

*Государственное образовательное учреждение высшего  
профессионального образования*

**«Московский государственный технический  
университет имени Н.Э. Баумана»  
(МГТУ им. Н.Э. Баумана)**

---

ЛАБОРАТОРНАЯ РАБОТА №5  
ПО КУРСУ «АНАЛИЗ АЛГОРИТМОВ»

**Конвейерная обработка**

Студент: Нгуен Фьюк Санг

Группа ИУ7-56Б

Преподаватели: Волокова Л. Л., Строганов Ю.В.

Оценка:

*Москва, 2020 г.*

# Оглавление

<b>Введение</b>	<b>2</b>
<b>1 Аналитический раздел</b>	<b>3</b>
1.1 Конвейерная обработка данных . . . . .	3
1.2 Вывод . . . . .	3
<b>2 Конструкторская часть</b>	<b>4</b>
2.1 Разработка конвейерной обработки данных . . . . .	4
<b>3 Технологический раздел</b>	<b>5</b>
3.1 Требования к программному обеспечению . . . . .	5
3.2 Средства реализации . . . . .	5
3.3 Листинг кода . . . . .	5
3.4 Вывод . . . . .	10
<b>4 Экспериментальный раздел</b>	<b>11</b>
4.1 Сравнительный анализ . . . . .	11
4.2 Вывод . . . . .	11
<b>Заключение</b>	<b>13</b>
<b>Литература</b>	<b>14</b>

# Введение

При обработке данных могут возникать ситуации, когда необходимо обработать множество данных последовательно несколькими алгоритмами. В этом случае удобно использовать конвейерную обработку данных.

Цель данной работы: получить навык организации асинхронного взаимодействия потоков на примере конвейерной обработки данных.

В рамках выполнения работы необходимо решить следующие задачи:

- рассмотреть и изучить конвейерную обработку данных;
- реализовать конвейер с количеством лент не меньше трех в многопоточной среде;
- на основании проделанной работы сделать выводы.

# 1. Аналитический раздел

В данном разделе будет описан принцип конвейерной обработки.

## 1.1 Конвейерная обработка данных

Если задача заключается в применении одной последовательности операций ко многим независимым элементам данных, то можно организовать распараллеленный конвейер. Здесь можно провести аналогию с физическим конвейером: данные поступают с одного конца, подвергаются ряду операций и выходят с другого конца. Для того, чтобы распределить работу по принципу конвейерной обработки данных, следует создать отдельный поток для каждого участка конвейера, то есть для каждой операции. По завершении операции элемент данных помещается в очередь, откуда его забирает следующий поток. В результате поток, выполняющий первую операцию, сможет приступить к обработке следующего элемента, пока второй поток трудится над первым элементом. Конвейеры хороши также тогда, когда каждая операция занимает много времени; распределяя между потоками задачи, а не данные, мы изменяем качественные показатели производительности [?].

## 1.2 Вывод

В данном разделе был описан принцип конвейерной обработки.

## 2. Конструкторская часть

### 2.1 Разработка конвейерной обработки данных

Принцип работы стадийной обработки представлен на рис. 2.1.

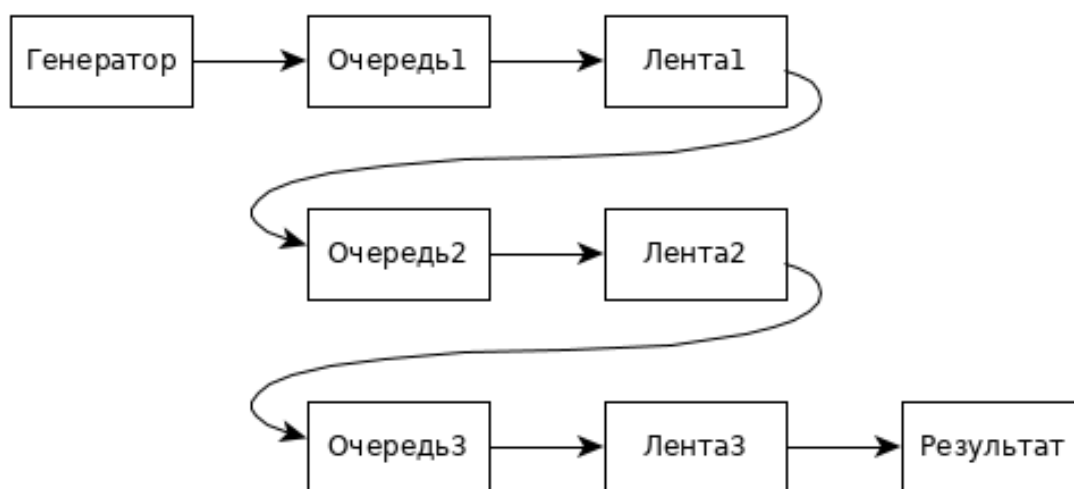


Рис. 2.1: Конвейерная обработка данных

## 3. Технологический раздел

В данном разделе будут предъявлены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также представлен листинг кода программы.

### 3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать линейную, конвейерную обработку данных. Пользователь должен иметь возможность вводить количество объектов, которые будут обрабатываться.

### 3.2 Средства реализации

Для реализации программы был использован язык C++ [1]. Для замера процессорного времени была использована функция `rdtsc()` из библиотеки `stdrin.h`.

### 3.3 Листинг кода

В листинге 3.1 описан класс конвейера. В листинге 3.2 представлена реализация линейной и конвейерной обработки матриц, а в листинге 3.3 - параллельной.

Листинг 3.1: Класс конвейера

```
1 class Conveyor
2 {
3 private:
4     size_t obj_count;
5     size_t queue_count;
6     size_t averege_time;
7     const size_t delay_time = 3;
8     std::vector<int> time_stay_at_queue[4];
9
10 public:
11     Conveyor(size_t _objs, size_t _queues, size_t msec) :
12         obj_count(_objs), queue_count(_queues), averege_time(msec) {}
13
14     void execute_linear();
15     void execute_parallel();
16
17 private:
18     size_t get_time();
19
20     void log_print_obj_queue(MatrixSet& obj, size_t qu);
```

```

21 void log_print_start(MatrixSet& obj, size_t qu, size_t time);
22 void log_print_end(MatrixSet& obj, size_t qu, size_t time);
23 void log_print_time(MatrixSet& obj, size_t qu, size_t time);
24
25 void do_linear_work1(MatrixSet& obj, size_t queue, bool log=true);
26 void do_linear_work2(MatrixSet& obj, size_t queue, bool log=true);
27 void do_linear_work3(MatrixSet& obj, size_t queue, bool log=true);
28
29 void* do_parallel_work1(void *_args);
30 void* do_parallel_work2(void *_args);
31 void* do_parallel_work3(void *_args);
32 };

```

Листинг 3.2: Реализация линейной обработки матрицы

```

1 void Conveyor::do_linear_work1(MatrixSet& obj, size_t queue, bool log)
2 {
3     size_t start = get_time();
4     if (log)
5         log_print_start(obj, queue, start);
6     obj.sum(0, obj.size / 3);
7     size_t end = get_time();
8     if (log)
9     {
10         log_print_end(obj, queue, end);
11         log_print_time(obj, queue, end - start);
12     }
13 }
14
15 void Conveyor::do_linear_work2(MatrixSet& obj, size_t queue, bool log)
16 {
17     size_t start = get_time();
18     if (log)
19         log_print_start(obj, queue, start);
20     obj.sum(obj.size / 3, 2 * obj.size / 3);
21     size_t end = get_time();
22     if (log)
23     {
24         log_print_end(obj, queue, end);
25         log_print_time(obj, queue, end - start);
26     }
27 }
28
29 void Conveyor::do_linear_work3(MatrixSet& obj, size_t queue, bool log)
30 {
31     size_t start = get_time();
32     if (log)
33         log_print_start(obj, queue, start);
34     obj.sum(2 * obj.size / 3, obj.size);
35     size_t end = get_time();
36     if (log)
37     {
38         log_print_end(obj, queue, end);
39         log_print_time(obj, queue, end - start);

```

```

40 }
41 }
42
43 void Conveyor::execute_linear()
44 {
45
46     std::queue<MatrixSet> obj_generator;
47
48     for (size_t i = 0; i < obj_count; i++)
49         obj_generator.push(MatrixSet(i + 1, 1038, -200, 200));
50
51     std::vector<MatrixSet> obj_pools;
52
53     while (obj_pools.size() != obj_count)
54     {
55         MatrixSet obj = obj_generator.front();
56         obj_generator.pop();
57
58         for (size_t i = 0; i < queue_count; i++)
59         {
60             if (i == 0)
61                 do_linear_work1(obj, i);
62             else if (i == 1)
63                 do_linear_work2(obj, i);
64             else if (i >= 2)
65                 do_linear_work3(obj, i);
66         }
67
68         obj_pools.push_back(obj);
69     }
70 }

```

Листинг 3.3: Реализация параллельной обработки матрицы

```

1 void* Conveyor::do_parallel_work1(void *_args)
2 {
3     par_args *args = (par_args*) _args;
4     size_t start = get_time();
5     args->obj.sum(0, args->obj.size / 3);
6
7     args->mutex.lock();
8     args->queue.push(args->obj);
9     args->mutex.unlock();
10
11     size_t end = get_time();
12     if (args->log)
13         log_print_time(args->obj, args->queue_num, end - start);
14     time_stay_at_queue[args->queue_num + 1].push_back(-end);
15     return NULL;
16 }
17
18 void* Conveyor::do_parallel_work2(void *_args)
19 {
20     par_args *args = (par_args*) _args;

```



```

21  size_t start = get_time();
22  args->obj.sum(args->obj.size / 3, 2 * args->obj.size / 3);
23
24  args->mutex.lock();
25  args->queue.push(args->obj);
26  args->mutex.unlock();
27
28  size_t end = get_time();
29  if (args->log)
30      log_print_time(args->obj, args->queue_num, end - start);
31  time_stay_at_queue[args->queue_num + 1].push_back(-end);
32  return NULL;
33 }
34
35 void* Conveyor::do_parallel_work3(void * _args)
36 {
37     par_args *args = (par_args*) _args;
38     size_t start = get_time();
39     args->obj.sum(2 * args->obj.size / 3, args->obj.size);
40
41     args->mutex.lock();
42     args->queue.push(args->obj);
43     args->mutex.unlock();
44
45     size_t end = get_time();
46     if (args->log)
47         log_print_time(args->obj, args->queue_num, end - start);
48     time_stay_at_queue[args->queue_num + 1].push_back(-end);
49     return NULL;
50 }
51
52 void Conveyor::execute_parallel()
53 {
54     std::queue<MatrixSet> obj_generator;
55
56     for (size_t i = 0; i < obj_count; i++)
57         obj_generator.push(MatrixSet(i + 1, 1038, -200, 200));
58
59     std::vector<MatrixSet> obj_pool;
60
61     std::vector<std::thread> threads(3);
62     std::vector<std::queue<MatrixSet>> queues(3);
63     std::vector<std::mutex> mutexes(4);
64     size_t prev_time = get_time() - delay_time;
65
66     while (obj_pool.size() != obj_count)
67     {
68         size_t cur_time = get_time();
69
70         if (!obj_generator.empty() && prev_time + delay_time < cur_time)
71         {
72             MatrixSet obj = obj_generator.front();
73             obj_generator.pop();

```

```

74     queues[0].push(obj);
75
76     prev_time = get_time();
77     time_stay_at_queue[0].push_back(-prev_time);
78 }
79
80 for (unsigned i = 0; i < queue_count; i++)
81 {
82     if (threads[i].joinable())
83         threads[i].join();
84
85     if (!queues[i].empty() && !threads[i].joinable())
86     {
87         mutexes[i].lock();
88         MatrixSet obj = queues[i].front();
89         queues[i].pop();
90         mutexes[i].unlock();
91
92         size_t start = get_time();
93         size_t last = time_stay_at_queue[i].size() - 1;
94         time_stay_at_queue[i][last] += start;
95
96         par_args args1 = {
97             obj, std::ref(queues[i + 1]), i,
98             std::ref(mutexes[i + 1]), false
99         };
100
101         par_args args2 = {
102             obj, std::ref(queues[i + 1]), i,
103             std::ref(mutexes[i + 1]), false
104         };
105
106         if (i == 0)
107             threads[i] = std::thread(&Conveyor::do_parallel_work1,
108                                     this, (void *) &args1);
109         else if (i == 1)
110             threads[i] = std::thread(&Conveyor::do_parallel_work2,
111                                     this, (void *) &args1);
112         else if (i == queue_count - 1)
113             threads[i] = std::thread(&Conveyor::do_parallel_work3,
114                                     this, (void *) &args2);
115     }
116 }
117 }
118
119 for (size_t i = 0; i < queue_count; i++)
120 {
121     if (threads[i].joinable())
122         threads[i].join();
123 }
124 }

```

## 3.4 Вывод

В данном разделе была рассмотрена конкретные реализации линейной и конвейерной обработки сложения матриц, необходимые для сравнительного анализа данных реализаций.

## 4. Экспериментальный раздел

В данном разделе приведены результаты работы двух различных реализаций обработки сложения матриц.

### 4.1 Сравнительный анализ

Все замеры проводились на процессоре 2.3 GHz Intel Core i5 с памятью 12 ГБ. В таблице 4.1 и на графике 4.1 представлены результаты измерения времени работы линейной и конвейерной реализации обработки сложения матриц.

Таблица 4.1: Время работы различных методов обработки в миллисекундах

Количество объектов	Линейная обработка	Конвейерная обработка
50	1499	1522
100	2896	3025
200	6455	6047
300	12236	9242
400	16805	13934
500	22768	18497
600	26723	22460
700	35227	28188
800	45388	34728
900	59026	42102
1000	68211	49761

Сравнение времени работы приведены для сложения квадратных матриц размера 1038x1038. Такая размерность матрицы была выбрана, из-за того, что реализация линейной и конвейерной обработки основывается на трех очередях, и чтобы загрузить каждую очередь одинаково, нужно выбрать размерность матрицы кратную трем. В нашем случае каждому этапу обработки достается сложение 346 элементов.

### 4.2 Вывод

По данным эксперимента можно сделать вывод о том, что линейная обработка оказалась менее эффективной, чем конвейерная. На небольшом количестве объектов эффективность конвейерной обработки не заметна. Это связано с тем, что значительную часть времени работы программы конвейерной обработки занимает инициализация потоков. Но на больших объемах входных данных (1000 обрабатываемых объектов) линейная обработка работает в 1.37 раза дольше.

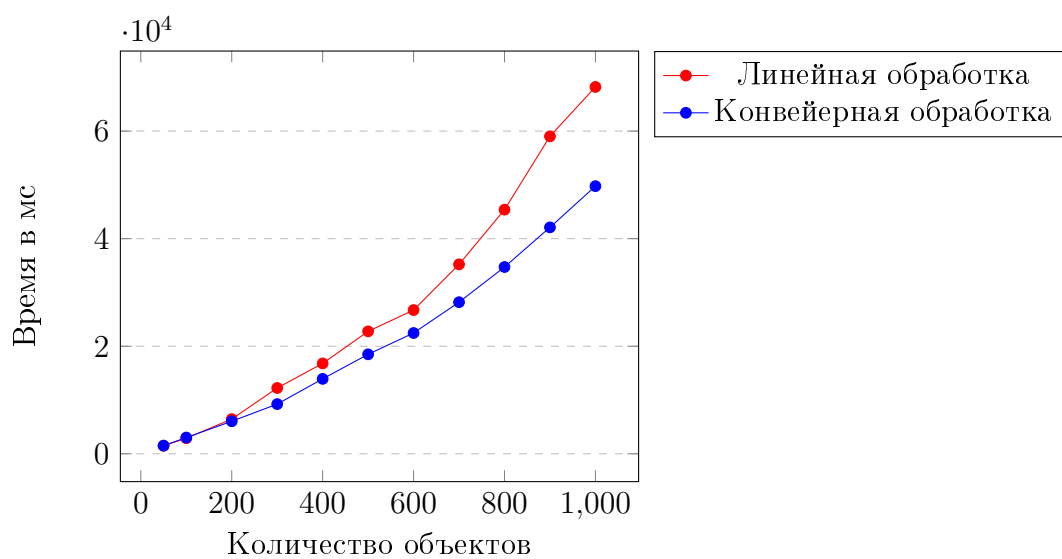


Рис. 4.1: График времени работы различных методов обработки в миллисекундах

## Заключение

В рамках лабораторной работы была рассмотрена и изучена конвейерная обработка данных. Благодаря ней возможна крайне удобная реализация задач, требующих поэтапной обработки некоторого набора данных, а также в некоторых случаях позволяет значительно ускорить выполнение программы

# Литература

[1] <https://cprference.com/> [Электронный ресурс]