Министерство образования Российской Федерации
Московский Государственный Технический
Университет им. Н.Э. Баумана

Отчет по лабораторной работе №6
По курсу «Анализ алгоритмов»

Тема: «Конвейер»

Студент: Мхитарян В.К.
Группа: **ИУ7-54**

Преподаватель: **Погорелов Д.А.**

Москва, 2018

Содержание

Введение

Целью данной лабораторной работы является применение конвейерной обработки данных. Конвейерная обработка данных может быть полежной, когда каждая операция занимает много времени. В данной работе будет рассмотрена задача заполнения массива случайными числами, вычисление суммы элементов массива и запись результата в файл.

1. Аналитическая часть

В данном разделе будут приведены теоретические сведения о алгоритмах, их описание и разбор.

1.1. Описание алгоритмов

Для реализации конвейерной обработки с помощью потоков было выделено три этапа, на каждом из которых производится своя операция обработки данных. В первом потоке выполняется первая операция (заполнение массива случайными числами), после чего выходные данные передаются во второй поток для выполнения второй операции (вычисление суммы элементов массива). Выходные данные после выполнения третьей операции подаются в третий поток для выполнения третьей операции (запись результата в файл).

2. Конструкторская часть

В данном разделе будет рассмотрен способ организации конвейера.

2.1. Способ организации конвейера

Конвейер состоит из 3х потоков для 3х последовательных задач и 4х очередей. Сначала, id массивов, которые необходимо обработать, заносятся в очередь Q_0. Из очереди Q_0, если она не пуста и первый поток свободен, достается элемент, который и передается в первый поток.

В первом потоке с указанным id создается матрица, заполненная случайным образом, и находится ее обратная матрица. После выполнения действий над текущим элементом первый поток добавляет, осуществляя монополизированный доступ к переменной очереди Q_1, помещает результат выполнения операции в очередь Q_1.

Далее, по аналогии с первым потоком, второй поток извлекает из очереди Q_1 элемент, производит над ним вычисления (возведение матрицы в квадрат) и заносит результат в очередь Q_2. Аналогично с третьим потоком. Процесс заканчивается, когда в очереди Q_3 (очереди полностью готовых элементов) аккумулируется столько же элементов, сколько поступило на вход.

3. Технологическая часть

В данном разделе будет описаны требования к программному обеспечению, средства реализации и листинг кода.

3.1. Требования к программному обеспечению

Минимальные системные требования: PC с операционной системой Windows 7, MacOS X или Linux(Ubuntu). Процессор с частотой 2.0GHz или выше. С оперативной памятью не менее 2 Гб. Требуются устройства ввода: клавиатура, мышь.

3.2. Средства реализации

В данной работе был выбран открытый мультипарадигмальный компилируемый язык программирования C++. C++ является одним из самых популярных языков программирования и имеет огромную область применения. Среда для разработки выбрана CLion.

3.3. Листинг кода

Листинг 1: Конвейер

```
while (q[2].size() != OBJ_COUNT) {
        if (current_element < OBJ_COUNT) {
            Array array(ARRAY_SIZE, obj[current_element]);
            queue1.push(array);
            current_element++;
        }

        // Level 1
        if (threads[0].joinable())
            threads[0].join();
        if (!queue1.empty() && !threads[0].joinable()) {
            Array front_array = queue1.front();
            queue1.pop();

            auto level1 = [](Array obj, queue<Array> &queue
                ) {
                obj.rand_vec();

                mutex queue_mutex;
```

```
19                  queue_mutex.lock();
20                  queue.push(obj);
21                  queue_mutex.unlock();
22              };
23
24              threads[0] = thread(level1, front_array, ref(q
                    [0]));
25          }
26
27          // Level 2
28          if (threads[1].joinable()){
29              threads[1].join();
30          }
31          if (!q[0].empty() && !threads[1].joinable()) {
32              Array front_array = q[0].front();
33              q[0].pop();
34
35              auto level2 = [](Array obj, queue <Array> &
                    queue) {
36                  obj.summa();
37
38                  mutex queue_mutex;
39                  queue_mutex.lock();
40                  queue.push(obj);
41                  queue_mutex.unlock();
42              };
43
44              threads[1] = thread(level2, front_array, ref(q
                    [1]));
45          }
46
47          // Level 3
48          if (threads[2].joinable()){
49              threads[2].join();
50          }
51          if (!q[1].empty() && !threads[2].joinable()) {
52              Array front_array = q[1].front();
53              q[1].pop();
54
55              auto level3 = [](Array obj, queue <Array> &
                    queue) {
56                  string str = "file" + to_string(obj.get_id
                        ()) + ".txt";
```

```
57                  obj.print(str);
58
59                  mutex queue_mutex;
60                  queue_mutex.lock();
61                  queue.push(obj);
62                  queue_mutex.unlock();
63              };
64
65              threads[2] = thread(level3, front_array, ref(q
                    [2]));
66          }
67      }
```

Листинг 2: Класс Array

```
1  class Array {
2  private:
3      std::vector<int> array;
4      int length;
5      int id;
6
7      void timeout() {
8          int wait_time = rand()%2000;
9          std::this_thread::sleep_for(std::chrono::
               nanoseconds(wait_time));
10     }
11
12 public:
13     Array(int length, int id) {
14         this->length = length;
15         this->id = id;
16         for (int i = 0; i < length; ++i) {
17             array.push_back(0);
18         }
19     };
20
21     void rand_vec() {
22         timeout();
23
24         for (int i = 0; i < length; ++i) {
25             array[i] = rand()%1000;
26         }
27     }
28
```

```
29      void summa() {
30          timeout();
31          int sum = 0;
32          for (int i = 0; i < length; i++){
33              sum += array[i];
34          }
35          array[0] = sum;
36      }
37
38      void print(std::string file_name) {
39          std::ofstream file (file_name);
40
41          timeout();
42
43          file << "Summa: ";
44          file << array[0];
45          file << "\n";
46      }
47
48      int get_id() {
49          return id;
50      }
51  };
```

4. Эксперементальная часть

В данном разделе будут приведены эксперименты и сравнительный анализ алгоритмов на основе приведенных данных.

4.1. Постановка эксперимента

Задачей этого эксперимента является сравнение эффективности последовательного выполнения команд и конвейерной обработки. Будет проведет анализ на основе тестов.

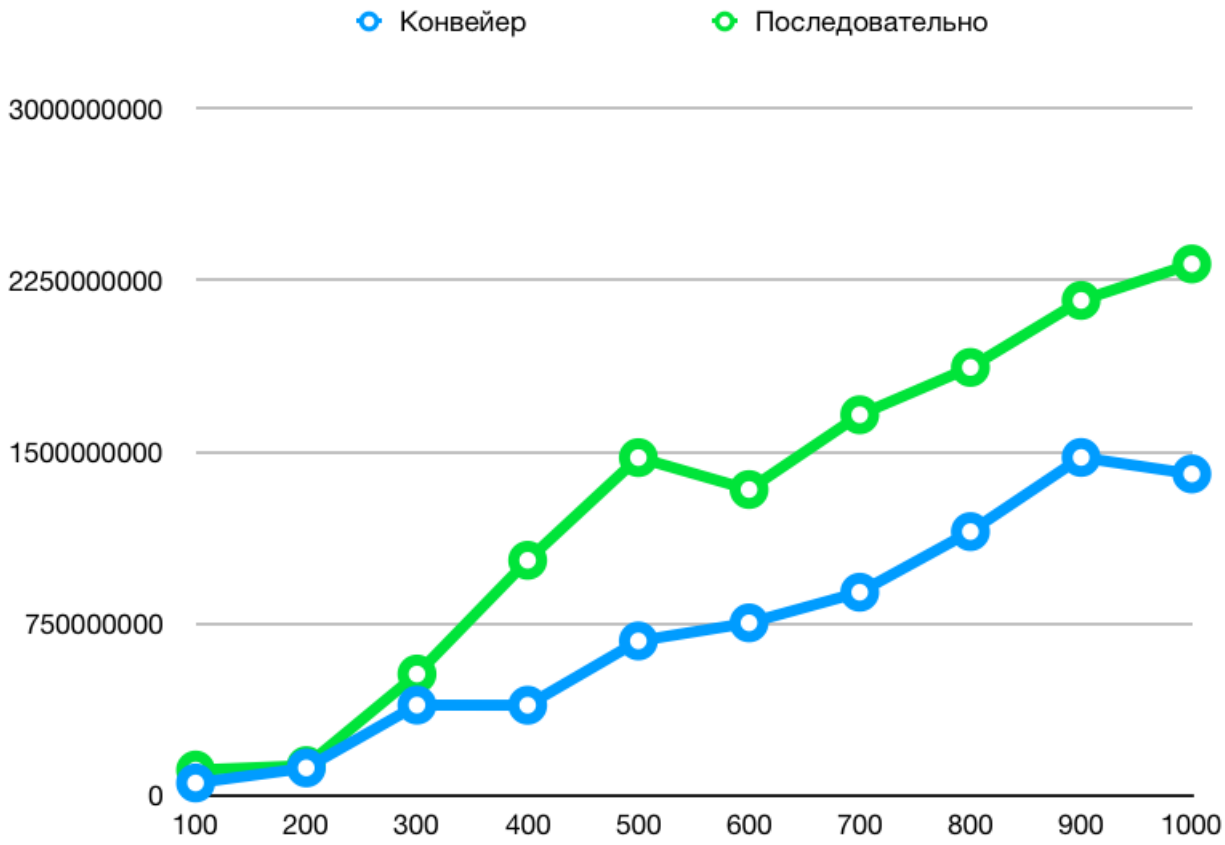
4.2. Сравнительный анализ на материале экспериментальных данных

Измерения проводились для целочисленных массивов постоянной величины равной 500 с помощью библиотеки std::chrono. Значения в таблицах представлены в наносекундах(среднее из 10 замеров).

Таблица 1. Результаты замеров времени.

	Конвейер	Последовательно
100	54686477	111191164
200	120387625	130504283
300	394637157	530076330
400	393970683	1026480147
500	674612983	1474704303
600	753455103	1335335403
700	887185877	1868453597
800	1151811773	1661585940
900	1475499240	2319997067
1000	1203419145	2160687625

Представление тестовых данных из таблицы 1.



По результатам проведенного эксперимента можно сделать вывод, что выполнение конвейерной обработкой эффективнее чем последовательной. В данном примере конвейерная реализация при разделении на 3 этапа дает прирост в производительности примерно в 2 раза.

Заключение

В ходе проведенной работы, были изучена и реализована конвейерная обработка данных на языке программирования C++. Были получены навыки работы с потоками и mutex. Было выполнено сравнение последовательных вычислении и конвейерной обработки.