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 «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт по лабораторной работе №5 по дисциплине "Анализ алгоритмов"

Тема Конвейерные вычисления

Студент Михаил Коротыч

Группа ИУ7-55Б

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

Москва — 2021 г.

Содержание

Введение

При обработке данных могут возникать ситуации, когда один набор данных необходимо обработать последовательно несколькими алгоритмами. В таком случае удобно использовать конвейерную обработку данных, что позволяет на каждой следующей «линии» конвейера использовать данные, полученные с предыдущего этапа. Помимо линейной конвейерной обработки данных, существуют параллельная конвейерная обработка данных. При таком подходе все линии работают с меньшим времени простоя, так как могут обрабатывать задачи независимо от других линий.

Целью данной лабораторной работы является изучение и реализация параллельной и линейной реализации конвейерной обработки данных. В рамках выполнения работы необходимо решить следующие задачи:

- изучить конвейерную обработку данных;
- реализовать систему конвейерных вычислений с количеством линий не меньше трёх;
- сравнить параллельную и линейную реализацию конвейерных вычислений;
- сделать выводы на основе проделанной работы.

1 Аналитическая часть

В данном разделе представленные теоретические сведения о рассматриваемых алгоритмах.

1.1 Конвейерная обработка данных

Конвейер - система поточного производства. В терминах программирования ленты конвейера представлены функциями, выполняющими над неким набором данных операции и передающие их на следующую ленту конвейера. Моделирование конвейерной обработки хорошо сочетается с технологией многопоточного программирования - под каждую ленту конвейера выделяется отдельный поток, все потоки работают в асинхронном режиме.

1.2 Описание задачи

В качестве алгоритма, реализованного для распределения на конвейере, был выбран процес сборки автомобиля, состоящий из трёх этапов:

- сборка движка (возведение числа в степень);
- сборка корпуса (проверка числа на простоту);
- сборка колёс (вычисление числа Фибоначчи).

1.3 Вывод

Ввод в программе не предусматривается. Результат представляет собой таблицу, в которой представлены количество задач, конвейеры и общее время работы в миллисекундах. К программе предъявляется ряд требований:

- на вход алгоритму подаётся количество задач (количество машин, которые нужно собрать);
- на выходе - время, затраченное на обработку заявок;
- в процессе обрабатывания задач необходимо фиксировать время прихода и ухода заявки с линии.

В разделе были рассмотрены особенности построения конвейерных вычислений, описание решаемой задачи.

2 | Конструкторская часть

В данном разделе представлены схемы рассматриваемых алгоритмов.

2.1 Разработка алгоритмов

На рисунке 2.1 приведена схема организации конвейерных вычислений.

Рис. 2.1: Схема организации конвейерных вычислений.

2.2 Использованные структуры данных

Для задачи по сборке движка, корпуса и колёс были созданы классы `Engine`, `Carcass` и `Wheels` с основными характеристиками соответствующих механизмов. Класс `Car` является

централизованным классом, который содержит экземпляры всех трёх остальных классов. Для логирования был создан класс `Logger`. Он содержит метод, отвечающий за вывод данных в таблицу.

2.3 Вывод

На основе теоретических данных, полученных из аналитического раздела, была построена схема алгоритма конвейерных вычислений.

3 | Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Средства реализации

Для реализации ПО я выбрал язык программирования C++ [1]. Данный выбор обусловлен не только моим опытом разработки программ на этом языке, но также и моим желанием расширить свои знания в области применения данного языка программирования.

3.2 Реализация алгоритмов

В листингах 3.1 и 3.2 приведены реализации конвейерных вычислений (класс `Cloveyor`), реализация сборки машины (класс `Car`) и реализация класса отвечающего за логирование (класс `Logger`). На рисунках 2.2-2.5 представлены интерфейс созданных классов.

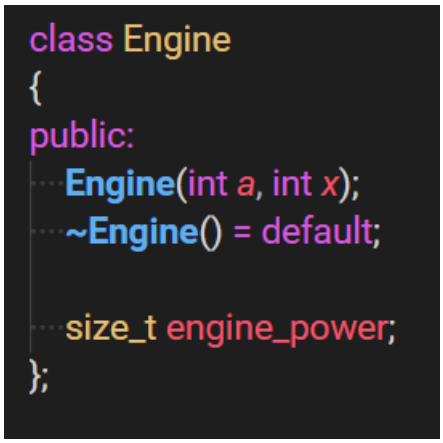

Рис. 3.1: Скриншот класса.

```
class Carcass
{
public:
    Carcass(size_t num);
    ~Carcass() = default;

    bool is_freight;
};
```

Рис. 3.2: Скриншот класса.

```
class Wheels
{
public:
    Wheels(int n);
    ~Wheels() = default;

    size_t wheels_cnt;
};
```

Рис. 3.3: Скриншот класса.

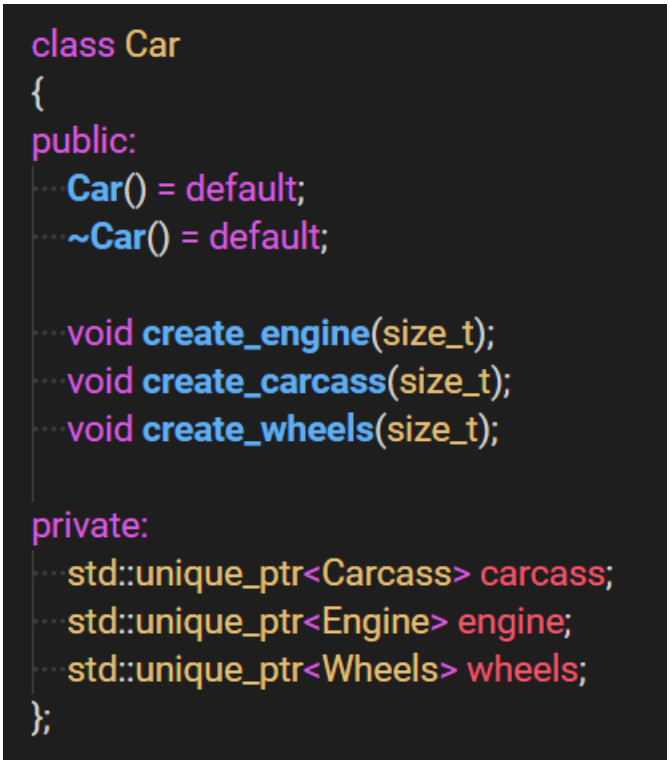

Рис. 3.4: Скриншот класса.

Листинг 3.1: Реализация класса конвейера

```
#include <thread>
#include <queue>

#include "car.h"

#define THRD_CNT 3

class Conveyor
{
public:
  Conveyor() = default;
  ~Conveyor() = default;

  void run(size_t cars_cnt);

  void create_engine();
  void create_carcass();
  void create_wheels();

private:
  std::thread threads[THRD_CNT];
  std::vector<std::shared_ptr<Car>> cars;

  std::queue<std::shared_ptr<Car>> q1;
```

```
25    std::queue<std::shared_ptr<Car>> q2;
26    std::queue<std::shared_ptr<Car>> q3;
27  };
28
29  void Conveyor::run_parallel(size_t cars_cnt)
30  {
31    for (size_t i = 0; i < cars_cnt; ++i)
32    {
33      std::shared_ptr<Car> new_car(new Car);
34      cars.push_back(new_car);
35      q1.push(new_car);
36    }
37
38    this->threads[0] = std::thread(&Conveyor::create_carcass, this);
39    this->threads[1] = std::thread(&Conveyor::create_engine, this);
40    this->threads[2] = std::thread(&Conveyor::create_wheels, this);
41
42    for (int i = 0; i < THRD_CNT; ++i)
43      this->threads[i].join();
44  }
45
46  void Conveyor::run_linear(size_t cars_cnt)
47  {
48    for (size_t i = 0; i < cars_cnt; ++i)
49    {
50      std::shared_ptr<Car> new_car(new Car);
51      cars.push_back(new_car);
52      q1.push(new_car);
53    }
54
55    for (size_t i = 0; i < cars_cnt; ++i)
56    {
57      std::shared_ptr<Car> car = q1.front();
58      car->create_carcass(i + 1);
59      q2.push(car);
60      q1.pop();
61
62      car = q2.front();
63      car->create_engine(i + 1);
64      q3.push(car);
65      q2.pop();
66
67      car = q3.front();
68      car->create_wheels(i + 1);
69      q3.pop();
70    }
71  }
72
73  void Conveyor::create_carcass()
74  {
```

```
75    size_t task_num = 0;
76
77    while (!this->q1.empty())
78    {
79      std::shared_ptr<Car> car = q1.front();
80      car->create_carcass(++task_num);
81
82      q2.push(car);
83      q1.pop();
84    }
85  }
86
87  void Conveyor::create_engine()
88  {
89    size_t task_num = 0;
90
91    do
92    {
93      if (!this->q2.empty())
94      {
95        std::shared_ptr<Car> car = q2.front();
96        car->create_engine(++task_num);
97
98        q3.push(car);
99        q2.pop();
100     }
101   } while (!q1.empty() || !q2.empty());
102 }
103
104 void Conveyor::create_wheels()
105 {
106   size_t task_num = 0;
107
108   do
109   {
110     if (!this->q3.empty())
111     {
112       std::shared_ptr<Car> car = q3.front();
113       car->create_wheels(++task_num);
114       q3.pop();
115     }
116   } while (!q1.empty() || !q2.empty() || !q3.empty());
117 }
```

Листинг 3.2: Реализация класса сборки машины

```
1 #include <memory>
2 #include <cmath>
3
4 #include "logger.h"
```

```
5
6  class Carcass
7  {
8  public:
9    Carcass(size_t num);
10   ~Carcass() = default;
11
12   bool is_freight;
13 };
14
15 class Engine
16 {
17 public:
18   Engine(int a, int x);
19   ~Engine() = default;
20
21   size_t engine_power;
22 };
23
24 class Wheels
25 {
26 public:
27   Wheels(int n);
28   ~Wheels() = default;
29
30   size_t wheels_cnt;
31 };
32
33 class Car
34 {
35 public:
36   Car() = default;
37   ~Car() = default;
38
39   void create_engine(size_t);
40   void create_carcass(size_t);
41   void create_wheels(size_t);
42
43 private:
44   std::unique_ptr<Carcass> carcass;
45   std::unique_ptr<Engine> engine;
46   std::unique_ptr<Wheels> wheels;
47 };
48
49 Carcass::Carcass(size_t num)
50 {
51   this->is_freight = false;
52
53   for (size_t i = 2; i <= sqrt(num); ++i)
54     if (!(num % i))
```

```
55        return;
56
57    this->is_freight = true;
58  }
59
60  Engine::Engine(int a, int x)
61  {
62    this->engine_power = a;
63
64    for (int i = 0; i < x; i++)
65      this->engine_power *= a;
66  }
67
68  Wheels::Wheels(int n)
69  {
70    size_t f1 = 1, f2 = 1;
71    this->wheels_cnt = f1;
72
73    for (int i = 2; i < n; ++i)
74    {
75      this->wheels_cnt = f1 + f2;
76      f1 = f2;
77      f2 = this->wheels_cnt;
78    }
79  }
80
81  void Car::create_engine(size_t task_num)
82  {
83    Logger::log_current_event(task_num, "Part 2 | Start");
84
85    if (this->carcass->is_freight)
86      this->engine = std::unique_ptr<Engine>(new Engine(10, 150000));
87    else
88      this->engine = std::unique_ptr<Engine>(new Engine(5,  150000));
89
90    Logger::log_current_event(task_num, "Part 2 | End  ");
91  }
92
93  void Car::create_carcass(size_t task_num)
94  {
95    Logger::log_current_event(task_num, "Part 1 | Start");
96    this->carcass = std::unique_ptr<Carcass>(new Carcass(27644437));
97    Logger::log_current_event(task_num, "Part 1 | End  ");
98  }
99
100 void Car::create\_wheels(size\_t task\_num)
101 {
102   Logger::log_current_event(task_num, "Part 3 | Start");
103   this->wheels = std::unique_ptr<Wheels>(new Wheels(this->engine->engine\
      _power));
```

```
104   Logger::log_current_event(task_num, "Part 3 | End  ");
105 }
```

Листинг 3.3: Реализация класса логирования

```
1  #include <iostream>
2  #include <chrono>
3
4  using namespace std::chrono;
5
6  class Logger
7  {
8  public:
9    Logger() = default;
10   ~Logger() = default;
11
12   static void log_current_event(size_t task_num, const char *const event);
13 }
14
15 void Logger::log_current_event(size_t task_num, const char *const event)
16 {
17   system_clock::time_point now = system_clock::now();
18   system_clock::duration tp = now.time_since_epoch();
19
20   tp -= duration_cast<seconds>(tp);
21
22   time_t tt = system_clock::to_time_t(now);
23   tm t = *gmtime(&tt);
24
25   std::printf
26   (
27     "Task #%lu | %s | %02u:%02u:%02u.%3u\n",
28     task_num,
29     event,
30     t.tm_hour,
31     t.tm_min,
32     t.tm_sec,
33     static_cast<unsigned>(tp / milliseconds(1))
34   );
35 }
```

3.3 Вывод

В данном разделе была разработана и рассмотрена реализация конвейерных вычислений.

4 | Исследовательская часть

В данном разделе приведен анализ характеристик разработанного ПО и примеры работы ПО.

4.1 Пример работы программы

4.2 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- операционная система: Windows 10 21H2;
- оперативная память: 8 ГБ;
- процессор: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz. [2].

4.3 Время выполнения алгоритмов

Время выполнения алгоритма замерялось с помощью применения технологии профайлинга [3]. Данный инструмент даёт детальное описание количество вызовов и количества времени CPU, затраченного на выполнение каждой функции.

В таблице 4.1 приведено сравнение времени выполнения параллельной обработки данных (сборка машины), в зависимости от количества входных задач (количества машин). Линия №1 - сборка каркасов автомобилей (проверка числа на простоту), линия №2 - сборка двигателей автомобилей (возведение числа в степень), линия №3 - сборка колёс автомобилей (вычисление числа Фибоначчи). Время указано в секундах.

На рисунке 4.2 представлен график зависимости времени от количества задач для линейной и параллельной обработки конвейера.

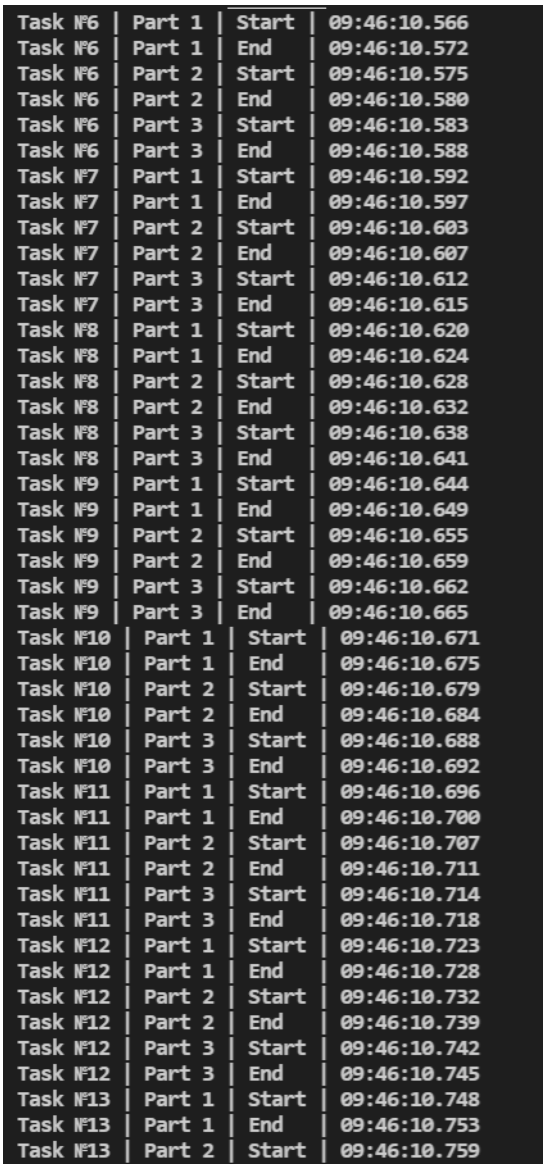

Рис. 4.1: Пример работы программы (параллельная обработка)

Таблица 4.1: Таблица времени выполнения параллельной обработки данных, время в секундах

№ линии	Task M	Начальное время	Конечное время
1	1	19:18:41.210	19:18:41.220
1	2	19:18:41.224	19:18:41.228
1	3	12:43:08.020	12:43:08.224
2	1	19:18:41.224	19:18:41.232
2	2	19:18:41.241	19:18:41.252
2	3	19:18:41.264	19:18:41.275
3	1	19:18:41.241	19:18:41.248
3	2	19:18:41.264	19:18:41.267
3	3	19:18:41.290	19:18:41.294
4	1	19:18:41.260	19:18:41.279
4	2	19:18:41.297	19:18:41.320
4	3	19:18:41.330	19:18:41.339

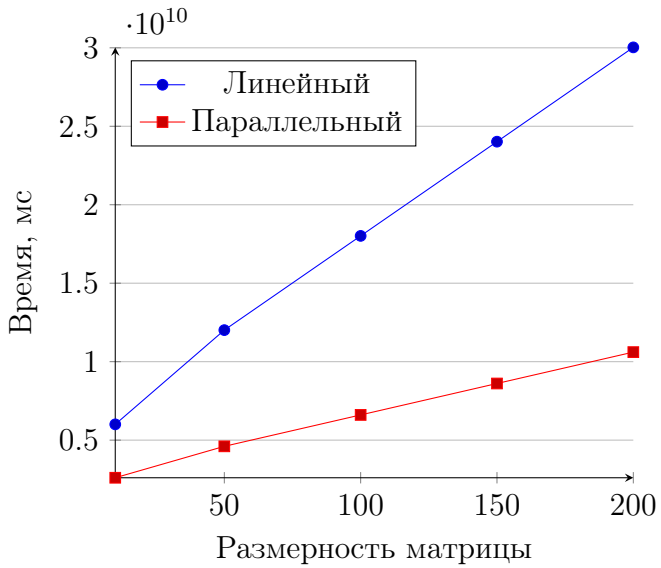

Рис. 4.2: Зависимость времени работы реализации конвейеров от количества задач

4.4 Вывод

В данном разделе приведены время исполнения параллельного алгоритма и его сравнение с линейной реализацией конвейера. Как видно из таблицы 4.1, вторая линия, то есть сборка двигателей (возведение числа в степень) занимает в среднем 70% времени от выполнения всей программы. Линия №3 в среднем работает быстрее чем линия №1.

Параллельная реализация конвейерной обработки выигрывает по времени исполнения у линейной реализации. Как видно из рисунка 4.2, линейная реализация работает в 3 раза медленнее при 200 задачах.

Заключение

В рамках данной лабораторной работы лабораторной работы была достигнута её цель: изучена параллельная и линейная реализация конвейерной обработки данных. Также выполнены следующие задачи:

- изучена конвейерная обработка данных;
- реализована система конвейерных вычислений с количеством линий не меньше трёх;
- сравнены параллельные и линейные реализации конвейерных вычислений;
- сделаны выводы на основе проделанной работы;

Параллельные конвейерные вычисления позволяют организовать непрерывную обработку данных, что позволяет выиграть время в задачах, где требуется обработка больших объёмов данных за малый промежуток времени.

Литература

[1] C++ Standard. Режим доступа: https://isocpp.org/. Дата обращения: 01.12.2020.

[2] Процессор Intel(R) Core(TM) i7-8550U [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/122589/intel-core-i7-8550u-processor-8m-cache-up-to-4-00-ghz.html. Дата обращения: 20.11.2021.

[3] GNU gprof – Introducing to Profiling. Режим доступа: https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_chapter/gprof_1.html. Дата обращения: 01.12.2020.

[4] Windows 10. Режим доступа: https://www.microsoft.com/ru-ru/windows/get-windows-10. Дата обращения: 21.11.2021

[5] Вычислительные конвейеры. Режим доступа: http://kspt.icc.spbstu.ru/media/files/2018/kuzmin/Ch
Дата обращения 21.11.2021