Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-211БВ-24

Студент: Антыгин В.Е.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 29.10.25

Москва, 2025

Постановка задачи

Вариант 13

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

Наложить K раз фильтр, использующий матрицу свертки, на матрицу, состоящую из вещественных чисел. Размер окна задается пользователем

Общий метод и алгоритм решения

Использованные системные вызовы:

pthread_create – создаёт новый поток, который выполняет функцию обработки части матрицы.

pthread_join – ожидает завершения выполнения потока.

gettimeofday – получает текущее время для измерения производительности.

rand / srand – генерация случайных чисел для заполнения матрицы.

Алгоритм

– Пользователь вводит:

- размеры матрицы (строки и столбцы);
- количество потоков.

– Матрица A заполняется случайными числами, а фильтр (ядро) задаётся вручную (например, фильтр повышения резкости).

– Программа создаёт заданное число потоков, делит между ними строки матрицы.

– Каждый поток выполняет свёртку своей части:

- для каждой ячейки суммируются произведения соседних элементов матрицы и коэффициентов ядра;
- результат записывается в матрицу B.

– После завершения всех потоков матрица B копируется обратно в A (для следующей итерации фильтрации).

– После заданного числа итераций программа:

- выводит результат;
- вычисляет время выполнения;

Код программы

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

#define KSIZE 3
#define K_ITER 10

float **A;
float **B;
float kernel[KSIZE][KSIZE] = {{0, -1, 0}, {-1, 5, -1}, {0, -1, 0}};

int rows, cols;
int threadCount;

void *apply_kernel_thread(void *arg) {
  int id = *(int *)arg;
  int chunk = rows / threadCount;
  int start = id * chunk;
  int end = (id == threadCount - 1) ? rows : start + chunk;
  int offset = KSIZE / 2;

  for (int i = start; i < end; ++i) {
    for (int j = 0; j < cols; ++j) {
      float sum = 0.0f;
      for (int u = 0; u < KSIZE; ++u) {
        for (int v = 0; v < KSIZE; ++v) {
          int x = i + u - offset;
          int y = j + v - offset;
```

```
          float val = 0.0f;
          if (x >= 0 && x < rows && y >= 0 && y < cols)
            val = A[x][y];
          sum += val * kernel[u][v];
        }
      }
      B[i][j] = sum;
    }
  }
  return NULL;
}

float **alloc_matrix(int r, int c) {
  float **m = malloc(r * sizeof(float *));
  for (int i = 0; i < r; ++i)
    m[i] = malloc(c * sizeof(float));
  return m;
}

void free_matrix(float **m, int r) {
  for (int i = 0; i < r; ++i)
    free(m[i]);
  free(m);
}

void print_matrix(float **m, int r, int c) {
  for (int i = 0; i < r; ++i) {
    for (int j = 0; j < c; ++j)
      printf("%6.2f ", m[i][j]);
    printf("\n");
  }
  printf("\n");
}

int main(int argc, char *argv[]) {
  if (argc > 4) {
    printf("Usage: %s <num_threads> <rows> <cols>\n", argv[0]);
    return 1;
  } else if (argc < 3) {
    printf("Usage: %s <num_threads> <rows> (for square matrix)\n", argv[0]);
    return 1;
  }
  rows = atoi(argv[2]);
  cols = argc == 4 ? atoi(argv[3]) : rows;
```

```
if (rows <= 0 || cols <= 0) {
  printf("Invalid matrix size.\n");
  return 1;
}
threadCount = atoi(argv[1]);
if (threadCount < 1 || threadCount > rows) {
  printf("Invalid thread count. Must be between 1 and %d.\n", rows);
  return 1;
}

A = alloc_matrix(rows, cols);
B = alloc_matrix(rows, cols);

srand(time(NULL));
for (int i = 0; i < rows; ++i)
  for (int j = 0; j < cols; ++j)
    A[i][j] = (rand() % 100000) / 1000.0f;

printf("Initial matrix:\n");
print_matrix(A, rows, cols);

pthread_t threads[threadCount];
int ids[threadCount];

struct timeval start, end;
gettimeofday(&start, NULL);

for (int k = 0; k < K_ITER; ++k) {
  for (int t = 0; t < threadCount; ++t) {
    ids[t] = t;
    pthread_create(&threads[t], NULL, apply_kernel_thread, &ids[t]);
  }

  for (int t = 0; t < threadCount; ++t)
    pthread_join(threads[t], NULL);

  for (int i = 0; i < rows; ++i)
    for (int j = 0; j < cols; ++j)
      A[i][j] = B[i][j];
}

gettimeofday(&end, NULL);
double durationOfComputing = (end.tv_sec - start.tv_sec) * 1000.0 +
                             (end.tv_usec - start.tv_usec) / 1000.0;
```

```
  printf("Result after %d iterations:\n", K_ITER);
  print_matrix(A, rows, cols);

  printf("Computing matrix %dx%d time with %d threads is %.3lf ms\n", rows,
         cols, threadCount, durationOfComputing);

  free_matrix(A, rows);
  free_matrix(B, rows);

  return 0;
}
```

Протокол работы программы

OS/lab2 main ● ? ❯ ./prog.out 1 3600

./prog.out 2 3600

./prog.out 3 3600

./prog.out 7 3600

./prog.out 8 3600

./prog.out 12 3600

./prog.out 128 3600

./prog.out 1024 3600

./prog.out 3600 3600

Result after 10 iterations:

Computing matrix 3600x3600 time with 1 threads is 3736.969 ms

Result after 10 iterations:

Computing matrix 3600x3600 time with 2 threads is 2075.640 ms

Result after 10 iterations:

Computing matrix 3600x3600 time with 3 threads is 1461.713 ms

Result after 10 iterations:

Computing matrix 3600x3600 time with 7 threads is 2607.336 ms

Result after 10 iterations:

Computing matrix 3600x3600 time with 8 threads is 938.328 ms

Result after 10 iterations:

Computing matrix 3600x3600 time with 12 threads is 895.107 ms

Result after 10 iterations:

Computing matrix 3600x3600 time with 128 threads is 875.043 ms

Result after 10 iterations:

Computing matrix 3600x3600 time with 1024 threads is 1336.444 ms

Result after 10 iterations:

Computing matrix 3600x3600 time with 3600 threads is 1249.929 ms

Анализ результатов

Число потоков	Время исполнения, мс	Ускорение	Эффективность
1	3736.969	1	100.00%
2	2075.64	1.80	90.02%
3	1461.713	2.56	85.22%
7	2607.336	1.43	20.48%
8	938.328	3.98	49.78%
12	895.107	4.17	34.79%
128	875.043	4.27	3.34%
1024	1336.444	2.80	0.27%
3600	1249.929	2.99	0.08%

По таблице видно, что при увеличении числа потоков время выполнения сначала уменьшается, но после определённого момента снова начинает расти.

На 1 потоке программа работает последовательно, это базовый вариант. Когда потоков становится 2 или 3, время резко сокращается, ускорение почти линейное, эффективность держится около 90%. При 7 или 8 потоках ускорение всё ещё заметно, но эффективность уже снижается, потому что процессор полностью загружен. У меня 12 потоков и 8 ядер, поэтому когда потоков становится больше этого числа, система начинает перегружать планировщик операционной системы.

При 12 потоках достигается максимальное ускорение, примерно в 4 раза, но если потоков становится слишком много, например 128, 1024 или 3600, производительность падает. Это происходит потому, что системе приходится постоянно переключать контекст между большим количеством потоков, и на эти переключения уходит значительное время. Кроме того, потоки конкурируют за общие ресурсы, такие как кэш, память и шина данных. Когда их слишком много, они начинают мешать друг другу, особенно при обращении к одной и той же матрице, и процессор тратит всё больше времени на синхронизацию вместо вычислений.

Вывод

Таким образом, увеличение числа потоков действительно ускоряет работу программы, но только до определённого предела. После 12 потоков производительность начинает снижаться из-за перегрузки процессора, частых переключений контекста и конкуренции потоков за общие ресурсы.