|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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](#_Toc110605650)

[Описание структуры программы и реализованных способов взаимодействия потоков управления 3](#_Toc110605651)

[Описание основных используемых структур данных 4](#_Toc110605652)

[Блок-схема программы 5](#_Toc110605653)

[Примеры результатов работы программы 6](#_Toc110605654)

[Текст программы 7](#_Toc110605655)

# 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 значений узлов для текущего времени вычисления и предыдущего. Затем создаётся определённое количество потоков, полученное из аргументов командной строки. Каждый поток, получает определённое количество строк узлов сетки в зависимости от размерности сетки и количества потоков (рис.1), 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 вычисления 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 i,j-ого узла цепочки уравнение баланса токов имеет вид IRлев-IRправ+IRверх-IRниж -IC=0 или (Vki,j-1-Vki,j)/R-(Vki,j-Vki,j+1)/R-(Vkj-Vki-1,j)/R +(Vkj-Vki+1,j)/R-C\*dVk,i,j/dt=0, где k - номер врем**е**нного шага, V - потенциал узла. Для аппроксимации производной по времени используется выражение dVki,j/dt=(Vk+1i,j-Vki,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.

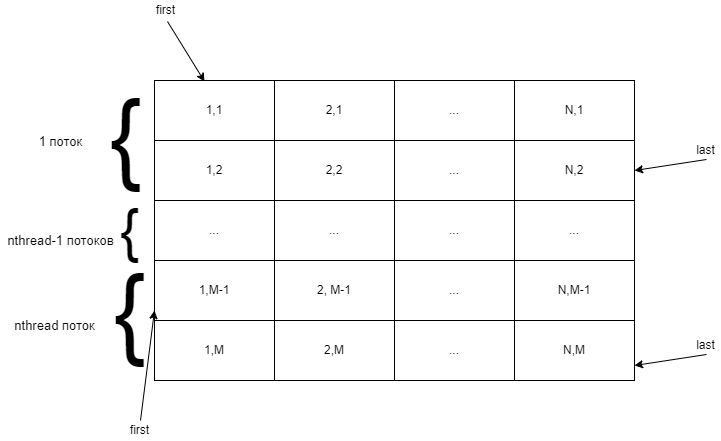


Рисунок . Рисунок деления узлов двухмерной прямоугольной сетки на потоки

# 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, в которой:

pthread\_t tid - идентификатор потока управления

int first - номер строки в массиве cur, с которой текущий поток должен начать вычисления.

int last - номер строки в массиве cur, на которой текущий поток должен закончить вычисления.

# Блок-схема программы

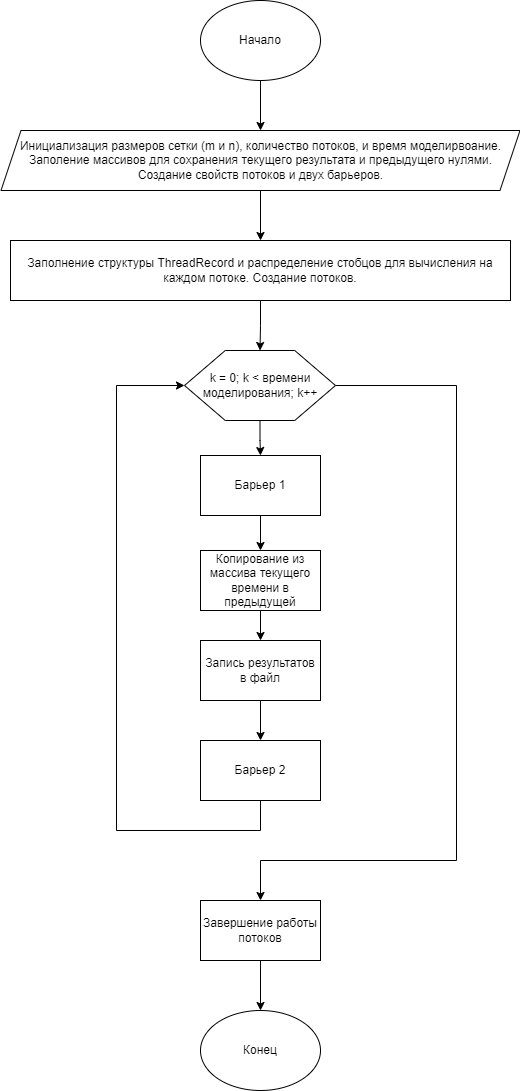


Рисунок . Блок-схема основного потока

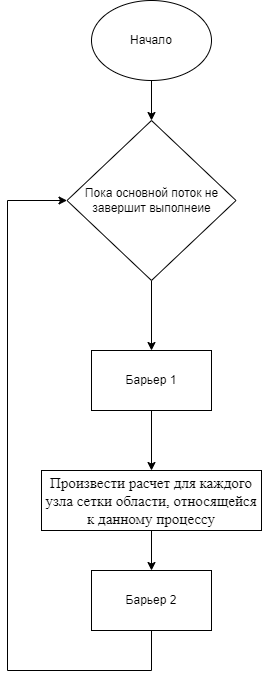


Рисунок . Блок схема функции, дочерних потоков

# Примеры результатов работы программы

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

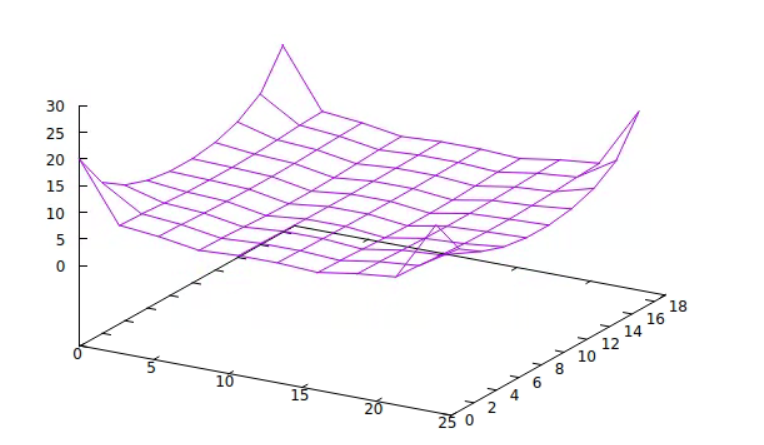


Рисунок 4. Результат работы программы для аргументов 25 19 1000

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 строки в массиве cur, с которой текущий поток должен начать вычисления.

threads[i].last = j \* (i + 1) - 1;//установка номера строки в массиве cur, с которой текущий поток должен закончить вычисления.

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;

}