Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 8

по курсу «Численные методы линейной алгебры»

«Метод Штрассена»

Студент группы ИУ9-72Б Князихин Д. П.

Преподаватель Посевин Д. П.

Москва 2023

1 Цель

1. Реализовать метод Штрассена.

2. Реализовать рекурсию через многопоточность.

3. Сравнить точность результата со стандартным алгоритмом умножения.

4. Построить на одном графике зависимость времени t (сек) умножения двух матриц размера N x N стандартным алгоритмом, методом Штрассена и методом Штрассена с многопоточностью от размера матрицы N.

2 Результаты

Листинг 1: Методы

```
#include <iostream>
#include <unistd.h>
#include <vector>
#include <future>
#include <matplot/matplot.h>
namespace plt = matplot;


double fRand(double fMin, double fMax)
{
    double f = (double)rand() / RAND_MAX;
    return fMin + f * (fMax - fMin);
}

std::vector<std::vector<double>> init_matrix(int N) {
    std::vector<std::vector<double>> out;

    for (int i = 0; i < N; i++) {
        std::vector<double> row;
        for(int j = 0; j < N; j++) {
            row.push_back(0.0);
        }
        out.push_back(row);
    }

    return out;
}

std::vector<std::vector<double>> gener_matrix(int N, double fMin, double
    fMax) {
```

```
30      std::vector<std::vector<double>> out = init_matrix(N);
31      for (int i = 0; i < N; i++) {
32          for (int j = 0; j < N; j++) {
33              out[i][j] = fRand(fMin, fMax);
34          }
35      }
36      return out;
37  }
38
39  std::vector<std::vector<double>> sum_matrix(const std::vector<std::
        vector<double>>& A, const std::vector<std::vector<double>>& B) {
40      int n = A.size();
41      auto C = init_matrix(n);
42      for (int i = 0; i < n; i++) {
43          for (int j = 0; j < n; j++) {
44              C[i][j] = A[i][j]+B[i][j];
45          }
46      }
47      return C;
48  }
49
50  std::vector<std::vector<double>> sub_matrix(const std::vector<std::
        vector<double>>& A, const std::vector<std::vector<double>>& B) {
51      int n = A.size();
52      auto C = init_matrix(n);
53      for (int i = 0; i < n; i++) {
54          for (int j = 0; j < n; j++) {
55              C[i][j] = A[i][j]-B[i][j];
56          }
57      }
58
59      return C;
60  }
61
62  std::vector<std::vector<double>> mult_matrix(const std::vector<std::
        vector<double>>& A, const std::vector<std::vector<double>>& B) {
63      std::vector<std::vector<double>> out;
64      int N = A.size();
65
66      out = init_matrix(N);
67
68      for(int i = 0; i < N; i++){
69          for(int j = 0; j < N; j++){
70              out[i][j] = 0;
71              for(int k = 0; k < N; k++){
72                  out[i][j] += A[i][k] * B[k][j];
```

```
73              }
74          }
75      }
76      return out;
77  }
78
79  std::vector<std::vector<double>> strass(const std::vector<std::vector<
        double>>& A,const std::vector<std::vector<double>>& B, int n_min) {
80      int n = A.size();
81      if (n <= n_min) {
82          return mult_matrix(A, B);
83      } else {
84          int m = n / 2;
85          auto a11 = init_matrix(m);
86          auto a12 = init_matrix(m);
87          auto a21 = init_matrix(m);
88          auto a22 = init_matrix(m);
89
90          auto b11 = init_matrix(m);
91          auto b12 = init_matrix(m);
92          auto b21 = init_matrix(m);
93          auto b22 = init_matrix(m);
94
95          for (int u = 0; u < m; u++) {
96              for (int delta = 0; delta < m; delta++) {
97                  a11[u][delta] = A[u][delta];
98                  a12[u][delta] = A[u][delta + m];
99                  a21[u][delta] = A[u+m][delta];
100                 a22[u][delta] = A[u+m][delta+m];
101
102                 b11[u][delta] = B[u][delta];
103                 b12[u][delta] = B[u][delta + m];
104                 b21[u][delta] = B[u+m][delta];
105                 b22[u][delta] = B[u+m][delta+m];
106             }
107         }
108
109         auto P1 = strass(sum_matrix(a11,a22), sum_matrix(b11, b22),
        n_min);
110         auto P2 = strass(sum_matrix(a21, a22), b11, n_min);
111         auto P3 = strass(a11, sub_matrix(b12, b22), n_min);
112         auto P4 = strass(a22, sub_matrix(b21, b11), n_min);
113         auto P5 = strass(sum_matrix(a11, a12), b22, n_min);
114         auto P6 = strass(sub_matrix(a21, a11), sum_matrix(b11, b12),
        n_min);
```

```
115         auto P7 = strass(sub_matrix(a12, a22), sum_matrix(b21, b22),
    n_min);
116
117         auto C11 = sum_matrix(sub_matrix(sum_matrix(P1, P4), P5), P7);
118         auto C12 = sum_matrix(P3, P5);
119         auto C21 = sum_matrix(P2, P4);
120         auto C22 = sum_matrix(sub_matrix(sum_matrix(P1, P3), P2), P6);
121
122         auto C = init_matrix(n);
123
124         for (int i = 0; i < m; i++) {
125             for (int j = 0; j < m; j++) {
126                 C[i][j] = C11[i][j];
127                 C[i][j+m] = C12[i][j];
128                 C[i+m][j] = C21[i][j];
129                 C[i+m][j+m] = C22[i][j];
130             }
131         }
132         return C;
133     }
134 }
135
136
137 std::vector<std::vector<double>> strass_parallel(const std::vector<std::
    vector<double>>& A, const std::vector<std::vector<double>>& B, int
    n_min) {
138     int n = A.size();
139     if (n <= n_min) {
140         return mult_matrix(A, B);
141     } else {
142         int m = n / 2;
143         auto a11 = init_matrix(m);
144         auto a12 = init_matrix(m);
145         auto a21 = init_matrix(m);
146         auto a22 = init_matrix(m);
147
148         auto b11 = init_matrix(m);
149         auto b12 = init_matrix(m);
150         auto b21 = init_matrix(m);
151         auto b22 = init_matrix(m);
152
153         for (int u = 0; u < m; u++) {
154             for (int delta = 0; delta < m; delta++) {
155                 a11[u][delta] = A[u][delta];
156                 a12[u][delta] = A[u][delta + m];
157                 a21[u][delta] = A[u+m][delta];
```

```
158                 a22[u][delta] = A[u+m][delta+m];
159
160                 b11[u][delta] = B[u][delta];
161                 b12[u][delta] = B[u][delta + m];
162                 b21[u][delta] = B[u+m][delta];
163                 b22[u][delta] = B[u+m][delta+m];
164             }
165         }
166
167         auto task1 = std::async(std::launch::async, strass, sum_matrix(
    a11,a22), sum_matrix(b11, b22), n_min);
168         auto task2 = std::async(std::launch::async, strass, sum_matrix(
    a21, a22), b11, n_min);
169         auto task3 = std::async(std::launch::async, strass, a11,
    sub_matrix(b12, b22), n_min);
170         auto task4 = std::async(std::launch::async, strass, a22,
    sub_matrix(b21, b11), n_min);
171         auto task5 = std::async(std::launch::async, strass, sum_matrix(
    a11, a12), b22, n_min);
172         auto task6 = std::async(std::launch::async, strass, sub_matrix(
    a21, a11), sum_matrix(b11, b12), n_min);
173         auto task7 = std::async(std::launch::async, strass, sub_matrix(
    a12, a22), sum_matrix(b21, b22), n_min);
174
175         auto P1 = task1.get();
176         auto P2 = task2.get();
177         auto P3 = task3.get();
178         auto P4 = task4.get();
179         auto P5 = task5.get();
180         auto P6 = task6.get();
181         auto P7 = task7.get();
182
183         auto C11 = sum_matrix(sub_matrix(sum_matrix(P1, P4), P5), P7);
184         auto C12 = sum_matrix(P3, P5);
185         auto C21 = sum_matrix(P2, P4);
186         auto C22 = sum_matrix(sub_matrix(sum_matrix(P1, P3), P2), P6);
187
188         auto C = init_matrix(n);
189
190         for (int i = 0; i < m; i++) {
191             for (int j = 0; j < m; j++) {
192                 C[i][j] = C11[i][j];
193                 C[i][j+m] = C12[i][j];
194                 C[i+m][j] = C21[i][j];
195                 C[i+m][j+m] = C22[i][j];
196             }
```

```
197         }
198         return C;
199     }
200 }
201
202
203
204 void print_matrix(std::vector<std::vector<double>> A) {
205     for (int i = 0; i < A.size(); i++) {
206         for (int j = 0; j < A.size(); j++) {
207             std::cout << A[i][j] << " ";
208         }
209         std::cout << std::endl;
210     }
211
212     std::cout << std::endl;
213 }
214
215 void print_vector(std::vector<double> v) {
216     for(auto elem: v) {
217         std:: cout << elem << " ";
218     }
219     std::cout << std::endl;
220 }
221
222 int main() {
223     int N = 16;
224     auto A = gener_matrix(N, -10, 10);
225     auto B = gener_matrix(N, -10, 10);
226
227     auto C = strass(A, B, 4);
228
229     print_matrix(C);
230
231     print_matrix(mult_matrix(A, B));
232
233
234     //
235     int maxN = 1 << 11;
236
237     //

238     std::vector<int> sizes;
239     std::vector<double> standardTimes;
240     std::vector<double> strassenTimes;
241     std::vector<double> parallelStrassenTimes;
```

```
242
243     int n_min = 64;
244     for (int N = 2; N <= maxN; N *= 2) {
245         sizes.push_back(N);
246         std::cout << N << std::endl;
247         //                                          A     B
248
249         auto A = gener_matrix(N, -10, 10);
250         auto B = gener_matrix(N, -10, 10);
251
252         //

253         auto start = std::chrono::high_resolution_clock::now();
254         auto m1 = mult_matrix(A, B);
255         auto end = std::chrono::high_resolution_clock::now();
256         double standardTime = std::chrono::duration<double>(end - start)
    .count();
257         standardTimes.push_back(standardTime);
258
259         //

260         start = std::chrono::high_resolution_clock::now();
261         auto m2 = strass(A, B, n_min);
262         end = std::chrono::high_resolution_clock::now();
263         double strassenTime = std::chrono::duration<double>(end - start)
    .count();
264         strassenTimes.push_back(strassenTime);
265
266         //

267         start = std::chrono::high_resolution_clock::now();
268         auto m3 = strass_parallel(A, B, n_min);
269         end = std::chrono::high_resolution_clock::now();
270         double parallelStrassenTime = std::chrono::duration<double>(end
    - start).count();
271         parallelStrassenTimes.push_back(parallelStrassenTime);
272
273
274     }
275
276     //

277     plt::plot(sizes, standardTimes, sizes, strassenTimes, sizes,
    parallelStrassenTimes);
278
279     plt::xlabel("Matrix Size (N)");
```

```
280     plt::ylabel("Time (s)");
281     auto ax1 = plt::nexttile();
282     plt::title("Matrix Multiplication Time Comparison");
283
284     ::matplot::legend(ax1, {"Classic mult", "Strassen", "Parallel
        Strassen"});
285
286
287     // //
288     plt::save("info.png");
289     plt::show();
290
291     print_vector(standardTimes);
292     print_vector(strassenTimes);
293     print_vector(parallelStrassenTimes);
294
295     return 0;
296 }
```

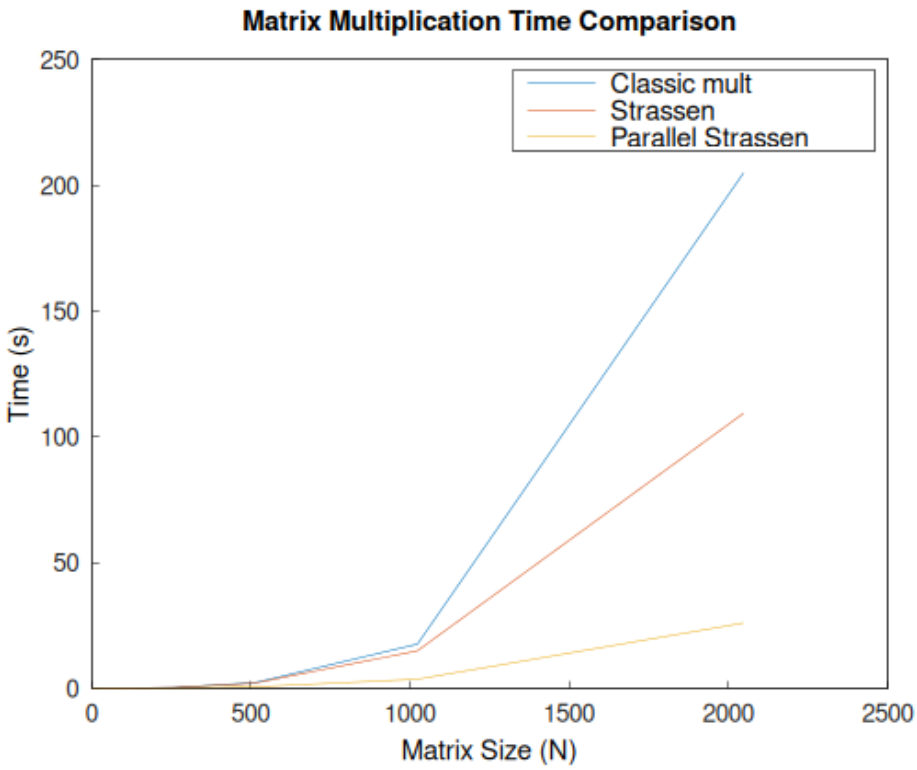

Рис. 1 — Результат