ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Лабораторная работа №3

Выполнил:
Пятаев Егор, гр. 15206

Постановка задачи

Реализовать примитив синхронизации "барьер" в системе с общей памятью.

Вариант задания: локальная синхронизация между "соседними" потоками.

Листинг программы C

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
// L1 / 2 = 256 * 128 = 32Kb
// L2 / 2 = 512 * 256 = 128Kb
// L3 / 2= 2048 * 1538 = 3072Kb
// L3 * 10 = 8192 * 7690 = 61520Kb
static inline uint64_t read_time(void)
{
    uint32_t a, d;
    __asm__ volatile("rdtscp\n\t":"=a"(a),"=d"(d));
    return ((uint64_t)d<<32)+a;
}
double cpu_Hz = 3100000000ULL;
pthread_mutex_t m=PTHREAD_MUTEX_INITIALIZER;
pthread_barrier_t barrier;
double gmin = 100000000.0;
double gavg = 0.0;
void set_min_avg(double* mini, double* avgi){
  pthread_mutex_lock(&m);
  if(*mini < gmin) gmin = *mini;
  gavg += *avgi;
  pthread_mutex_unlock(&m);
}
const int ITERS = 30;
const int N = 256;
const int M = 128;
int* matrix = NULL;
int* new = NULL;
int thread_num = 128;
pthread_t threads[128];
volatile int ids[128][4096] = {0};
void my_barrier(int id, int iter){
  int left  = (id == 0 ? (thread_num - 1) : (id - 1));
  int right = (id == (thread_num - 1) ? 0 : (id + 1));
  ++ids[id][0];
  while(ids[left][0] < iter || ids[right][0] < iter){}
}
void init_matrix(int* matrix){
  for(int i = 0; i < N + 2; i++){
    for(int j = 0; j < M + 2; j++){
      if(i != 0 && i != N + 1 && j != 0 && j != M + 1){
        matrix[i * (M + 2) + j] = rand() % 255;
      }
    }
  }
}
void print_matrix(int* matrix){
  printf("\n\n");
  for(int i = 0; i < N + 2; i++){
    printf("\t");
    for(int j = 0; j < M + 2; j++){
      printf("%4d ", matrix[i * (M + 2) + j]);
    }
    printf("\n\n");
  }
  printf("\n");
}
int cmpfunc(const void * a, const void * b) {
   return(*(int*)a - *(int*)b );
}
int get_median(int* filter, int length){
  qsort(filter, length, sizeof(int), cmpfunc);
  return filter[length / 2];
}
int* update_filter(int* matrix, int* filter, int i, int j){
  int count = 0;
  int ti = i;
  int tj = j;
  --i;
  for(int k = 0; k < 3; k++){
```

```
    j = tj;
    --j;
    for(int l = 0; l < 3; l++){
      if(matrix[i * (M + 2) + j] == 0){
        filter[count++] = matrix[ti * (M + 2) + tj];
      }
      else{
        filter[count++] = matrix[i * (M + 2) + j];
      }
      ++j;
    }
    ++i;
  }
  return filter;
}
void use_filter(int* matrix, int* new, int* filter, int id){
  for(int i = ((N + 1) / thread_num) * id + 1; i < (id + 1) * ((N + 1) / thread_num) + 1; i++){
    for(int j = 1; j < M + 1; j++){
      new[i * (M + 2) + j] = get_median(update_filter(matrix, filter, i, j), 9);
    }
  }
}
void* thread_func(void* arg){
  int id = *(int*)arg;
  int* filter = (int*)calloc(9, sizeof(int));
  int iter = 0;
  double min = 10000000.0;
  double avg = 0.0;
  for(int i = 0; i < ITERS; i++){
    my_barrier(*(int*)arg, ++iter);
    use_filter(matrix, new, filter, *(int*)arg);
    uint64_t start1, stop1;
    start1 = read_time();
    my_barrier(*(int*)arg, ++iter);
    // pthread_barrier_wait(&barrier);
    stop1 = read_time();
    double t = (double)(stop1 - start1) / cpu_Hz;
    avg += t;
    if(t < min && t != 0.0) min = t;
    memcpy(matrix + (((N + 1) / thread_num) * id + 1) * (M + 2), new + (((N + 1) / thread_num) * id + 1) * (M + 2), sizeof(int) * ((id + 1) * ((N + 1) /
thread_num) + 1 - (((N + 1) / thread_num) * id + 1)) * (M + 2));
  }
  set_min_avg(&min, &avg);
  free(filter);
}
int main(int argc, char* argv[]){
  matrix = (int*)calloc((1 + N + 1) * (1 + M + 1), sizeof(int));
  new = (int*)calloc((1 + N + 1) * (1 + M + 1), sizeof(int));
  int* args = (int*)malloc(128 * sizeof(int));
  srand(time(NULL));
  init_matrix(matrix);
  pthread_barrier_init(&barrier, NULL, thread_num);
  for(int i = 0; i < thread_num; i++){
    args[i] = i;
    if(0 != pthread_create(&threads[i], NULL, thread_func, (void*)(args + i))) printf("create error\n");
  }

  for(int i = 0; i < thread_num; i++){
    if(0 != pthread_join(threads[i], NULL)) printf("join error\n");
  }
  printf("gmin: %0.15f\ngagv: %0.15f\n", gmin, gavg / (3 * 129));
  free(matrix);
  free(new);
  return 0;
}
```

Результаты

Плотная упаковка:

Размер	Время, min	Время, avg
256 * 128	0.000000017096774	0.073191732048012
512 * 256	0.000000018485901	0.083274194930399
2048 * 2048	0.000000020322581	0.100105586252396
16384 * 5120	0.000000021290323	0.158877453415854

Кэш-строка:

Размер	Время, min	Время, avg
256 * 128	0.000000017618019	0.083454075233808
512 * 256	0.000000018064516	0.085554586934234
2048 * 2048	0.000000031290323	0.090788328976411
16384 * 5120	0.000000036451613	0.203030644439443

Страница:

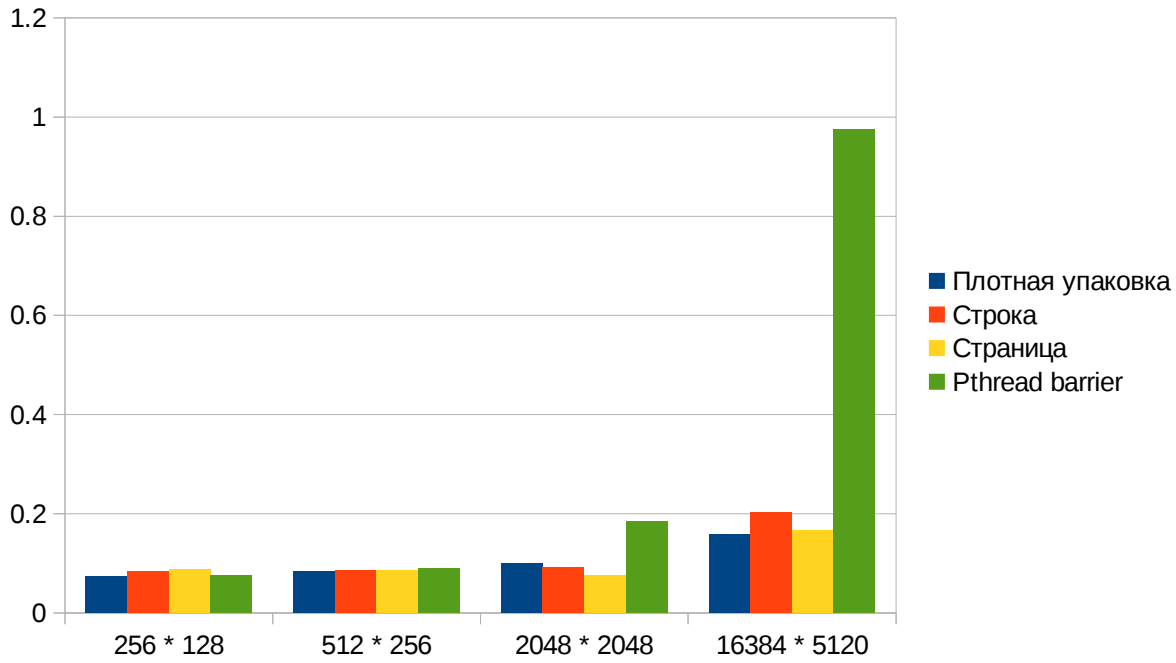
Размер	Время, min	Время, avg
256 * 128	0.000000021290323	0.087801240872718
512 * 256	0.000000022258065	0.086658745499708
2048 * 2048	0.000000047741935	0.074807112038009
16384 * 5120	0.000000070967742	0.165915268205385

Pthread:

Размер	Время, min	Время, avg
256 * 128	0.0000003987	0.075953
512 * 256	0.0002177338	0.088692
2048 * 1538	0.0012795687	0.184066
8192 * 7690	0.0015687800	0.974163

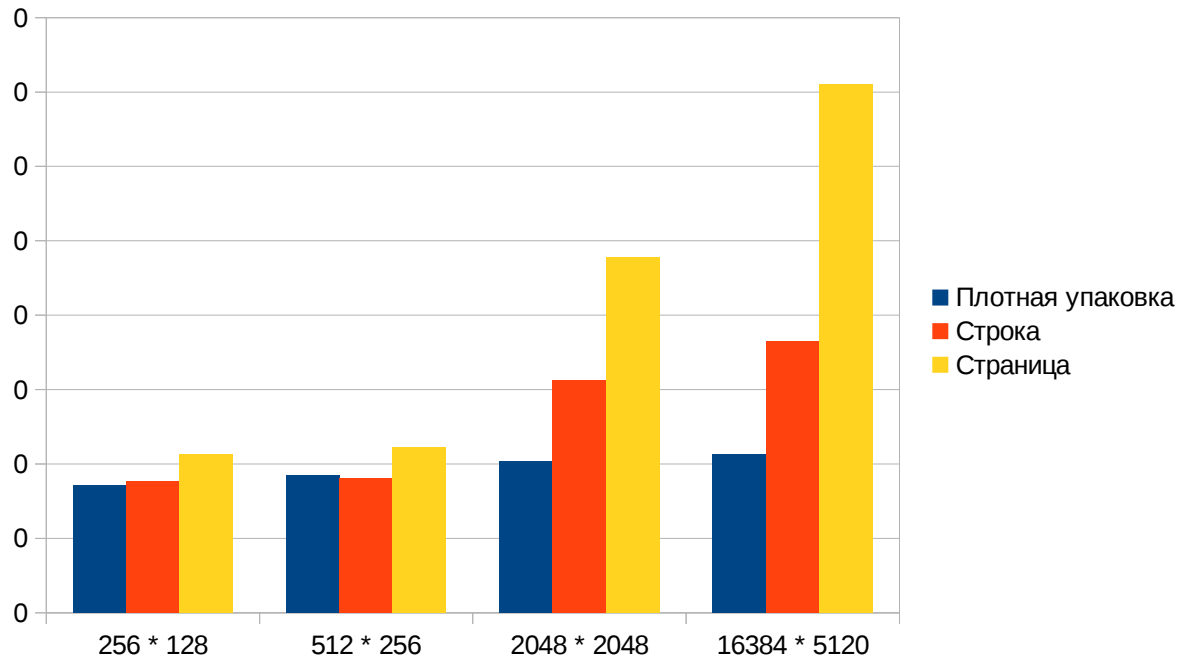
Среднее время:

Размер	Плотная упаковка	Строка	Страница	Pthread barrier
256 * 128	0.073191732048012	0.083454075233808	0.087801240872718	0.075953
512 * 256	0.083274194930399	0.085554586934234	0.086658745499708	0.088692
2048 * 2048	0.100105586252396	0.090788328976411	0.074807112038009	0.184066
16384 * 5120	0.158877453415854	0.203030644439443	0.165915268205385	0.974163



Минимальное время:

Размер	Плотная упаковка	Строка	Страница	Pthread barrier
256 * 128	0.000000017096774	0.000000017618019	0.000000021290323	0.0000003987
512 * 256	0.000000018485901	0.000000018064516	0.000000022258065	0.0002177338
2048 * 2048	0.000000020322581	0.000000031290323	0.000000047741935	0.0012795687
16384 * 5120	0.000000021290323	0.000000036451613	0.000000070967742	0.0015687800



Выводы

Для решения поставленной задачи была реализована программа на языке C реализующая медианный фильтр обработки изображения, барьер синхронизации между “соседними” потоками.

Из результатов замеров видно, что при увеличении расстояния между переменными синхронизации увеличивается время обработки матрицы.