Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ, Информатика и системы управления

КАФЕДРА ИУ7, Программное обеспечение ЭВМ и информационные технологии

ЛАБОРАТОРНАЯ РАБОТА №5

ПО ДИСЦИПЛИНЕ

“Анализ алгоритмов”

Студент ИУ7-54Б _____ **А.А. Андреев**
(Группа) (Подпись, дата) (И.О.Фамилия)

Преподаватель _____ **Л.Л. Волкова**
(Подпись, дата) (И.О.Фамилия)

2021 г.

Оглавление

Введение.

Данная лабораторная работа посвящена исследованию конвейерной обработки.

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 выпол- няет 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 последова- тельность независимых стадий с сохранением результатов в конце каждой стадии. Это позволяет управляющим цепям процессора получать инструкции со скоростью самой мед- ленной стадии обработки, однако при этом намного быстрее, чем при выполнении эксклю- зивной полной обработки каждой инструкции от начала до конца.

Цель данной лабораторной работы: исследование конвейерной обработки.

Задачи данной лабораторной работы:

1. Разработка и реализация алгоритмов;
2. Исследование работы конвейерной обработки с использованием многопоточности и без;
3. Описание и обоснование полученных результатов;

1. Аналитическая часть

В данном разделе будет описана теоретическая основа конвейерной обработки.

1.1 Конвейерная обработка

Если задача заключается в применении одной последовательности операций ко мно- гим 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 распре- делить 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 по- токами задачи, а не данные, мы изменяем качественные показатели производительности [1].

1.2 Описание задачи

В качестве алгоритма, реализованного для распределения на конвейере, был выбран алгоритма сложения матриц.

Вывод

В данном разделе был рассмотрен принцип конвейерной обработки.

2. Конструкторская часть.

В данном разделе будет приведены блок-схемы алгоритмов, описанных в аналитическом разделе п.1.

2.1 Разработка алгоритмов

На рисунке 2.1 приведена схема организации конвейерных вычислений.

Рисунок 2.1: Схема реализации конвейерных вычислений

Вывод

В данном разделе были рассмотрены блок-схемы, которые позволяют перейти к технологической части.

3. Технологическая часть.

В данном разделе будут рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач.

3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать поставленную на лабораторную работу задачу. Интерфейс для взаимодействия с программой - командная строка. Пользователь должен иметь возможность вводить количество объектов, которые будут обрабатываться.

3.2 Выбор и обоснование языка и среды программирования.

Для разработки данной программы применён язык C++ [2] с функцией rdtsc() из библиотеки stdrin.h [3] для вычисления времени работы процессора, чтобы расширить знания в области данного языка программирования.

3.3 Реализация алгоритмов

В листингах 1-9 приведена реализация алгоритмов умножения матриц.

Листинг 3.1: Класс конвейера (Conveyor.hpp), Часть 1

```
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);
```

Листинг 3.2: Класс конвейера (Conveyor.hpp), Часть 2

```
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 3.3: Линейная обработка матрицы (Conveyer.cpp), Часть 1

```
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);
```

Листинг 3.4: Линейная обработка матрицы (Conveyer.cpp), Часть 2

```
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 3.5: Параллельная обработка матрицы (Conveyer.cpp), Часть 1

```
1. void* Conveyor::do_parallel_work1(void *_args)
2. {
3.    par_args *args = (par_args*) _args;
4.    size_t start = get_time();
5.
6.    args->obj.sum(0, args->obj.size / 3);
7.
8.    args->mutex.lock();
9.    args->queue.push(args->obj);
10.      args->mutex.unlock();
11.
12.      size_t end = get_time();
13.      if (args->log)
14.          log_print_time(args->obj, args->queue_num, end -
   start);
15.      time_stay_at_queue[args->queue_num + 1].push_back(-end);
16.      return NULL;
17.   }
18.
19.   void* Conveyor::do_parallel_work2(void *_args)
20.   {
21.      par_args *args = (par_args*) _args;
22.      size_t start = get_time();
23.
24.      args->obj.sum(args->obj.size / 3, 2 * args->obj.size / 3);
25.
26.      args->mutex.lock();
27.      args->queue.push(args->obj);
28.      args->mutex.unlock();
29.
30.      size_t end = get_time();
31.      if (args->log)
32.          log_print_time(args->obj, args->queue_num, end -
   start);
33.      time_stay_at_queue[args->queue_num + 1].push_back(-end);
34.      return NULL;
35.   }
36.
37.   void* Conveyor::do_parallel_work3(void *_args)
38.   {
39.      par_args *args = (par_args*) _args;
40.      size_t start = get_time();
41.
42.      args->obj.sum(2 * args->obj.size / 3, args->obj.size);
43.
44.      args->mutex.lock();
45.      args->queue.push(args->obj);
46.      args->mutex.unlock();
47.
48.      size_t end = get_time();
49.      if (args->log)
50.          log_print_time(args->obj, args->queue_num, end -
```

Листинг 3.6: Параллельная обработка матрицы (Conveyer.cpp), Часть 2

```
51.   start);
52.      time_stay_at_queue[args->queue_num + 1].push_back(-end);
53.      return NULL;
54.   }
55.
56.   void Conveyor::execute_parallel()
57.   {
58.      std::queue<MatrixSet> obj_generator;
59.
60.      for (size_t i = 0; i < obj_count; i++)
61.          obj_generator.push(MatrixSet(i + 1, 1038, -200, 200));
62.
63.      std::vector<MatrixSet> obj_pool;
64.
65.      std::vector<std::thread> threads(3);
66.      std::vector<std::queue<MatrixSet> > queues(3);
67.      std::vector<std::mutex> mutexes(4);
68.      size_t prev_time = get_time() - delay_time;
69.
70.      while (obj_pool.size() != obj_count)
71.      {
72.          size_t cur_time = get_time();
73.
74.          if (!obj_generator.empty() && prev_time + delay_time <
   cur_time)
75.          {
76.              MatrixSet obj = obj_generator.front();
77.              obj_generator.pop();
78.              queues[0].push(obj);
79.
80.              prev_time = get_time();
81.              time_stay_at_queue[0].push_back(-prev_time);
82.          }
83.
84.          for (unsigned i = 0; i < queue_count; i++)
85.          {
86.              if (threads[i].joinable())
87.                  threads[i].join();
88.
89.              if (!queues[i].empty() && !threads[i].joinable())
90.              {
91.                  mutexes[i].lock();
92.                  MatrixSet obj = queues[i].front();
93.                  queues[i].pop();
94.                  mutexes[i].unlock();
95.
96.                  size_t start = get_time();
97.                  size_t last = time_stay_at_queue[i].size() -
   1;
98.                  time_stay_at_queue[i][last] += start;
99.
100.                 par_args args1 = {
101.                     obj, std::ref(queues[i + 1]), i,
```

Листинг 3.7: Параллельная обработка матрицы (Conveyer.cpp), Часть 3

```
102.         std::ref(mutexes[i + 1]), false
103.                 };
104.
105.                 par_args args2 = {
106.                     obj, std::ref(queues[i + 1]), i,
107.                     std::ref(mutexes[i + 1]), false
108.                 };
109.
110.                 if (i == 0)
111.                 {
112.                     threads[i] =
    std::thread(&Conveyor::do_parallel_work1,
113.                             this, (void *) &args1);
114.                 }
115.                 else if (i == 1)
116.                     threads[i] =
    std::thread(&Conveyor::do_parallel_work2,
117.                             this, (void *) &args1);
118.                 else if (i == queue_count - 1)
119.                     threads[i] =
    std::thread(&Conveyor::do_parallel_work3,
120.                             this, (void *) &args2);
121.             }
122.         }
123.     }
124.
125.     for (size_t i = 0; i < queue_count; i++)
126.     {
127.         if (threads[i].joinable())
128.             threads[i].join();
129.     }
```

Вывод

В данном разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач.

4. Исследовательская часть.

4.1 Демонстрация работы программы

Пример работы программы представлен на рисунке 4.1.

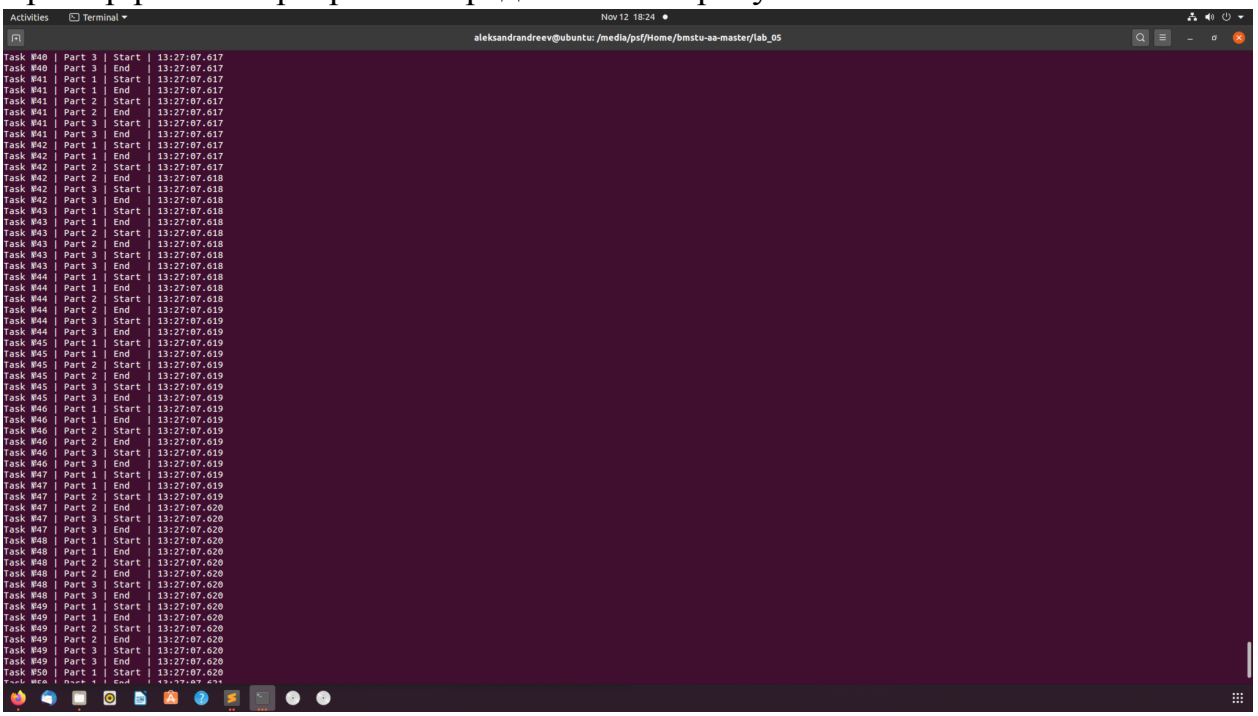

Рисунок 4.1: Демонстрация работы программы

4.2 Технические характеристики

В Таблице 4.2 приведены технические характеристики ЭВМ, на котором проводилось тестирование разрабатываемого программного обеспечения.

Таблица 4.2: Технические характеристики ЭВМ, на котором проводилось тестирование разрабатываемого программного обеспечения

ОС	Mac OS Mojave 64-bit
ОЗУ	8 Gb 2133 MHz LPDDR3
Процессор	2,3 GHz Intel Core i5

4.3 Время выполнения алгоритмов

В Таблице 4.3. приведена информация о времени выполнения алгоритмов на случайных данных в микросекундах. Каждый замер проводился 10 раз, результат усреднялся.

Таблица 4.3: Таблица времени выполнения алгоритмов на случайных данных с четной размерностью (в микросекундах)

№	Количество объектом матриц размером 1038x1038	Время	
		Линейная обработка	Конвейерная обработка
1	50	1499	1522
2	100	2896	3025
3	200	6455	6047
4	300	12236	9242
5	400	16805	13934
6	500	22768	18497
7	600	26723	22460
8	700	35227	28188
9	800	45388	34728
10	900	59026	42102
11	1000	68211	49761

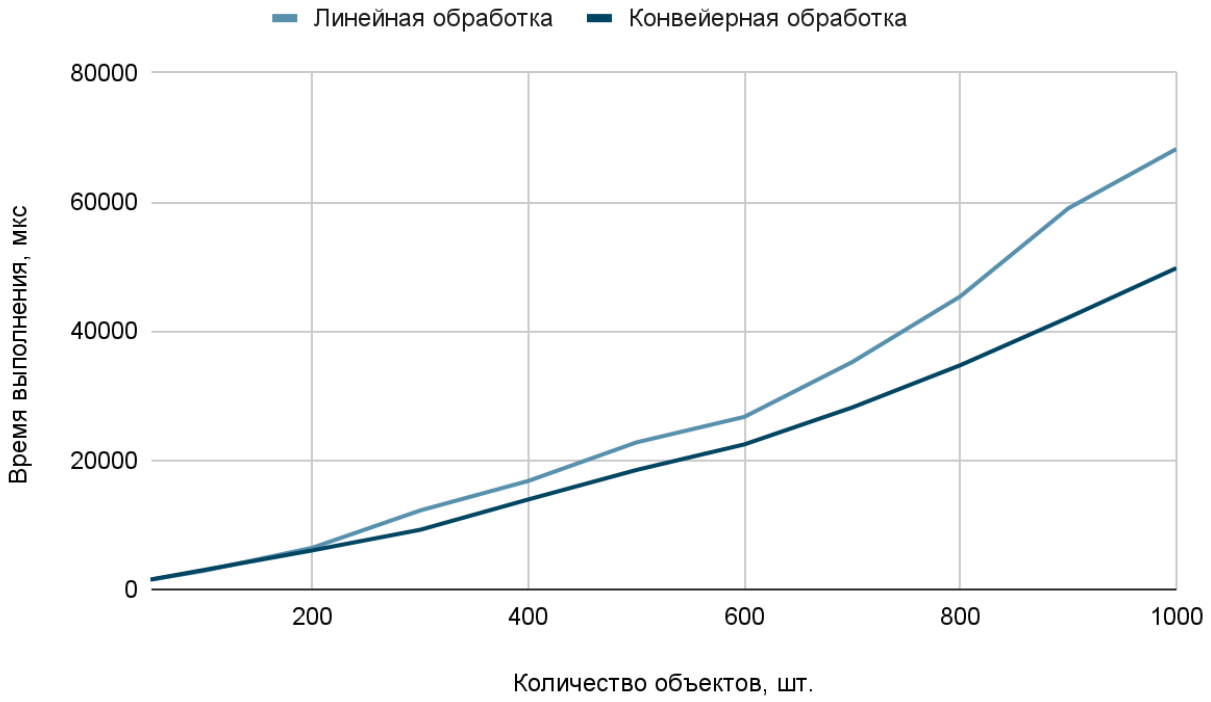

Рисунок 4.3: График времени выполнения алгоритмов на случайных данных (в микросекундах) на основе Таблицы 4.3

Вывод

В данном разделе были протестированы принципы конвейерной обработки, которая показала себя эффективнее линейной в 1.37 при количестве объектов, равном 1000.

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 об- работка (в 1.37 раза быстрее при количестве объектов, равном 1000).

Список использованной литературы

[1] Кормен Т.Х., Лейзерсон Ч.И., Алгоритмы: Построение и анализ, год выпуска 2019, тираж 1328, 700 страниц.

[2] ISO/IEC JTC1 SC22 WG21 N 3690 «Programming Languages — C++» [Электронный ресурс]. https://devdocs.io/cpp/ Дата обращения: 13.09.2021

[3] Гасфилд, Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. Невский Диалект БВХ-Петербург, год выпуска 2003, типаж 900, 653 страницы.

[4] Вычисление процессорного времени выполенения программы [Электронный ресурс] Режим доступа: https://www.tutorialspoint.com/python/time_clock.htm. Дата обращения: 13.09.2021