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 *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине: «Разработка программных систем»

Студент	Харитонов Евгений Юрьевич
Группа	РК6-62б
Тип задания	лабораторная работа
Тема лабораторной работы	многопоточное программирование

Студент _____ **Харитонов Е.Ю.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Федорук В.Г.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2020 г.

Оглавление

Задание на лабораторную работу

Вариант 13.

Разработать многопотоковый вариант программы моделирования распространения электрических сигналов в двухмерной прямоугольной сетке RC-элементов. Метод формирования математической модели - узловой. Метод численного интегрирования - явный Эйлера. Внешнее воздействие - источники тока и напряжения. Количество потоков, временной интервал моделирования и количество (кратное 8) элементов в сетке - параметры программы. Программа должна демонстрировать ускорение по сравнению с последовательным вариантом. Предусмотреть визуализацию результатов посредством утилиты gnuplot.

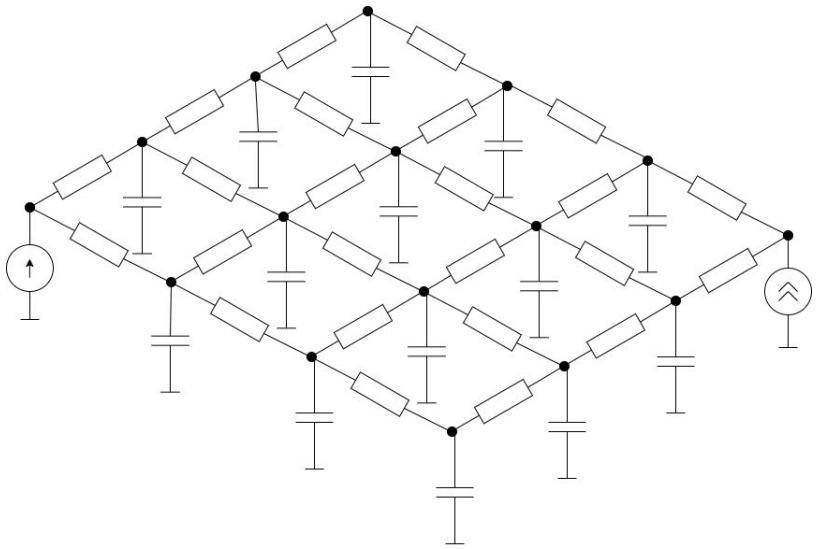


Рис. 1. Пример двумерной прямоугольной сетки RC элементов

Примечание. Узловой метод для формирования математической модели системы использует второй закон Кирхгофа - сумма токов в узле равна нулю. В данном задании для каждого j-ого узла цепочки уравнение баланса токов имеет вид

$I_{Rлев}-I_{Rправ}-I_C=0$ или $(V^i_{j-1}-V^i_j)/R-(V^i_j-V^i_{j+1})/R-C*dV^i_j/dt=0$, где i - номер временного шага, V - потенциал узла.

Описание структуры программы и реализованных способов взаимодействия потоков управления

Программа при запуске принимает в качестве аргументов количество потоков(threads_count), время моделирования(t), длину сетки M, ширину сетки N. Временной шаг h, емкость конденсаторов C и сопротивление резисторов R задаются как глобальные переменные.

После старта приложение инициализирует объект структуры Grig, хранящий массивы напряжений в узлах сетки в текущий и предыдущий момент времени(двумерные массивы длиной M;N), массив источников тока и массив источников напряжения (структуры VoltS и CurrS). Далее происходит инициализация атрибутов потоков и создается 2 барьера для синхронизации работы потоков с шириной (threads_count + 1). Создаются сами потоки, которые будут обеспечивать работу функции расчета напряжений в узлах сетки (функция solver). Узлы сетки делятся между потоками по принципу:

Если количество узлов сетки M*N кратно числу потоков threads_count, то все потоки обрабатывают одинаковое количество узлов сетки (M*N/thread_count). Если количество узлов сетки M*N не кратно числу потоков threads_count, то первые (thread_count - 1) потоков обрабатывают (M*N/thread_count + 1) узлов каждый, а один поток обрабатывает (M*N - (M*N/thread_count + 1)*(thread_count-1)) узлов.

Функция solver рассчитывает в цикле новые значения в тех узлах сетки, за которые “отвечает” переданный поток (структура Thread хранит номер первого и номер последнего узла, которые рассчитывает этот поток (поскольку в текущей реализации двумерная сетка хранится в одномерном

массиве, где i - строка из M строк, j - столбец из N столбцов, номер узла - i*N+j)).

Перед началом расчета значений узлах в текущий момент времени потоки синхронизируются, встречая барьер barr1, а после выполняют расчеты. В этот момент поток, работающий в функции main, при помощи gnuplot выводит график напряжений в узлах в предыдущий момент времени. После расчетов и вывода графика потоки приостанавливают выполнение для синхронизации на барьере barr2, чтобы поток, работающий в функции main, поменял местами указатели, ссылающиеся на массив напряжений в текущий и предыдущий момент времени. Происходит новая итерация цикла и действия в функции solver и функции main (участок кода, отвечающий за вывод графика) повторяются. Так происходит до тех пор, пока не будут рассчитаны напряжения в узлах сетки для всего временного интервала.

Схема взаимодействия потоков представлена на рис.2

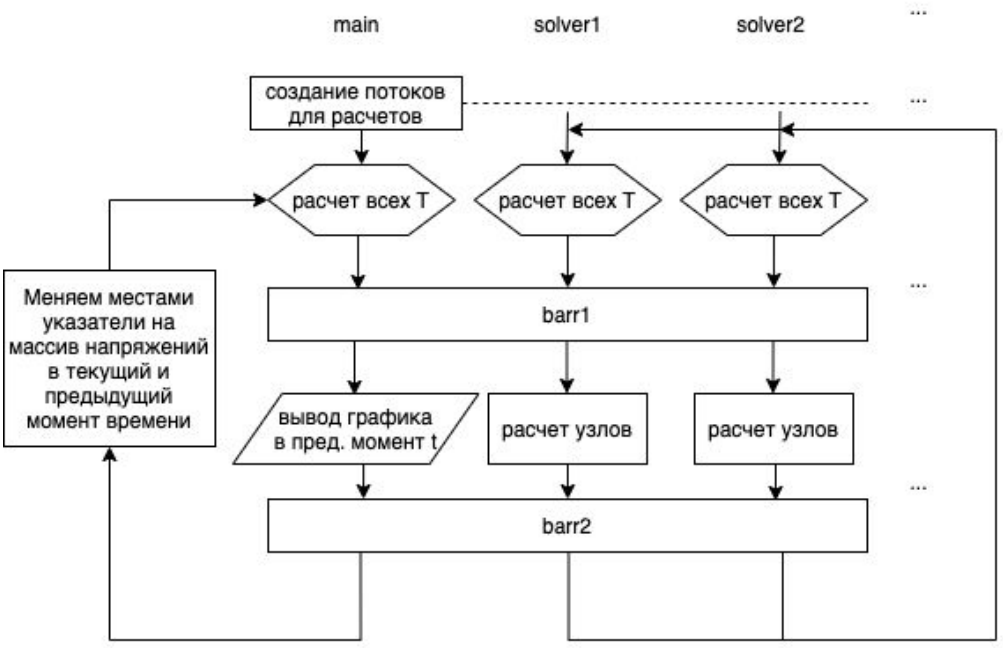


Рис 2. Схема взаимодействия потоков

Описание основных используемых структур данных

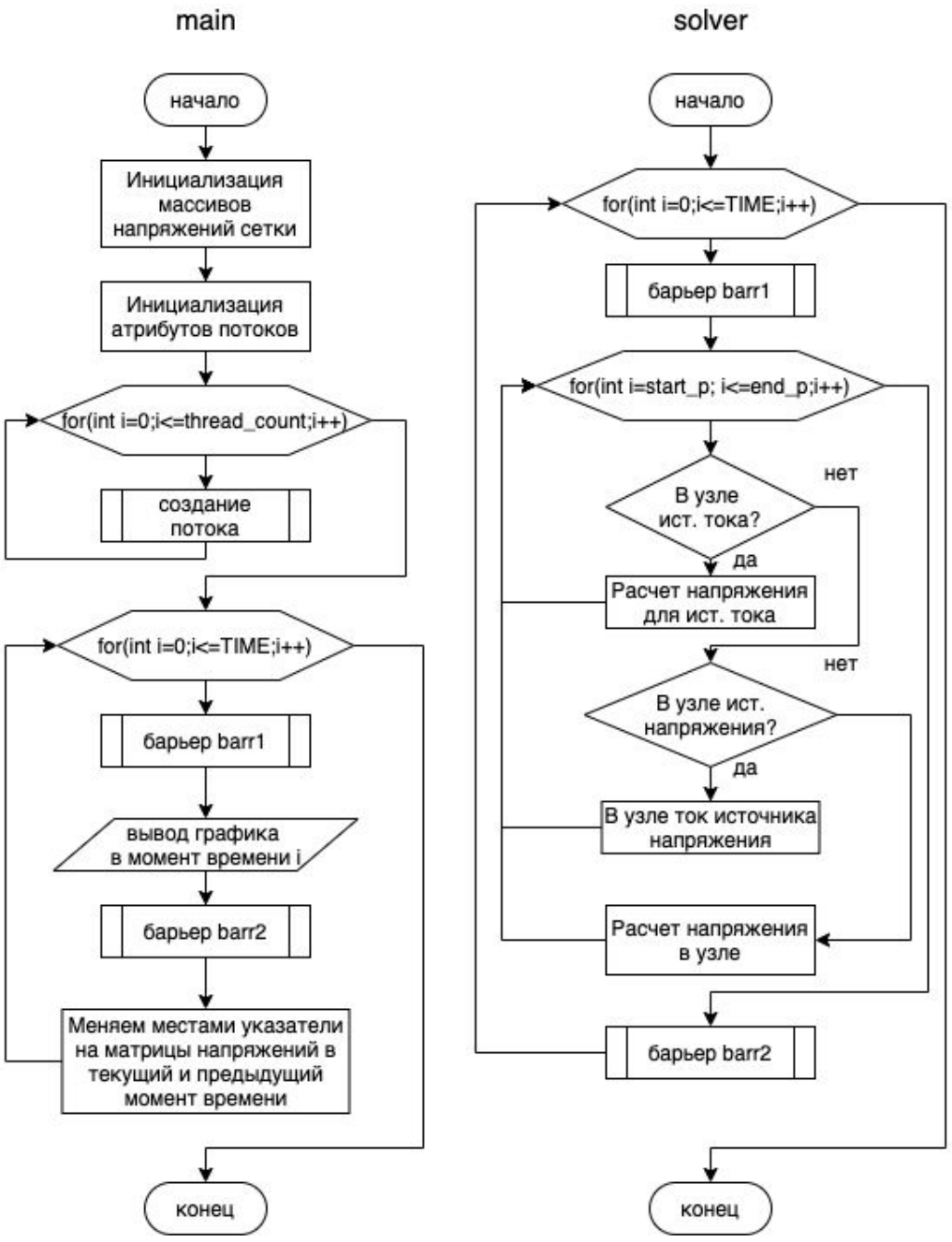
VoltS - структура данных для хранения информации об источнике напряжения. Хранит координаты узла и напряжение

CurrS - структура данных для хранения информации об источнике тока. Хранит координаты узла и силу тока

Grid - структура данных для хранения одномерных массивов напряжений в сетке (N*M) в текущий и предыдущий момент времени, хранит в 2 массивах источники тока и напряжения, их количество.

Thread - структура данных, хранящая номер первого и номер последнего узла, которые рассчитывает конкретный поток

Блок схема


main
начало
Инициализация массивов напряжений сетки
Инициализация атрибутов потоков
for(int i=0;i<=thread_count;i++)
создание потока
for(int i=0;i<=TIME;i++)
барьер barr1
вывод графика в момент времени i
барьер barr2
Меняем местами указатели на матрицы напряжений в текущий и предыдущий момент времени
конец
solver
начало
for(int i=0;i<=TIME;i++)
барьер barr1
for(int i=start_p; i<=end_p;i++)
В узле ист. тока?
нет
да
Расчет напряжения для ист. тока
нет
В узле ист. напряжения?
да
В узле ток источника напряжения
Расчет напряжения в узле
барьер barr2
конец

Примеры результатов работы программы

Пример результата запуска при M = 32, N = 16 на временном слое 1000 представлен на рис. 3

В сетке находится 2 источника тока:

В узле 3;2, 70A. В узле M-5;N-7, 150A.

И 2 источника напряжения:

В узле M-4;3, 65V. В узле 7;N-6, 110V.

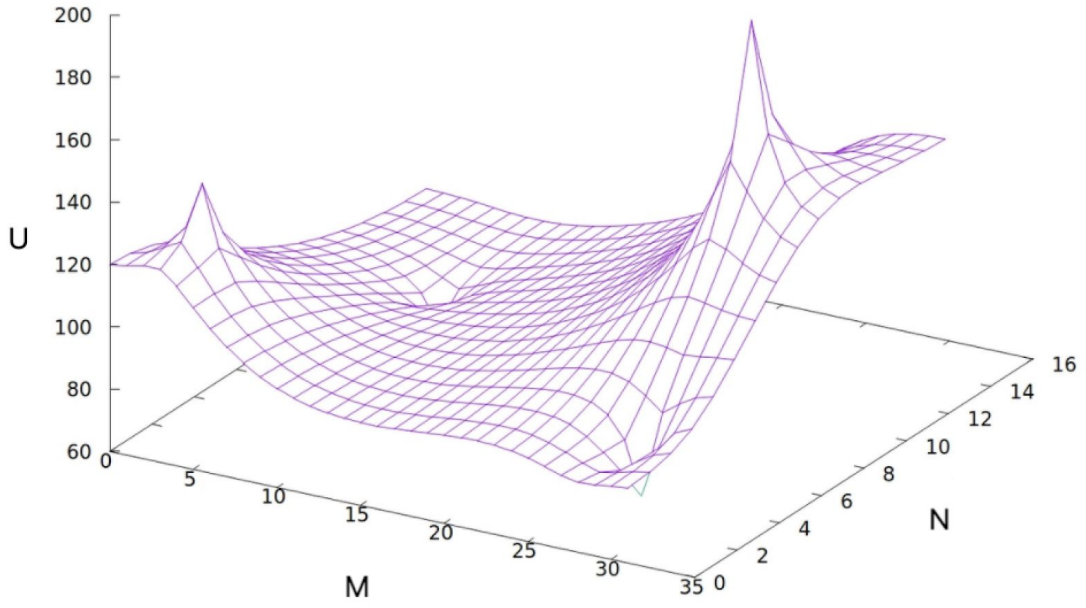


Рис. 3. Пример результатов запуска программы

Для оценки эффективности разбиения на потоки были сделаны замеры времени работы с помощью утилиты time при разном числе потоков на сетке размером 2048 на 1024 и временем работы 1024. Результаты представлены в таблице 1.

Количество потоков	real	user	sys
1	1m52,936s	1m52,852s	0m0,061s
2	0m58,003s	1m54,833s	0m0,085s
4	0m31,831s	2m0,259s	0m0,128s
8	0m20,919s	2m23,491s	0m0,189s

Таблица 1. Время работы программы при разном количестве потоков

Текст программы

main.h

```
#ifndef LAB2_MAIN_H

#define LAB2_MAIN_H


#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <sys/time.h>

#include <unistd.h>

#include <pthread.h>


#define GRAPH 0


typedef struct {

    pthread_t tid;

    int start;

    int end;

} Thread;                  //потоки


typedef struct {

    double U;

    size_t x;

    size_t y;

} VoltS;                   //структура для источников напряжения


typedef struct {
```

```
    double I;
    size_t x;
    size_t y;
} CurrS;                    //структура для источников тока

typedef struct {
    double **prev;
    double **cur;
    VoltS *volts;
    CurrS *currs;
    size_t v_count;
    size_t c_count;
} Grid;                     //структура для сетки



int is_done;
size_t M;                   //длина
size_t N;                   //ширина
double R = 1;               //сопротивление резисторов
double h = 0.1;             //шаг
double C = 0.5;             //емкость конденсаторов
Grid grid;

Thread *threads;
pthread_barrier_t barr1, barr2;

int is_ok_knot(ssize_t m, ssize_t n);
int volts_solver(size_t m, size_t n);
```

```
int currs_solver(ssize_t m, ssize_t n);

void point_solver(size_t i);

void *solver(void *arg);

int grid_inicialize(Grid *g);

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

#endif //LAB2_MAIN_H
```

main.c

```
#include "main.h"

int is_ok_knot(ssize_t m, ssize_t n) {                    //провнрка, есть ли такая ячейка
   if (m >= 0 && m < M && n >= 0 && n < N) {
      return 1;
   }
   return 0;
}

int volts_solver(size_t m, size_t n) {                    //решатель для источника напряжения
   for (size_t i = 0; i < grid.v_count; i++) {
      if (grid.volts[i].x == m && grid.volts[i].y == n) {
         grid.cur[m][n] = grid.volts->U;
         return 1;
      }
   }
   return 0;
}

int currs_solver(ssize_t m, ssize_t n) {                  //решатель для источника тока
   for (size_t i = 0; i < grid.c_count; i++) {
      if (grid.currs[i].x == m && grid.currs[i].y == n) {
         size_t k = 0;
         double sum = 0;
         if (is_ok_knot((ssize_t)m - 1, n)) {
            k++;
            sum += grid.prev[(ssize_t)m - 1][n];
         }
         if (is_ok_knot(m + 1, n)) {
            k++;
            sum += grid.prev[m + 1][n];
         }
         if (is_ok_knot(m, (ssize_t)n - 1)) {
            k++;
            sum += grid.prev[m][(ssize_t)n - 1];
         }
```

```
            if (is_ok_knot(m, n + 1)) {
                k++;
                sum += grid.prev[m][n + 1];
            }
            grid.cur[m][n] = (grid.currs[i].I * R + sum) / k;
            return 1;
        }
    }
    return 0;
}

void point_solver(size_t i) {                          //решатель для узла
    size_t m = i / N;
    size_t n = i % N;
    if (!currs_solver(m, n) && !volts_solver(m, n)) {
        size_t k = 0;
        double sum = 0;
        if (is_ok_knot((ssize_t)m - 1, n)) {
            k++;
            sum += grid.prev[(ssize_t)m - 1][n];
        }
        if (is_ok_knot(m + 1, n)) {
            k++;
            sum += grid.prev[m + 1][n];
        }
        if (is_ok_knot(m, (ssize_t)n - 1)) {
            k++;
            sum += grid.prev[m][(ssize_t)n - 1];
        }
        if (is_ok_knot(m, (n + 1))) {
            k++;
            sum += grid.prev[m][n + 1];
        }
        grid.cur[m][n] = h * (sum - k * grid.prev[m][n]) / (C * R) + grid.prev[m][n];
    }
}

void *solver(void *arg) {                              //решатель для узлов в зоне ответственности
потока
    Thread *thread = (Thread *) arg;
    while (!is_done) {
        pthread_barrier_wait(&barr1);
        for (size_t i = thread->start; i <= thread->end; i++) {
            point_solver(i);
        }
        pthread_barrier_wait(&barr2);
    }
    return NULL;
```

```
}

int grid_inicialize(Grid *g) {                          //инициализация сетки и источников
    g->prev = calloc(M, sizeof(double *));
    g->cur = calloc(M, sizeof(double *));
    g->prev[0] = (double *)calloc(M * N, sizeof(double));
    g->cur[0] = (double *)calloc(M * N, sizeof(double));
    for (size_t i = 1; i < M; i++) {
        g->cur[i] = g->cur[i - 1] + N;
        g->prev[i] = g->prev[i - 1] + N;
    }

    if (!g->prev || !g->cur) {
        printf("Alloc error\n");
        return -1;
    }

    g->v_count = 2;
    g->c_count = 2;
    g->currs = calloc(g->c_count, sizeof(CurrS));
    g->volts = calloc(g->v_count, sizeof(VoltS));
    if (!g->currs || !g->volts) {
        printf("Alloc error\n");
        return -1;
    }

    g->currs[0].x = 3;
    g->currs[0].y = 2;
    g->currs[0].I = 70;
    g->currs[1].x = M - 5;
    g->currs[1].y = N - 7;
    g->currs[1].I = 150;

    g->volts[0].x = M - 4;
    g->volts[0].y = 3;
    g->volts[0].U = 65;
    g->volts[1].x = 7;
    g->volts[1].y = N - 6;
    g->volts[1].U = 110;
    return 0;
}

int main(int argc, char **argv) {
    if (argc != 5) {
        printf("Incorrect argc count\n");
        exit(EXIT_FAILURE);
    }
```

```
size_t threads_count = atol(argv[1]);
size_t tm = atol(argv[2]);
M = atol(argv[3]);
N = atol(argv[4]);

if (grid_inicialize(&grid) < 0) {                //инициализация сетки и источников
   exit(EXIT_FAILURE);
}

FILE *gnuplot = NULL;
if (GRAPH) {
   gnuplot = popen("gnuplot -persist", "w");     //графики
   if (gnuplot == NULL) {
      printf("gnuplot error\n");
      exit(EXIT_FAILURE);
   }
}

pthread_attr_t pattr;
pthread_attr_init(&pattr);
pthread_attr_setscope(&pattr, PTHREAD_SCOPE_SYSTEM);
pthread_attr_setdetachstate(&pattr, PTHREAD_CREATE_JOINABLE);
threads = (Thread *) calloc(threads_count, sizeof(Thread));
pthread_barrier_init(&barr, NULL, threads_count + 1);

size_t k = M * N / threads_count;
if (M * N % threads_count != 0) {
   k++;
}
for (size_t i = 0; i < threads_count; i++) {
   threads[i].start = i * k;
   if (i != threads_count - 1) {
      threads[i].end = (i + 1) * k - 1;
   } else {
      threads[threads_count - 1].end = M * N - 1;
   }

   if (pthread_create(&(threads[i].tid), &pattr, solver, (void *) &(threads[i]))) {
      printf("Creating thread error\n");
      exit(EXIT_FAILURE);
   }
}

is_done = 0;
for (size_t i = 0; i < tm; i++) {
   pthread_barrier_wait(&barr1);                 //барьер для ожидания потоков
   if (GRAPH) {
      if (!is_done) {
```

```
            fprintf(gnuplot, "set dgrid3d %zu,%zu\n", N, M);
            fprintf(gnuplot, "set mxtics (1)\n");
            fprintf(gnuplot, "set mytics (1)\n");
            fprintf(gnuplot, "set ticslevel 0\n");
            fprintf(gnuplot, "set hidden3d\n");
            fprintf(gnuplot, "set isosample 80\n");
            fprintf(gnuplot, "set zlabel \"U\"\n");
            fprintf(gnuplot, "set ylabel \"M\"\n");
            fprintf(gnuplot, "set xlabel \"N\"\n");
            fprintf(gnuplot, "splot '-' u 1:2:3 with lines\n");

            for (size_t x = 0; x < M; x++) {
               for (size_t y = 0; y < N; y++) {
                  fprintf(gnuplot, "%zu %zu %lf\n", x, y, grid.cur[x][y]);
               }
            }
            fprintf(gnuplot, "e\n");
            fflush(gnuplot);
         }
      }
      pthread_barrier_wait(&barr2);
      double **tmp = grid.prev;                    //меняем prev и cur для новой итерации
      grid.prev = grid.cur;
      grid.cur = tmp;
   }
   if (GRAPH) {
      pclose(gnuplot);
   }
   return 0;
}
```