

КАФЕДРА Системы автоматизированного проектирования (РК-6)

по дисциплине: «Разработка программных систем»

Вариант лабораторной работы 19

Оценка _____

Москва, 2022 г.

ОГЛАВЛЕНИЕ

Текст задания	3
Описание структуры программы.....	3
Описание основных структур данных	4
Блок-схема.....	6
Пример работы программы.....	7
Ускорение программы	8
Текст программы.....	9

Текст задания

Разработать, используя средства многопоточного 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:

$$\frac{d^2 z}{dt^2} = a^2 * \frac{d^2 z}{dx^2} + f(x, t)$$

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`. При этом утилита `gnuplot` должна вызываться отдельной командой после окончания расчета.

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 инициализируются 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 *int j*, равная: $(node - 2) / nt$, где *node* – это количество узлов, *nt* – количество потоков, *j* необходимо 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;
- *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.

Блок-схема

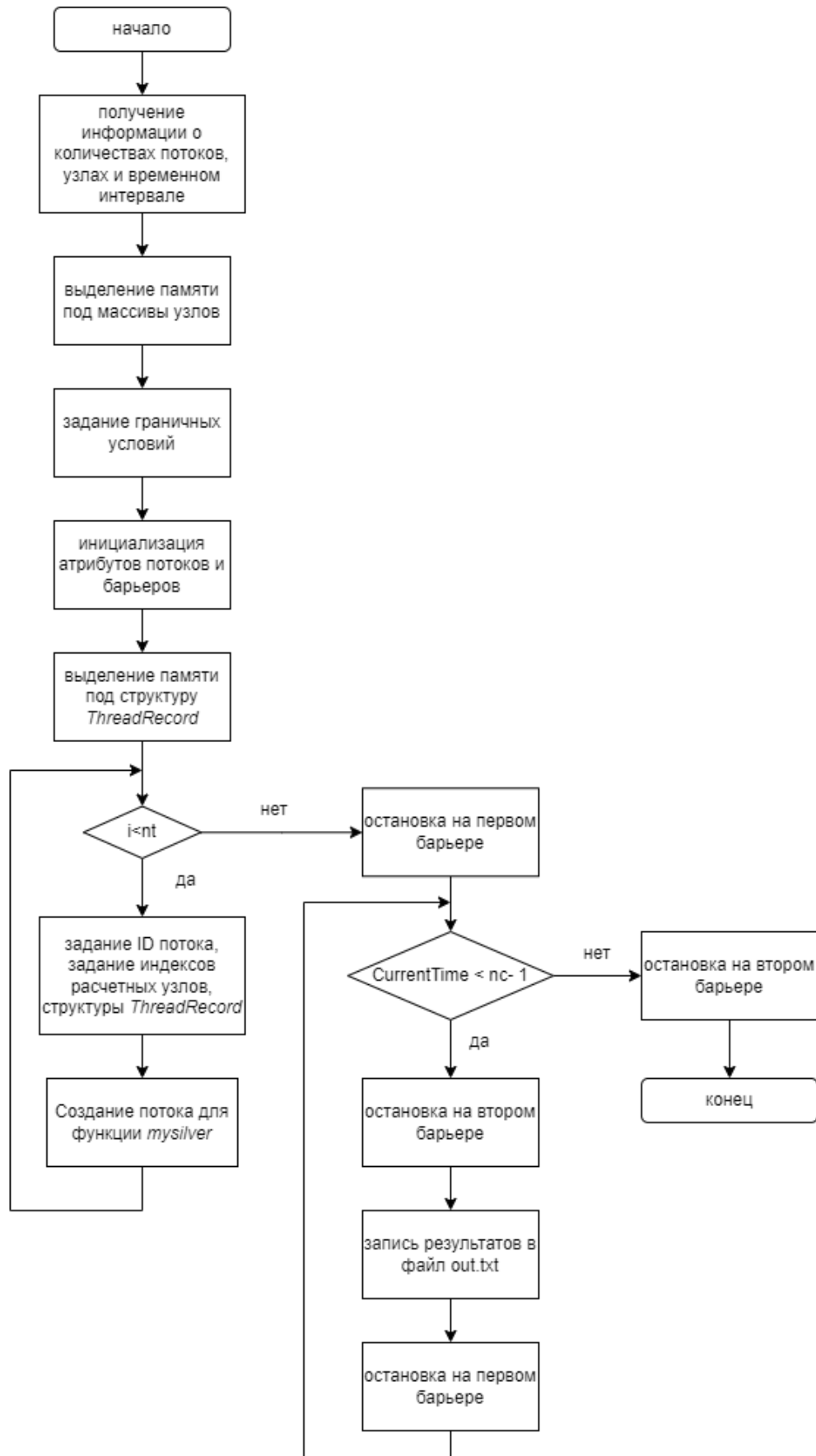


Рис. 1. Блок-схема функции-потока *main*

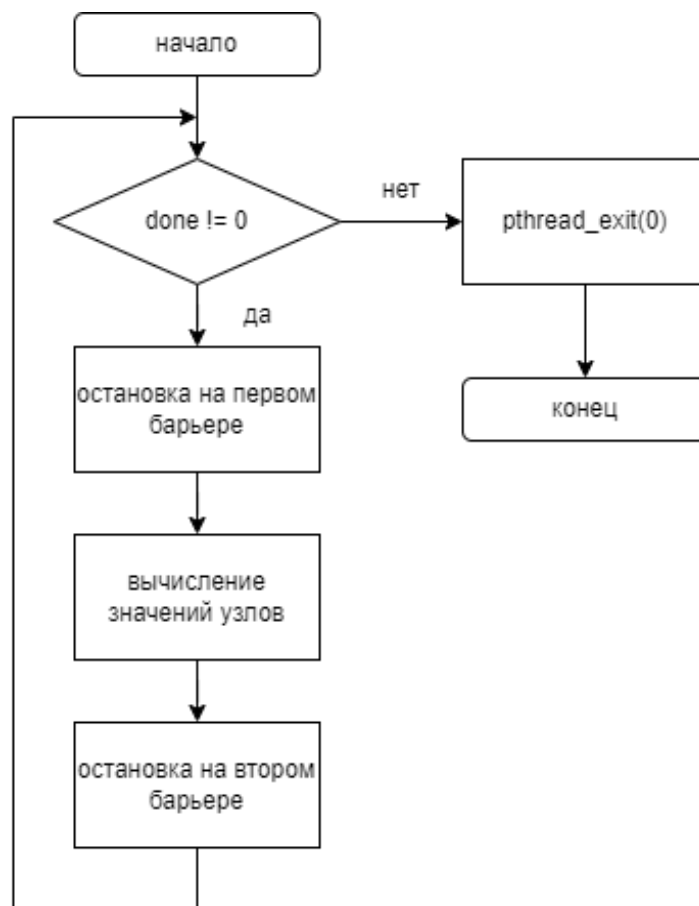


Рис. 2. Блок-схема расчётной функции-потока

Пример работы программы

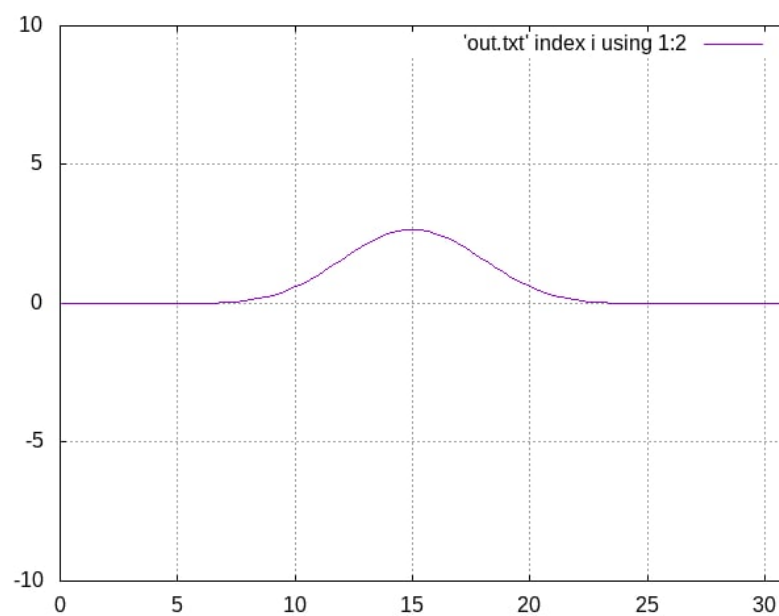


Рис. 3. Сила приложена по центру струны

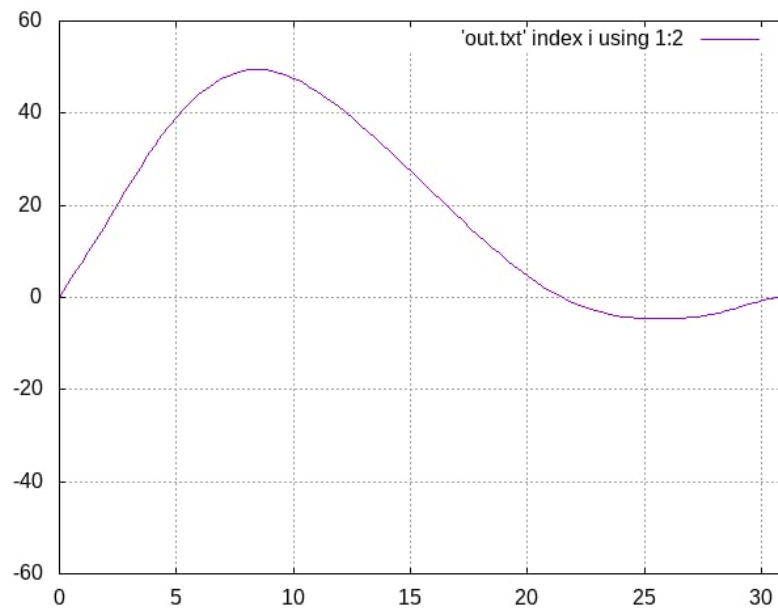


Рис. 4. Приложено две силы: первая – слева от центра струны, вторая – справа

Ускорение программы

Проведён эксперимент для проверки ускорения вычислений в зависимости от кол-во используемых потоков (временной интервал равен 100, кол-во узлов – 80.000.000):

Кол-во потоков	Время расчётов, мс	Прирост производительности, %
1	92870	-
2	47060	197
4	25041	371
8	20611	451
16	21676	428

Таблица 1. Таблица производительности

По таблице 1 видно, что при увеличении количества потоков до 8, наблюдается существенный прирост производительности. Это обусловлено тем,

что потоки успешно распараллеливаются, благодаря чему скорость вычислений растет.

При увеличении количества потоков до 16, производительность падает. Это связано с тем, что при распараллеливании тратятся дополнительные ресурсы на поддержку параллельной обработки, в частности, на переключение контекста процессора между потоками. Получается, дополнительные затраты появились, а потоки все равно не работают параллельно.

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>
#define F(x, t) 10.0
#define DELTA_X 1.0
#define DELTA_T 1.0
#define A 1.0
FILE *fp, *out;
typedef struct
{
    pthread_t tid;
    int begin;
    int end;
} ThreadInfo;
double **z;
double z_max = 0;
int node, nc;
int done = 0;
int CurrentTime = 0;
pthread_barrier_t barr1, barr2;
```

```

int TimeOfWork();

void WriteFile(int t, FILE *f);

void WriteGnuplotFile(FILE *out, int nodes, int time_interval, int max_elem);

void *mysolver(void *arg_p);

void CalculateFirstTime(int nc);


int main(int argc, char *argv[])
{
    if (argc != 4)
    {
        fprintf(stderr, "%s <Кол-во потоков> <Временной интервал> <Кол-во узлов>\n", argv[0]);
        exit(1);
    }

    // Количество потоков
    int nt = atoi(argv[1]);

    // Временной интервал
    nc = atoi(argv[2]);

    // Количество узлов
    node = atoi(argv[3]);

    printf("Input data:\nThreads = %d,\nTime = %d,\nNodes = %d\n", nt, nc, node);

    while (node % 8 != 0)
    {
        printf("<Nodes>: %d must divide by 8\n", node);
        printf("Enter <Nodes>: ");
        scanf("%d", &node);
    }

    while (node % nt != 0)
    {
        printf("<Nodes>: %d must divide by <Threads>: %d\n", node, nt);
        printf("Enter <Threads>: ");
        scanf("%d", &nt);
    }

    out = fopen("out.txt", "w");
    fp = fopen("graph.dat", "w");
}

```

```

// Выделение памяти
z = (double **)malloc(2 * sizeof(double *));

for (int i = 0; i < 2; i++)
{
    z[i] = (double *)malloc(node * sizeof(double));
}
CalculateFirstTime(nc);
pthread_attr_t pattr;

// Получаем дефолтные значения атрибутов
pthread_attr_init(&pattr);
pthread_attr_setscope(&pattr, PTHREAD_SCOPE_SYSTEM);
pthread_attr_setdetachstate(&pattr, PTHREAD_CREATE_JOINABLE);
ThreadInfo *threads = (ThreadInfo *)calloc(nt, sizeof(ThreadInfo));

// Отвечают за синхронизацию расчета струны и времени
pthread_barrier_init(&barr1, NULL, nt + 1);
pthread_barrier_init(&barr2, NULL, nt + 1);
int j = node / nt;

for (int i = 0; i < nt; i++)
{
    threads[i].begin = i * j;
    threads[i].end = (i + 1) * j - 1;
    if (i == 0)
    {
        threads[i].begin = 1;
    }
    if (i == nt - 1)
    {
        threads[i].end = node - 2;
    }
    printf("i = %d, k = [%d; %d]\n", i, threads[i].begin, threads[i].end);
    if (pthread_create(&(threads[i].tid), &pattr, mysolver, (void *)&(threads[i])))

```

```

        {
            perror("pthread_create");
        }
    }

    struct timeval ts;

    // Засекаем время
    gettimeofday(&ts, NULL);
    for (CurrentTime = 0; CurrentTime < nc; CurrentTime++)
    {
        // Расчет по времени
        pthread_barrier_wait(&barr1);
        WriteFile(CurrentTime % 2, out);

        // Расчет струны
        pthread_barrier_wait(&barr2);
    }
    done = 1;
    printf("Time of work: %d milliseconds\n", TimeOfWork(ts));
    WriteGnuplotFile(fp, node, nc, z_max);

    // Освобождаем выделенную память
    free(threads);
    for (int i = 0; i < 2; i++)
    {
        free(z[i]);
    }
    free(z);
    return 0;
}

int TimeOfWork(struct timeval time_start)
{
    struct timeval time_curr, dtv;

```

```

// Вычисляем текущее время
gettimeofday(&time_curr, NULL);

dtv.tv_sec = time_curr.tv_sec - time_start.tv_sec;
dtv.tv_usec = time_curr.tv_usec - time_start.tv_usec;

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

return dtv.tv_sec * 1000 + dtv.tv_usec / 1000;
}

// Функция записи строк файл
void WriteFile(int t, FILE *f)
{
    for (int i = 0; i < node; i++)
    {
        fprintf(f, "%d\t%f\n", i, z[t][i]);
    }
    fprintf(f, "\n\n");
}

// Функция создания GIF файла с анимацией колебаний струны
void WriteGnuplotFile(FILE *out, int nodes, int time_interval, int max_elem)
{
    // GIF
    fprintf(out, "set terminal gif animate loop 0\n");
    fprintf(out, "set grid\n");
    fprintf(out, "set cbrange [0.9:1]\n");
    fprintf(out, "set xrange [0:%d]\n", nodes - 1);
    fprintf(out, "set yrange [-%d:%d]\n", (int)max_elem, (int)max_elem);
    fprintf(out, "set output 'lab2.gif'\n");
}

```

```

    fprintf(out, "do for [i=0:%d]{\n", time_interval - 1);
    fprintf(out, "plot 'out.txt' index i using 1:2 smooth bezier}\n");
}

// Функция решения уравнения и записи результатов в двумерный массив Z
void *mysolver(void *arg_p)
{
    ThreadInfo *thr = (ThreadInfo *)arg_p;

    int cur, prev;

    double f = 0.0, f2 = 0.0;

    double A2 = A * A;

    double DELTA_T2 = DELTA_T * DELTA_T;
    double DELTA_X2 = DELTA_X * DELTA_X;

    while (!done)
    {
        pthread_barrier_wait(&barr1);

        cur = CurrentTime % 2;
        prev = (CurrentTime + 1) % 2;

        for (int i = thr->begin; i <= thr->end; i++)
        {
            if (i == node / 2 && CurrentTime == 0)
            {
                f = F(i, CurrentTime);
            }
            else
            {
                f = 0.0;
            }

            // Прикладываем силу к третьей четверти струны в начальный момент времени
            if (i == node / 2 + node / 4 && CurrentTime < nc / 8)
            {
                f2 = -F(i, CurrentTime);
            }
        }
    }
}

```

```

        else
        {
            f2 = 0.0;
        }

        // Вычисление значений уравнения
        z[cur][i] = DELTA_T2 * (A2 * (z[prev][i - 1] - 2 * z[prev][i] + z[prev][i + 1]) /
DELTA_X2 + f + 0 + 2 * z[prev][i] - z[cur][i]);

        if (abs(z[cur][i]) > z_max)
            z_max = abs(z[cur][i]);
    }

    pthread_barrier_wait(&barr2);
}

pthread_exit(0);
}

void CalculateFirstTime(int nc)
{
    for (int i = 0; i <= node; i++)
    {
        z[0][i] = 0.0;
        z[1][i] = 0.0;
    }

    WriteFile(1, out);
}

```