP]: time -> 5380ms

[P]: time -> 895ms

Test 4: A little size, min/maximum thread number.

Test 4.0:

Programm arguments: --matrix 1 2 --matrix 2 1 --threads 1

[NP]: time -> 0ms

[P]: time -> 0ms

Test 4.1:

Programm arguments: --matrix 1 2 --matrix 2 1 --threads 12

[NP]: time -> 0ms

[P]: time -> 0ms

Test 5: A large size, min/maximum thread number.

Test 5.0:

Programm arguments: --matrix 1000 200 --matrix 200 1000 --threads 1

[NP]: time -> 987ms

[P]: time -> 1173ms

Test 5.1:

Programm arguments: --matrix 1000 200 --matrix 200 1000 --threads 12

[NP]: time -> 964ms

[P]: time -> 157ms