МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №1

по дисциплине: Параллельное программирование
тема: «Сравнение парадигм конкурентности и параллелизма при разработке многопоточных программ в ОС Linux»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
доц. Островский Алексей Мичеславович

Белгород 2025 г.

Цель работы: исследовать чувствительность вычислительной схемы из индивидуального задания к ситуациям конкурентности, когда несколько потоков разделяют одно процессорное ядро; ситуациям параллелизма, когда каждый поток выполняется на отдельном ядре процессора (нет конкуренции за вычислительные ресурсы).

Условие индивидуального задания:

$$S = \sum_{i=1}^{N} \frac{cos(i^3)+i^4e^{-i}+ln(i+1)}{\sqrt{i^2+tan(i)+1}+i!}$$

Ход выполнения работы

Декомпозируем вычислительную задачу. Будем в каждом потоке вычислять не все элементы от 1 до N элемента, а N / M элементов, где M - количество потоков.

Исходный код:

```c
#define _GNU_SOURCE
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <unistd.h>
#include <stdint.h>
#include <math.h>

#ifndef NUM_THREADS
#define NUM_THREADS 4
#endif

#define NUM_ITERATIONS 10000000

volatile long double sums[NUM_THREADS];

long double factorial(const long double n)
{
    long double f = 1;
    for (long double i = 1; i <= n; ++i)
        f *= i;
    return f;
}

void *compute(void *arg)
{
    size_t iteration_num = (size_t)arg;
    size_t lower_bound = (NUM_ITERATIONS / NUM_THREADS) * iteration_num + 1;
    size_t upper_bound = (NUM_ITERATIONS / NUM_THREADS) * (iteration_num + 1) + 1;

    long double iter_factorial = factorial(lower_bound);

    for (uint64_t iter = lower_bound; iter < upper_bound; iter++)
    {
        sums[iteration_num] += (cos(pow(iter, 3)) + pow(iter, 4) * exp(-iter) + log(iter + 1)) /
               (sqrt(iter * iter + tan(iter) + 1) + iter_factorial);
        iter_factorial *= iter;
    }
```

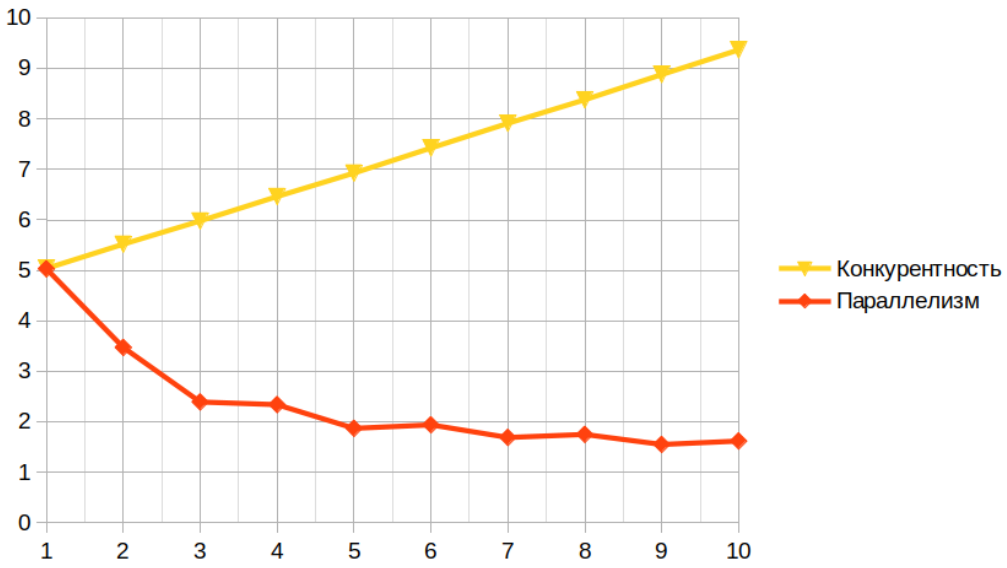
```
}
// Принудительное закрепление потока за конкретным ядром
void pin_thread_to_core(int core_id)
{
    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);
    CPU_SET(core_id, &cpuset);
    pthread_t current_thread = pthread_self();
    pthread_setaffinity_np(current_thread, sizeof(cpu_set_t), &cpuset);
}

void *compute_pinned(void *arg)
{
    int core_id = (int)(long)arg;
    pin_thread_to_core(core_id);
    return compute(arg);
}
int main(int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    if (argc < 2)
    {
        fprintf(stderr, "Запуск: %s <mode>\n"
                        "Опции:\n 1 - конкурентность, одно ядро\n"
                        "2 - параллелизм, разные ядра\n",
                argv[0]);
        return EXIT_FAILURE;
    }
    int mode = atoi(argv[1]);
    printf("Старт: %s, %d поток(а)(ов)...\n",
           mode == 1 ? "конкурентность, одно ядро" : "параллелизм, разные ядра", NUM_THREADS);
    for (size_t iter = 0; iter < NUM_THREADS; iter++)
    {
        sums[iter] = 0;
        if (mode == 1)
        {
            pthread_create(&threads[iter], NULL, compute, (void *)iter);
        }
        else
        {
            pthread_create(&threads[iter], NULL, compute_pinned, (void *)iter);
        }
    }
    for (size_t iter = 0; iter < NUM_THREADS; iter++)
        pthread_join(threads[iter], NULL);

    long double sum = 0;
    for (int i = 0; i < NUM_THREADS; i++) {
        sum += sums[i];
    }

    return EXIT_SUCCESS;
}
```

```
# Компилятор
CC=gcc

# Флаги компиляции
CFLAGS=-c -Wall

# Флаги линковки
LDFLAGS=-pthread

# Оверрайд переменных компиляции
DFLAGS="-DNUM_THREADS=2"

# Имя исполняемого файла
EXECUTABLE=lab1

# Исходники
SOURCES=lab1.c

# Объектные файлы
OBJECTS=$(SOURCES:.c=.o)



# Основная задача
all: $(SOURCES) $(EXECUTABLE)

# Задача по линковке. Линкует все .o файлы в один
$(EXECUTABLE): $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o $@ -lm

# Задача по компиляции, компилирует все .c файлы в .o
%.o : %.c
	$(CC) $(DFLAGS) -c $(CFLAGS) $< -o $@

clean:
	rm -rf *.o $(EXECUTABLE)
```

```
#!/bin/bash

echo "Компиляция..."
echo ""

MAX_THREADS=10

make clean
for i in $(seq 1 $MAX_THREADS); do
    make DFLAGS="-DNUM_THREADS=$i";
    mv ./lab1 ./lab1_$((i))_threads
    make clean
done

echo "Запуск конкуренции: "
echo ""
```

```
for i in $(seq 1 $MAX_THREADS); do
    time taskset -c 0 ./lab1_$((i))_threads 1
done

echo "Запуск параллелизма: "
echo ""

for i in $(seq 1 $MAX_THREADS); do
    time ./lab1_$((i))_threads 2
done

echo "Очистка: "
echo ""

for i in $(seq 1 $MAX_THREADS); do
    rm ./lab1_$((i))_threads
done
```

Результаты тестирования:



Вывод: в ходе лабораторной работы исследовали чувствительность вычислительной схемы из индивидуального задания к ситуациям конкурентности, когда несколько потоков разделяют одно процессорное ядро; ситуациям параллелизма, когда каждый поток выполняется на отдельном ядре процессора (нет конкуренции за вычислительные ресурсы). Ожидаемо при конкуренции за ресурсы росла и нагрузка на выбранное ядро, следовательно и время работы тоже увеличивалось. Опыт показал, что время работы увеличивалось линейно с увеличением потоков. В условиях отсутствия конкурентности время выполнения лишь падало, так как ядра процессора могли выполнять только свою задачу и не забивали задачами одно ядро. Причём после увеличения количества ядер до 6 дальнейшее увеличение количества задействованных ядер сильно на результат не влияло (что соответствует количеству производительных ядер процессора автора отчёта Intel Core i5 12600K). В условиях конкуренции за ресурсы увеличение потоков и декомпозиция задачи бессмысленна. А также в условиях отсутствия конкуренции бессмысленна и отсутствие декомпозиция задачи и её распараллеливание. Причём эффективное количе-

ство потоков соответствует количеству ядер процессора.