## Текст задания на лабораторную работу

Разработать, используя средства многопотокового программирования, параллельную программу решения уравнения струны методом конечных разностей с использованием *явной* вычислительной схемы. Количество потоков, временной интервал моделирования и количество (кратное 8) узлов расчетной сетки - параметры программы. Уравнение струны имеет следующий вид:

$$d^2z/dt^2 = a^2*d^2z/dx^2+f(x,t)$$

, где t - время, x - пространственная координата, вдоль которой ориентирована струна, z - отклонение (малое) точки струны от положения покоя, a - фазовая скорость, f(x,t) - внешнее "силовое" воздействие на струну. Предусмотреть возможность задания ненулевых начальных условий и ненулевого внешнего воздействия. Программа должна демонстрировать ускорение по сравнению с последовательным вариантом. Предусмотреть визуализацию результатов посредством утилиты gnuplot.

## Описание структуры программы

В программе динамически выделяется память под узлы сетки (в текущий и предыдущий моменты времени). Инициализируются атрибуты потоков и инициализируются барьеры. Выделяется память и инициализуются пользовательские структуры данных, хранящие ID потока и индексы расчетных узлов данного потока. Создаются потоки отвечающие за расчеты. Потоки разделяют набор расчетных узлов сетки и производят расчет очередного временного слоя. Синхронизация осуществляется путем использования барьеров. Вывод результатов на каждой итерации записывается в файл. Визуальное отображение работы программы реализовано посредством утилиты gnuplot.

## Описание основных структур данных

В программе используются пользовательский тип данных *struct ThreadRecord.*

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

Где *pthread_t tid* - идентификатор потока. *first, last* - первый и последний индексы расчётных узлов данного потока. Данная структура содержит информацию о работе расчетных потоков. Весь кооператив потоков характеризуется массивом указателей на переменные типа *struct ThreadRecord.*

Для того, чтобы вычислить какие узлы будут находиться в каком потоке, вводится переменная *j*. *(int j = (node - 2) / nt)*, где *node* это количество узлов а *nt* количество потоков. *j* необходима для вычисления индексов узлов, которые будут распределяться по потокам.

Также в программе используются следующие стандартные структуры данных:

- *struct timeval* – структура данных, в которую записывается информация о времени;
- *struct timezone* - структура данных, содержащая информацию о сезонной коррекции времени
- *pthread_attr_t* – структура данных в которую записываются параметры создаваемых потоков;
- *pthread_barrier_t* – тип данных для барьеров – средство синхронизации процессов;
- *pthread_t* – тип данных, хранящий информацию о созданных потоках
- *FILE* - указатель на управляющую таблицу открытого потока данных, если открытие файла произошло успешно.

- Z - двумерный массив чисел, в котором хранятся значения уравнения. Длина массива по оси X равна количеству узлов струны. В строках массива хранятся координаты узлов струны для текущего и предыдущего временного слоя. Координаты которые считались текущим временным слоем на текущем шаге, на следующем шаге по времени считаются координатами прошлого временного слоя. На каждом шаге по времени в строку с индексом cur записываются координаты рассчитанные на данном шаге.

| Current_time, сек | Индекс cur | Индекс prev |
|---|---|---|
| 1 | 1 | 0 |
| 2 | 0 | 1 |
| 3 | 1 | 0 |
| 4 | 0 | 1 |

- *nt* - количество потоков, *nc* - временной интервал, *node* - количество узлов

## Блок-схема программы

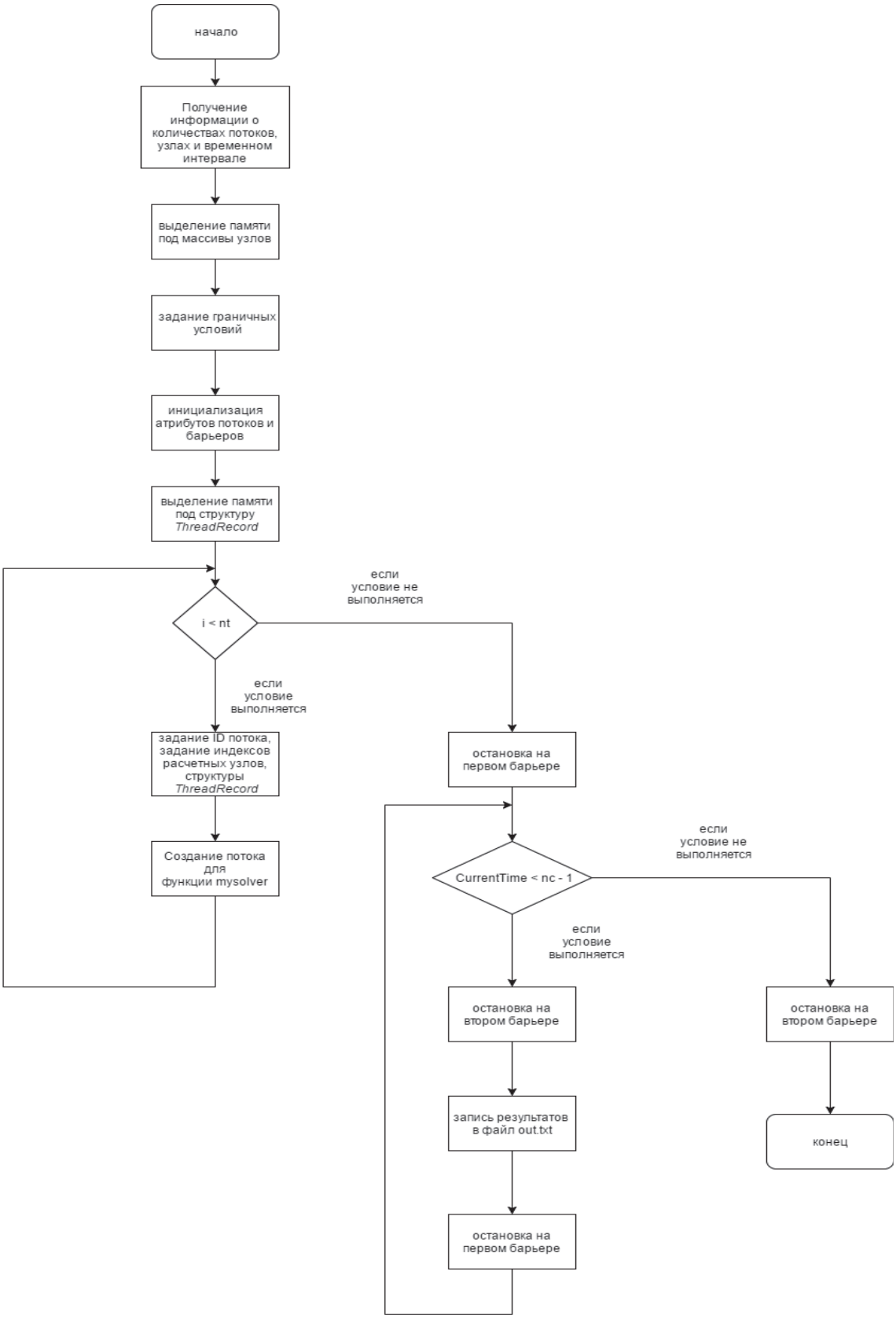


*Рис. 6. Блок-схема функции-потока main.*

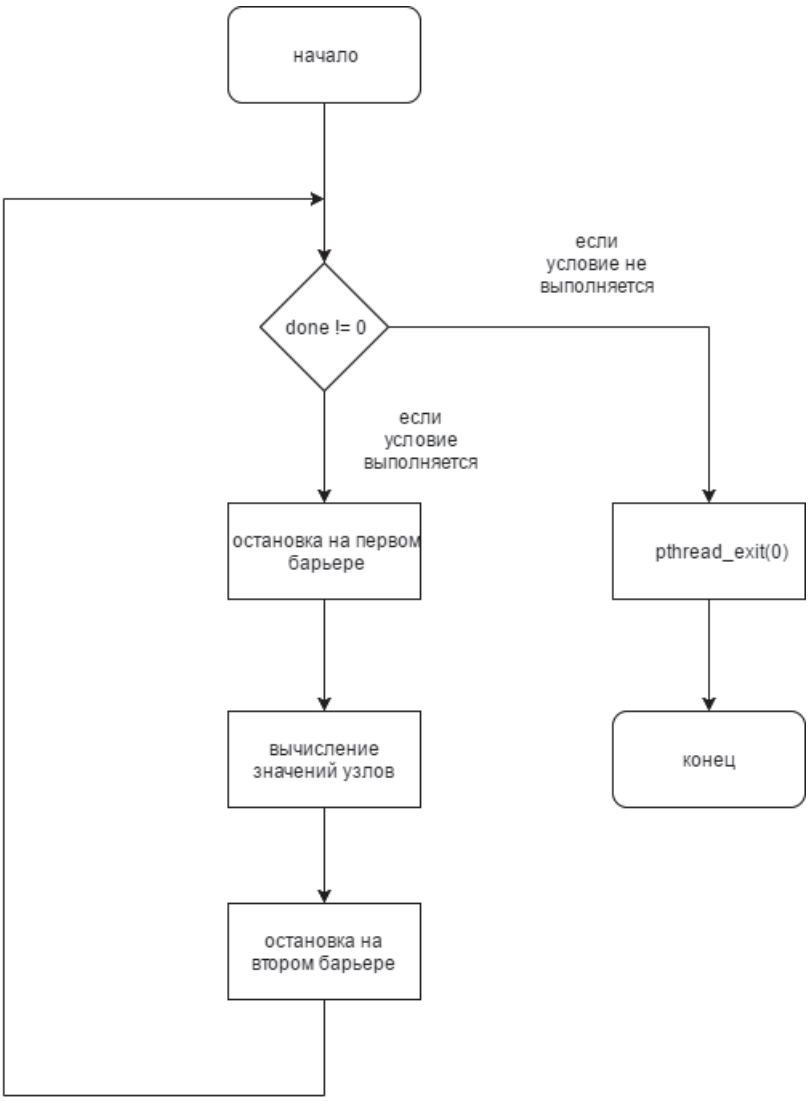


*Рис. 7. Блок-схема расчетной функции-потока.*

## Примеры результатов работы программы

Результат работы программы продемонстрированы посредством утилиты *gnuplot*. Количество узлов = 82, модельное время = 100, шаг по времени: 1.

Слева и справа заданы граничные условия первого рода (значение фазовой переменной — 0).

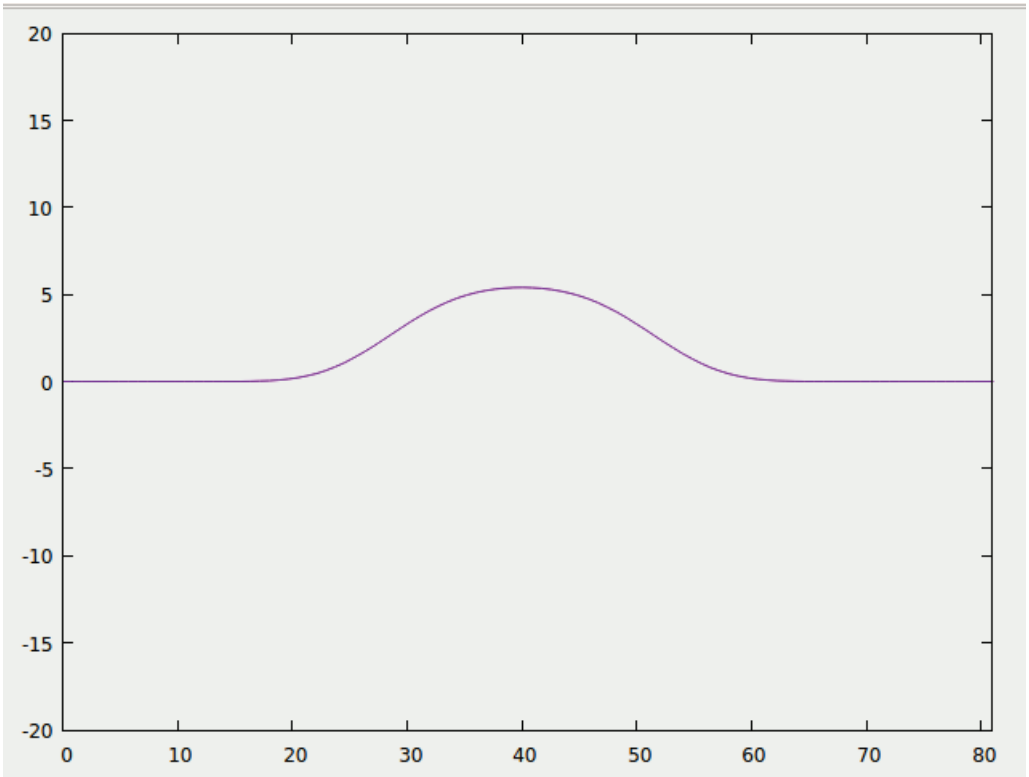


*Рис. 8. Результат работы программы с нулевыми начальными условиями и внешней силой приложенной в центре струны.*

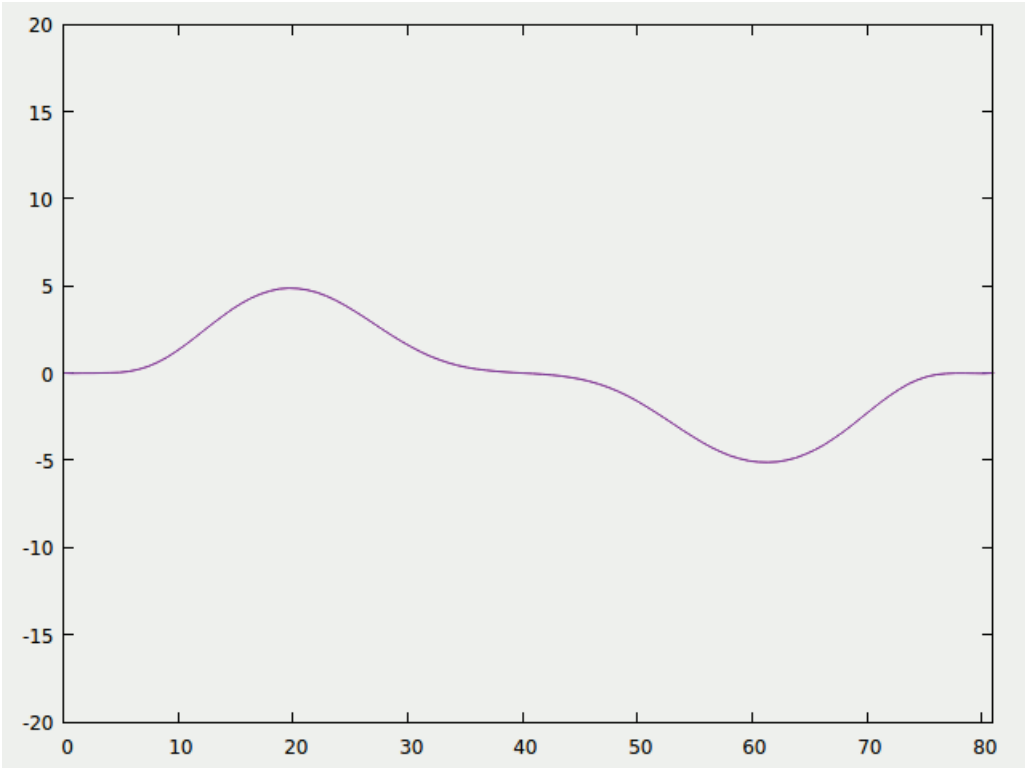


*Рис. 9. Результат работы программы с силой действующей вверх в 1 четверти струны и действующей вниз в 3 четверти.*

## Ускорение программы

В лабораторных условиях проведен эксперимент для проверки ускорения вычислений в зависимости от количества параллельно выполняющихся потоков.

Количество узлов: 1000000
Количество временных интервалов: 1000

| Количество потоков | Время выполнения, с |
|---|---|
| 1 | 130,101 |
| 2 | 66,067 |
| 4 | 37,54 |
| 8 | 21,325 |

*Таблица. 1. Таблица временных затрат.*

## Текст программы

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <pthread.h>
#include <sched.h>
#include <sys/time.h>

FILE *out,*fp;

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

struct timeval tv1;
struct timezone tz;

int node,nc;
int done = 0;
int precision = 1;
int CurrentTime = 0;
double **Z;
double A = 0.9;
double DELTA_T = 1.0;
double DELTA_X = 1.0;
ThreadRecord *threads;
pthread_barrier_t barr1, barr2;

int timeOfWork() {
    struct timeval tv2,dtv;
    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;
    }
    return dtv.tv_sec * 1000 + dtv.tv_usec / 1000;
}

void writeIntoFile(int t) {
    int j;
        for(j = 0; j < node; j++)
            fprintf(out,"%d\t%f\n",j,Z[t][j]);
        fprintf(out, "\n\n");
}
```

```
void* mysolver (void *arg_p) {
    ThreadRecord* thr;
    thr = (ThreadRecord*) arg_p;
    int i, cur, prev;
    double f; double f2;
    while ( !done ) {
        pthread_barrier_wait(&barr1);
        cur = CurrentTime % 2;
        prev = (CurrentTime + 1) % 2;
        for (i = thr->first; i <= thr->last; i++) {
            if ( i == (node-2) / 2 - (node-2) / 4 && CurrentTime < 10 )
                f = 1;
            else
                f = 0;
                 if ( i == (node-2) / 2 + (node-2) / 4 && CurrentTime < 10
)
                f2 = -1;
            else
                f2 = 0;
            Z[cur][i] = DELTA_T*DELTA_T*( A*A*((double)(Z[prev][i-1] -
                        2*Z[prev][i]+Z[prev][i+1]) / (DELTA_X*DELTA_X )) +
                        f + f2) + 2*Z[prev][i] - Z[cur][i];
        }
        pthread_barrier_wait(&barr2);
    }
    pthread_exit(0);
}


void calculateFisrtTime(int nc) {
    int i;
    for(i = 1; i < node-1; i++)
        Z[0][i] = Z[1][i] = 0;
    for(i = 0; i < 2; i++)
        writeIntoFile(i);
}

int main (int argc, char* argv[]) {

    if (argc != 4) {
        fprintf(stderr,"./a.out потоки интервал узлы\n", argv[0]);
        exit (1);
    }
    int nt = atoi(argv[1]); // количество потоков
    nc = atoi(argv[2]);     // временной интервал
    node = atoi(argv[3]);   // количество узлов

    if ( (node-2) % nt != 0 ) {
        printf("Must divide: %d\n",(node-2));
        scanf("%d",&nt);
    }

    if ( nt >= node-2 ) nt = node-2;
```

```
    out = fopen("out.txt","w");
    //открытие файла для графического представления в gnuplot.
    fp = fopen("vgraph0.dat","w");
    fprintf(fp, "set cbrange [0.9:1]\n");
    fprintf(fp, "set xrange[0:%d]\n", node-1);
    fprintf(fp, "set yrange[-50:50]\n");
    fprintf(fp, "do for [i=0:%d]{\n", nc - 1);
    fprintf(fp, "plot 'out.txt' index i using 1:2 smooth bezier\n");
    fprintf(fp, "pause 0.05}\npause -1\n");

    int i, k;
    Z = (double **) malloc (2 * sizeof(double*));

    for(i = 0; i < 2; i++)
        Z[i] = (double *) malloc (node * sizeof(double));

    calculateFisrtTime(nc);

    pthread_attr_t pattr;
    pthread_attr_init (&pattr);
    pthread_attr_setscope (&pattr, PTHREAD_SCOPE_SYSTEM);
    pthread_attr_setdetachstate (&pattr, PTHREAD_CREATE_JOINABLE);

    threads = (ThreadRecord *) calloc (nt, sizeof(ThreadRecord));

    pthread_barrier_init(&barr1, NULL, nt+1);
    pthread_barrier_init(&barr2, NULL, nt+1);

    int j = (node) / nt;
    for (i = 0, k = 1; i < nt; i++, k += j) {
        threads[i].first = k;
        threads[i].last = k + j - 1;
        if ( pthread_create (&(threads[i].tid), &pattr, mysolver,
                                                    (void *)
&(threads[i])) )
            perror("pthread_create");
    }

    gettimeofday(&tv1, &tz); // засекаем время

    for (CurrentTime = 0; CurrentTime < nc; CurrentTime++) {
        printf("%d\n", CurrentTime);
        pthread_barrier_wait(&barr1);
        if (CurrentTime >= nc)
            done = 1;
        pthread_barrier_wait(&barr2);
        writeIntoFile(CurrentTime % 2);
    }

    writeIntoFile(CurrentTime % 2);
    printf("Time: %d milliseconds\n", timeOfWork());

    return 0;
}
```