Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики Кафедра вычислительной математики и программирования

> Лабораторная работа №3 по курсу «Операционные системы»

Студент: Боев Савелий Сергеевич
Группа: М8О-207Б-21
Вариант: 11
Преподаватель: Миронов Евгений Сергеевич
Оценка:
Дата:
Подпись:

Содержание

Репозиторий	3
Постановка задачи	3
Цель работы	3
Задание	
Исходный код	4
Демонстрация работы программы	9
Замеры времени	9
Выводы	12

Репозиторий

https://github.com/IamNoobLEL/Labs-OSi

Постановка задачи

Цель работы

Изучение операционных систем

Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков 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 входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

Наложить K раз фильтры эрозии и наращивания на матрицу, состоящую из вещественных чисел. На выходе получается 2 результирующие матрицы.

Исходный код

```
1. #include <stdio.h>
2. #include <ctype.h>
3. #include <stdlib.h>
4. #include <stdbool.h>
5. #include <unistd.h>
6. #include <pthread.h>
7. #include <time.h>
8. #include <dirent.h>
9. #include <sys/time.h>
10.
11.
12. void print usage(char* cmd) {
13. printf("Usage: %s [-threads num]\n", cmd);
14.}
15.
16.bool read_matrix(float* matrix, size_t rows, size_t cols) {
17. for (size t i = 0; i < rows; i++) {
         for (size_t j = 0; j < cols; j++) {</pre>
18.
19.
              if (scanf("%f", &matrix[i * cols + j]) != 1) {
                   perror("Error while reading matrix");
20.
21.
                  return false;
22.
               //scanf("%f", &matrix[i*cols + j]);
              //printf("=== matrix[%ld] = %f\n", i*cols + j, matrix[i*cols + j]);
25.
         }
26.
27.
     return true;;
28.}
29.
30. bool print matrix(float* matrix, size t rows, size t cols) {
31. for (size t i = 0; i < rows; i++) {
32.
         for (size_t j = 0; j < cols; j++) {</pre>
33.
              printf("%.20g ", matrix[i * cols + j]);
34.
          }
35.
         printf("\n");
36.
     }
37.
     return false;
38.}
39.
40. void copy_matrix(float* from, float* to, size_t rows, size_t cols) {
41. for (size_t i = 0; i < rows; i++) {
       for (size_t j = 0; j < cols; j++) {</pre>
             to[i * cols + j] = from[i * cols + j];
43.
44.
45.
     }
46.}
47.
48. typedef struct {
```

```
49.
    int thread num;
50.
     int th count;
      int rows;
51.
52.
      int cols;
53.
     int w dim;
     float** matrix1;
     float** result1;
      float** matrix2;
     float** result2;
57.
58. } thread arg;
60.void* edit line(void* argument) {
      thread arg* args = (thread arg*)argument;
      const int thread num = args->thread num;
      const int th count = args->th count;
64.
65.
      const int rows = args->rows;
66.
      const int cols = args->cols;
67.
     int offset = args->w_dim / 2;
68.
     float** matrix1 ptr = args->matrix1;
      float** matrix2_ptr = args->matrix2;
71.
      float** result1 ptr = args->result1;
72.
      float** result2 ptr = args->result2;
73.
74.
     const float* matrix1 = *matrix1 ptr;
75.
      const float* matrix2 = *matrix2 ptr;
76.
     float* result1 = *result1 ptr;
      float* result2 = *result2 ptr;
78.
79.
       //printf("\n=== IN THREAD %d ===\n", thread num);
       // printf("offset = %d\n", offset);
80.
81.
82.
      for (int th_row = thread_num; th_row < rows; th_row += th_count) {</pre>
83.
           //printf("THREAD %d ROW %d\n", thread num, th row);
           for (int th col = 0; th col < cols; th col++) {</pre>
              // printf(" th col = %d\n", th col);
86.
              float max = matrix1[th_row * cols + th_col];
87.
              float min = matrix2[th_row * cols + th_col];
              for (int i = th row - offset; i 
88.
89.
                   for (int j = th col - offset; j 
90.
                      float curr1, curr2;
                       if ((i < 0) \mid | (i >= rows) \mid | (j < 0) \mid | (j >= cols)) {
91.
92.
                          curr1 = 0;
93.
                          curr2 = 0;
                       } else {
94.
95.
                          curr1 = matrix1[i * cols + j];
96.
                          curr2 = matrix2[i * cols + j];
97.
                      }
98.
                      // printf("[%d][%d] ", i, j);
99.
                      if (curr1 > max) {
```

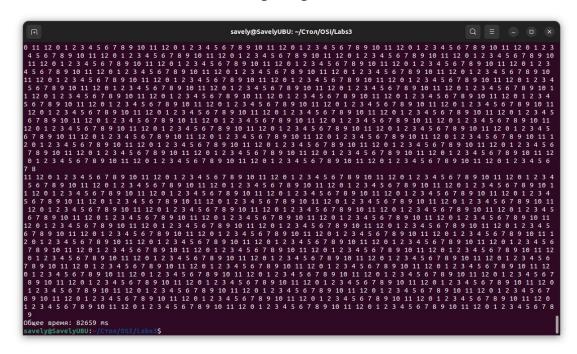
```
100.
                                   max = curr1;
101.
102.
                               if (curr2 < min) {</pre>
                                   min = curr2;
103.
104.
105.
                           // printf("\n");
106.
107.
108.
109.
                       result1[th row * cols + th col] = max;
                       result2[th row * cols + th col] = min;
110.
111.
112.
                  //printf("\n");
113.
114.
               pthread exit(NULL); // Заканчиваем поток
115.
         }
116.
117
          void put_filters(float** matrix_ptr, size_t rows, size_t cols, size_t w_dim,
   float** res1_ptr, float** res2_ptr, int filter_cnt, int th_count) {
118.
               float* tmp1 = (float*)malloc(rows * cols * sizeof(float));
               if (!tmp1) {
120.
                   perror("Error while allocating matrix\n");
121.
                   exit(1);
122.
123.
              float** matrix1_ptr = &tmp1;
              float* tmp2 = (float*) malloc(rows * cols * sizeof(float));
124.
125.
               if (!tmp2) {
126.
                  perror("Error while allocating matrix\n");
127.
                   exit(1);
128.
129.
              float** matrix2 ptr = &tmp2;
               copy matrix(*matrix ptr, tmp1, rows, cols);
130.
131
              copy matrix(*matrix ptr, tmp2, rows, cols);
132.
133.
              pthread t ids[th count];
134.
               thread arg args[th count];
135.
               for (int k = 0; k < filter cnt; k++) {
136.
                   for (int i = 0; i < th_count; i++) {</pre>
137.
138.
                       args[i].thread num = i;
139.
                       args[i].th count = th count;
140.
                       args[i].rows = rows;
                       args[i].cols = cols;
141.
142.
                       args[i].w_dim = w_dim;
143.
                       args[i].matrix1 = matrix1 ptr;
144.
                       args[i].result1 = res1 ptr;
                       args[i].matrix2 = matrix2_ptr;
145.
146.
                       args[i].result2 = res2 ptr;
147.
                       if (pthread create(&ids[i], NULL, edit line, &args[i]) != 0) {
148.
149.
                           perror("Can't create a thread.\n");
```

```
150.
                       }
151.
152.
153.
                   for(int i = 0; i 
                       if (pthread join(ids[i], NULL) != 0) {
154.
155.
                           perror("Can't wait for thread\n");
156.
157.
158.
159.
                  if (filter cnt > 1) {
                      float** swap = res1 ptr;
160.
161.
                      res1 ptr = matrix1 ptr;
162.
                      matrix1 ptr = swap;
163.
164.
                      swap = res2_ptr;
165.
                      res2_ptr = matrix2_ptr;
166.
                      matrix2 ptr = swap;
167.
168.
               }
169.
170.
              free(tmp1);
171.
               free(tmp2);
172.
          }
173.
174.
          int main(int argc, char* argv[]) {
                  printf("Enter K = ");
175.
176.
                  int threads;
177.
                  scanf("%d", &threads);
178.
              if (argc == 3) {
179.
180.
                  threads = \underline{atoi}(argv[2]);
               } else if (argc != 1) {
181.
182.
                  print_usage(argv[0]);
                  return 0;
183.
184.
              }
185.
              int rows;
186.
187.
              int cols;
188.
              printf("Enter matrix dimensions:\n");
              scanf("%d", &cols);
189.
190.
              scanf("%d", &rows);
191.
              float* matrix = (float*)malloc(rows * cols * sizeof(float));
              float* res1 = (float*)malloc(rows * cols * sizeof(float));
192.
193.
              float* res2 = (float*)malloc(rows * cols * sizeof(float));
194.
              if (!matrix || !res1 || !res2) {
                  perror("Error while allocating matrix\n");
195.
196.
                  return 1;
197.
198.
               read matrix(matrix, rows, cols);
199.
200.
              int w dim;
```

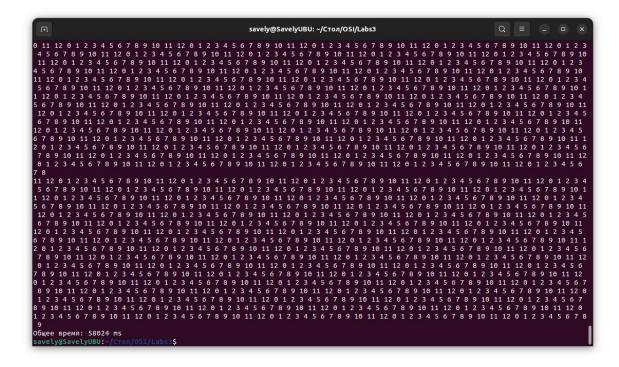
```
201.
              printf("Enter window dimension:\n");
202.
              scanf("%d", &w dim);
203.
             if (w_dim % 2 == 0) {
204.
                  perror("Window dimension must be an odd number\n");
                 return 1;
205.
206.
207.
208.
             printf("Result \n");
209.
             int k;
210.
             scanf("%d", &k);
211.
212.
             struct timeval start, end;
213.
             gettimeofday(&start, NULL);
214.
215.
             put_filters(&matrix, rows, cols, w_dim, &res1, &res2, k, threads);
216.
217.
             gettimeofday(&end, NULL);
218.
219.
             long sec = end.tv_sec - start.tv_sec;
220.
              long microsec = end.tv_usec - start.tv_usec;
221.
             if (microsec < 0) {</pre>
222.
                 --sec;
223.
                microsec += 1000000;
224.
              }
225.
             long elapsed = sec*1000000 + microsec;
226.
227.
228.
             printf("Dilation:\n");
229.
             print matrix(res1, rows, cols);
230.
             printf("Erosion:\n");
231.
             print_matrix(res2, rows, cols);
232.
             printf("Total time: %ld ms\n", elapsed);
233.
234.
             free (res1);
235.
             free(res2);
236.
             free (matrix);
237.
             return 0;
       }
238.
```

Демонстрация работы программы

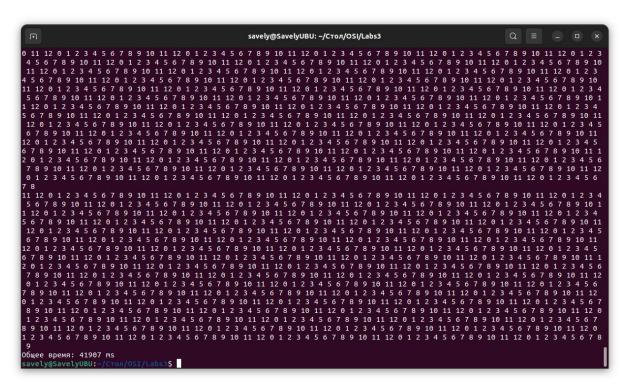
Замеры времени



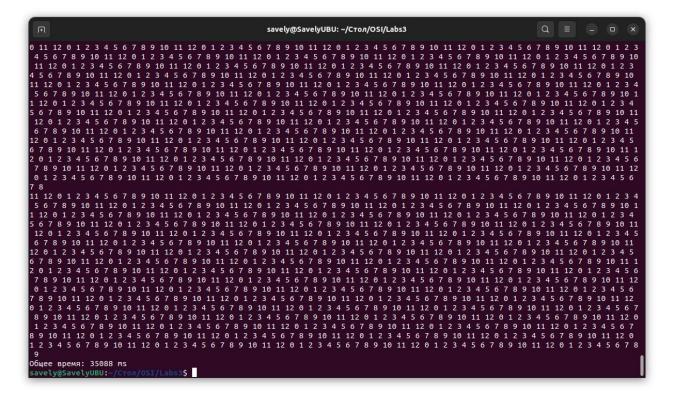
1 поток



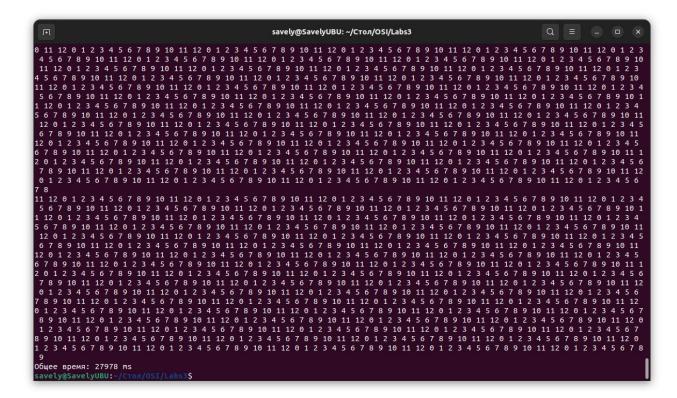
2 потока



3 потока



4 поток



5 потоков

Выводы

Мною было освоено написании программы, создающей и производящей вычисления в нескольких потоках, а также синхронизация данных между этими потоками. Это позволяет серьезно сократить время выполнения некоторых задач, ибо действия выполняются не последовательно, а параллельно.