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

Выполнил: Сорокин А.П., гр. ИУ7-52Б

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

Москва, 2019 г.

Оглавление

Введение

Сам термин «конвейер» пришёл из промышленности, где используется аналогичный принцип работы — материал автоматически подтягивается по ленте конвейера к рабочему, который осуществляет с ним необходимые действия, следующий за ним рабочий выполняет свои функции над получившейся заготовкой, следующий делает еще что-то, таким образом, к концу конвейера цепочка рабочих полностью выполняет все поставленные задачи, не срывая, однако, темпов производства. Например, если на самую медлительную операцию затрачивается одна минута, то каждая деталь будет сходить с конвейера через одну минуту.

Идея заключается в разделении обработки компьютерной инструкции на последовательность независимых стадий с сохранением результатов в конце каждой стадии. Это позволяет управляющим цепям процессора получать инструкции со скоростью самой медленной стадии обработки, однако при этом намного быстрее, чем при выполнении эксклюзивной полной обработки каждой инструкции от начала до конца.

1. Аналитический раздел

В данном разделе будет описан принцип конвейерной обработки.

1.1 Цель и задачи

Цель лабораторной работы: изучений конвейрной обработки. Для достижения этой цели были поставлены следующие задачи:

1. разработка и реализация алгоритмов
2. исследование работы конвейерной обработки с использование многопоточности и без
3. описание и обоснование полученных результатов

1.2 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].

1.3 Вывод

В данном разделе был описан принцип конвейерной обработки.

2. Технологический раздел

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

2.1 Требования к программному обеспечению

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

2.2 Средства реализации

Для реализации программы был использован язык C++ [2]. Для замера процессорного времени была использована функция rdtsc() из библиотеки stdrin.h.

2.3 Листинг кода

В листинге 2.1 описан класс конвейера. В листинге 2.2 представлена реализация линейной и конвейерной обработки матриц, а в листинге 2.3 - параллельной.

Листинг 2.1: 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 2.2: 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);
```

```
  }
}

void Conveyor::execute_linear()
{

  std::queue<MatrixSet> obj_generator;

  for (size_t i = 0; i < obj_count; i++)
  obj_generator.push(MatrixSet(i + 1, 1038, -200, 200));

  std::vector<MatrixSet> obj_pools;

  while (obj_pools.size() != obj_count)
  {
    MatrixSet obj = obj_generator.front();
    obj_generator.pop();

    for (size_t i = 0; i < queue_count; i++)
    {
      if (i == 0)
        do_linear_work1(obj, i);
      else if (i == 1)
        do_linear_work2(obj, i);
      else if (i >= 2)
        do_linear_work3(obj, i);
    }

    obj_pools.push_back(obj);
  }
}
```

Листинг 2.3: Реализация параллельной обработки матрицы

```
void* Conveyor::do_parallel_work1(void *_args)
{
  par_args *args = (par_args*) _args;
  size_t start = get_time();
  args->obj.sum(0, args->obj.size / 3);

  args->mutex.lock();
  args->queue.push(args->obj);
  args->mutex.unlock();

  size_t end = get_time();
  if (args->log)
    log_print_time(args->obj, args->queue_num, end - start);
  time_stay_at_queue[args->queue_num + 1].push_back(-end);
  return NULL;
}

void* Conveyor::do_parallel_work2(void *_args)
{
  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 }
```

2.4 Вывод

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

3. Экспериментальный раздел

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

3.1 Сравнительный анализ

Все замеры проводились на процессоре 2.3 GHz Intel Core i5 с памятью 12 ГБ. В таблице 3.1 и на графике 3.1 представлены результаты измерения времени работы линейной и конвейерной реализации обработки сложения матриц.

Таблица 3.1: 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.

3.2 Вывод

По данным эксперимента можно сделать вывод о том, что линейная обработка оказалась менеее 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.

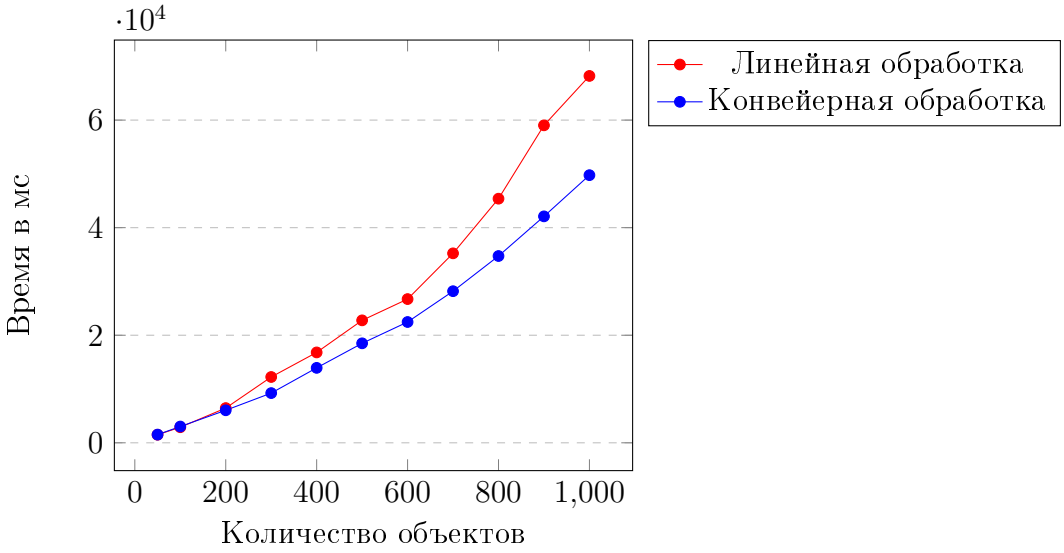

Рис. 3.1: График времени работы различных методов обработки в миллисекундаха

Заключение

В ходе выполнения данной лабораторной работы были изучены принципы конвейерной обработки. Было проведено исследование работы алгоритма при различных параметрах, показавшее, что конвейерная обработка работает значительно быстрее, чем линейная обработка (в 1.37 раза быстрее при количестве объектов, равном 1000).

Литература

[1] ISO/IEC JTC1 SC22 WG21 N 3690 «Programming Languages — C++» [Электронный ресурс]. https://devdocs.io/cpp/

[2] https://cppreference.com/ [Электронный ресурс]