Министерство образования и науки 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 №2

Студент	Журавлев Николай Вадимович
Группа	РК6-62б
Тип задания	Лабораторная работа
Тема лабораторной работы	Многопоточное программирование

Студент _____ **Н.В. Журавлев**
подпись, дата *фамилия, и.о.*

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

Оценка _____

Москва, 2022 г.

Оглавление

Текст задания на лаб. работу..... 3

Описание структуры программы и реализованных способов взаимодействия потоков управления..... 3

Описание основных используемых структур данных..... 4

Блок-схема программы..... 5

Примеры результатов работы программы..... 6

Текст программы..... 7

Текст задания на лаб. работу.

Разработать многопотоковый вариант программы моделирования распространения электрических сигналов в двухмерной прямоугольной сетке RC-элементов (одномерный аналог которых представлен на рис. выше). 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. При этом утилита gnuplot должна вызываться отдельной командой после окончания расчета.

Описание структуры программы и реализованных способов взаимодействия потоков управления

В начале выполнения функции main выделяется память под расчётные массивы для хранения текущего времени вычисления и предыдущего. Затем создаётся определённое количество потоков, полученное из аргументов командной строки. Каждый поток, получает определённое количество столбцов узлов сетки в зависимости от размерности сетки и количества потоков, затем это заносится в специальную структуру для передачи этих данных в потоки. Затем создаются сами потоки, после чего запускается таймер. Затем основной поток и остальные останавливаются барьером 1 перед выполнением вычислений. Затем нужное количество раз, в зависимости от выбранного времени, проводятся вычесления по следующему принципу: основной поток ждёт завершения дочерних, т.к. дошёл до барьера 2, а дочерние потоки заполняют массив cur, который заполняется по следующей формуле 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. Для аппроксимации производной по времени используется выражение $dV^i_j/dt=(V^{i+1}_j-V^i_j)/h$. После все потоки доходят до барьера 2 и основной поток выполняет

копирование результатов в массив prev и сохранение результатов в файл. И так продолжается до завершения цикла по времени. После этого таймер останавливается и выводится время расчётов.

Описание основных используемых структур данных

Задействованы переменные: количества узлов в длину M и ширину N; флага завершения работы потоков управления done, количество потоков nthread.

Используются одномерные массивы для хранения значений напряжений в узлах: в текущий момент времени cur, в предыдущий момент времени.

Также используется структура ThreadRecord:

```
typedef struct {
pthread_t tid; //идентификатор потока управления
int first; //номер первого узла, лежащего на левой границе расчетной
ленты данного потока
int last; //номер первого узла, лежащего на правой границе расчетной
ленты данного потока
} ThreadRecord;
```

Блок-схема программы

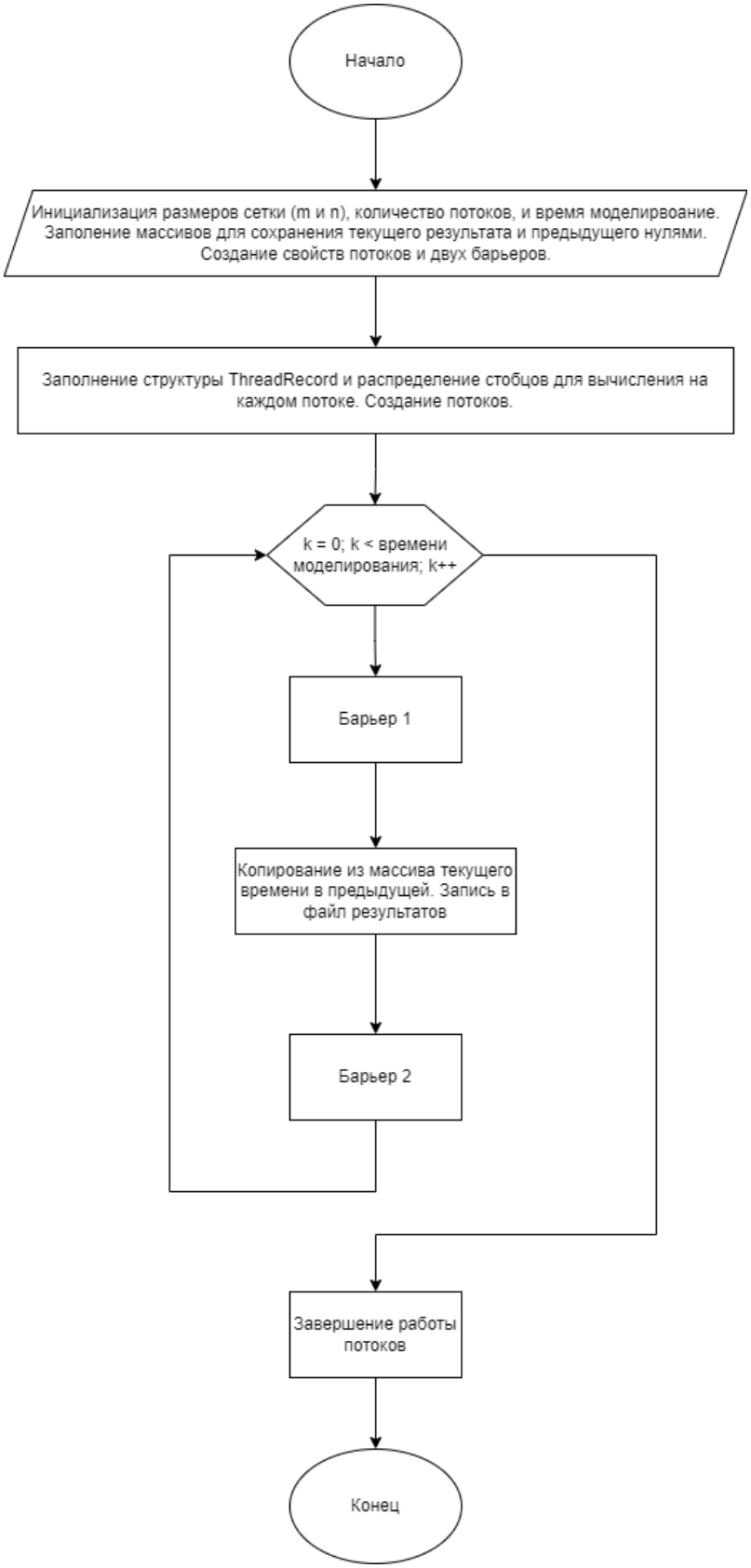

Рисунок 1. Блок-схема основного потока

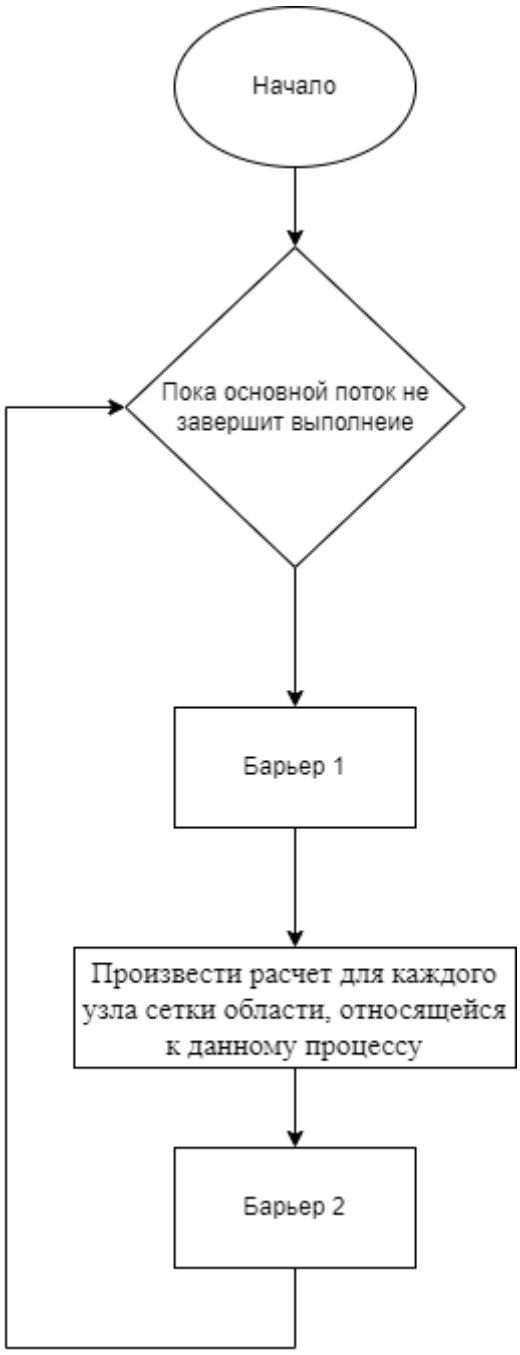

Рисунок 2. Блок схема функции, дочерних потоков

Примеры результатов работы программы

Результаты работы программы представлены в графическом виде с помощью стандартной UNIX-утилиты gnuplot - рис.3.

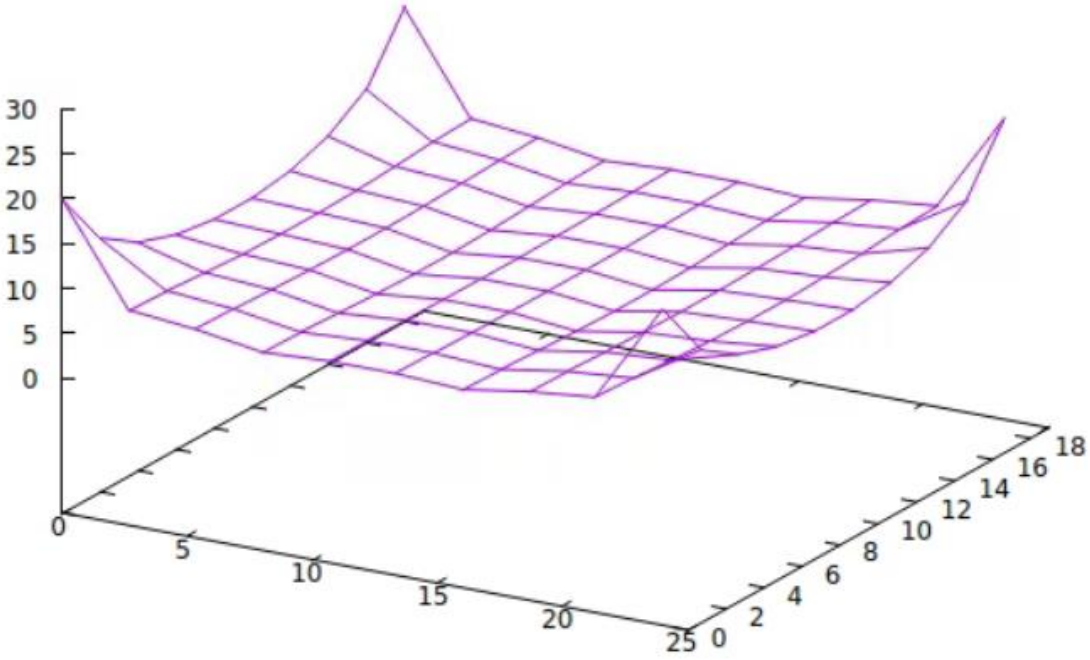

Рисунок 3. Результат работы программы для аргументов 25 19 1000

Таблица временных затрат – таб.1.

Таблица 1

Количество потоков	Время работы программы (ms)
1	109891
2	61563
4	38386
8	34104

Текст программы

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <sys/time.h>
#define U 20.0
#define C 1
```

```
#define R 2.5
#define H 0.2

typedef struct {
    pthread_t tid;
    int first;
    int last;
} ThreadRecord;

int M, N;
double *cur, *prev;
int done = 0;
ThreadRecord *threads;
pthread_barrier_t barr1, barr2;

void * mysolver(void *arg_p) {
    ThreadRecord *thr;
    int i, j;
    double tmp;
    thr = (ThreadRecord *)arg_p;
    while (!done) {
        pthread_barrier_wait(&barr1);
        for (i = thr->first; i <= thr->last; i++) {
            for (j = 0; j < N; j++) {
                int count;
//Далее - условия вычислений для четырех концов сетки RC-элементов
                if (j == 0 && i == 0) {
                    cur[i * N + j] = U;
                    continue;
                } else if (j == N - 1 && i == 0) {
```

```
                    cur[i * N + j] = U;
                    continue;
                } else if (j == 0 && i == M - 1) {
                    cur[i * N + j] = U;
                    continue;
                } else if (j == N - 1 && i == M - 1){
                    cur[i * N + j] = U;
                    continue;
                } else if (j == 0) {
                    // Нижняя (ближняя)
                    tmp = prev[(i - 1) * N + j] + prev[i * N + j + 1] + prev[(i + 1) * N +
j];
                    count = 3;
                } else if (j == N - 1) {
                    // Верхния (дальняя)
                    tmp = prev[i * N + j - 1] + prev[(i - 1) * N + j] + prev[(i + 1) * N +
j];
                    count = 3;
                } else if (i == 0) {
                    // Левая
                    tmp = prev[i * N + j - 1] + prev[i * N + j + 1] + prev[(i + 1) * N +
j];
                    count = 3;
                } else if (i == M - 1) {
                    // Правая
                    tmp = prev[i * N + j - 1] + prev[(i - 1) * N + j] + prev[i * N + j + 1];
                    count = 3;
                } else {
                    tmp = prev[i * N + j - 1] + prev[(i - 1) * N + j] + prev[i * N + j + 1]
+ prev[(i + 1) * N + j];
```

```
                    count = 4;
                }
                cur[i * N + j] = H * tmp / C / R + prev[i * N + j] * (1 - count * H / C /
R);
            }
        }
        pthread_barrier_wait(&barr2); //Ожидание завершения копирования
функцией main
    }
    return NULL;
}

int main(int argc, char* argv[]) {
    int nthread;
    int ntime;
    int i, j, k;
    pthread_attr_t pattr;

    if (argc != 5)
        exit(1);

    M = atoi(argv[1]);
    N = atoi(argv[2]);
    nthread = atoi(argv[3]);
    ntime = atoi(argv[4]);

    if (M % nthread)
        exit(2);

    cur = (double*)malloc(sizeof(double) * M * N);
```

```
prev = (double*)malloc(sizeof(double) * M * N);

memset(cur, 0, sizeof(double) * M * N);
memset(prev, 0, sizeof(double) * M * N);

pthread_attr_init(&pattr);
pthread_attr_setscope(&pattr, PTHREAD_SCOPE_SYSTEM);
pthread_attr_setdetachstate(&pattr, PTHREAD_CREATE_JOINABLE);
threads = (ThreadRecord *)calloc(nthread, sizeof(ThreadRecord));
pthread_barrier_init(&barr1, NULL, nthread + 1);
pthread_barrier_init(&barr2, NULL, nthread + 1);
j = M / nthread;

for (i = 0; i < nthread; i++) {
   threads[i].first = j * i;//установка номера первого узла левой границы ленты каждого потока

   threads[i].last = j * (i + 1) - 1;//установка номера первого узла правой границы ленты каждого потока

   if (pthread_create(&(threads[i].tid), &pattr, mysolver, (void *) &(threads[i])))//создание потоков управления
      perror("thread not create");
}

struct timeval tv1, tv2, dtv;
struct timezone tz;
gettimeofday(&tv1, &tz);
FILE *fd = fopen("f.plt", "w");
```

pthread_barrier_wait(&barr1);//"Начало расчетов" - ожидание создания всех потоков управления

for (k = 0; k < ntime; k++) { //синхронизированное с расчетами потоков управления выполнение копирования значений "текущих" и "предыдущих" напряжений между собой и в результирующую матрицу

```
    pthread_barrier_wait(&barr2);

    fprintf(fd, "set dgrid3d\n");
    fprintf(fd, "set xrange[0:16]\n");
    fprintf(fd, "set yrange[0:16]\n");
    fprintf(fd, "set zrange[0:30]\n");
    fprintf(fd, "splot '-' using 1:2:3 with lines\n");
    for (i = 0; i < M; i++) {
      for (j = 0; j < N; j++) {
        fprintf(fd, "%d %d %lf\n", i, j, cur[i * N + j]);
      }
    }
    fprintf(fd, "e\n");
    fprintf(fd, "\n");

    memcpy(prev, cur, sizeof(double) * M * N);
    memset(cur, 0, M * N);
    pthread_barrier_wait(&barr1);
  }

  done = 1;

  gettimeofday(&tv2, &tz);
  dtv.tv_sec = tv2.tv_sec - tv1.tv_sec;
```

```
    dtv.tv_usec = tv2.tv_usec - tv1.tv_usec;

    if (dtv.tv_usec < 0) {
        dtv.tv_sec--;
        dtv.tv_usec += 1000000;
    }

    printf("%d s %ld ms\n", (int)dtv.tv_sec, dtv.tv_usec / 1000);
    fclose(fd);
    return 0;
}
```