



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 Романов А.В.

Группа ИУ7-53Б

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

Москва — 2020 г.

Оглавление

Введение	2
1 Аналитическая часть	3
1.1 Конвейерная обработка данных	3
1.2 Описание задачи	3
2 Конструкторская часть	4
2.1 Разработка алгоритмов	4
3 Технологическая часть	5
3.1 Требование к ПО	5
3.2 Средства реализации	5
3.3 Реализация алгоритмов	5
4 Исследовательская часть	12
4.1 Пример работы программы	12
4.2 Технические характеристики	13
4.3 Время выполнения алгоритмов	13
Заключение	15
Литература	15

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.

Цель лабораторной работы

Целью данной лабораторной работы является изучение и реализация параллельной и линейной реализации конвейерной обработки данных.

Задачи лабораторной работы

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 представлены 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 процесс сборки автомобиля, состоящий из трех этапов:

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

Вывод

В данном разделе были рассмотрены особенности построения конвейерных вычислений.

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.

Вывод

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

3 | Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Требование к ПО

К программе предъявляется ряд требований:

- на вход подается 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.

3.2 Средства реализации

Для реализации ПО я выбрал язык программирования C++ [1]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного языка программирования.

3.3 Реализация алгоритмов

В листингах 3.1 и 3.2 приведены реализации конвейерных вычислений (класс `Conveyor`), реализация сборки машины (класс `Car`) и реализация класса отвечающего за логгирование (класс `Logger`).

Листинг 3.1: Реализация класса конвейера

```
1 #include <thread>
2 #include <queue>
3
4 #include "car.h"
5
6 #define THRD_CNT 3
7
8 class Conveyor
9 {
10 public:
11     Conveyor() = default;
```

```

12 ~Conveyor() = default;
13
14 void run(size_t cars_cnt);
15
16 void create_engine();
17 void create_carcass();
18 void create_wheels();
19
20 private:
21     std::thread threads[THRD_CNT];
22     std::vector<std::shared_ptr<Car>> cars;
23
24     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     {
44         this->threads[i].join();
45     }
46 }
47
48 void Conveyor::run_linear(size_t cars_cnt)
49 {
50     for (size_t i = 0; i < cars_cnt; i++)
51     {
52         std::shared_ptr<Car> new_car(new Car);
53         cars.push_back(new_car);
54         q1.push(new_car);
55     }
56
57     for (size_t i = 0; i < cars_cnt; i++)
58     {
59         std::shared_ptr<Car> car = q1.front();
60         car->create_carcass(i + 1);
61         q2.push(car);

```

```

62     q1.pop();
63
64     car = q2.front();
65     car->create_engine(i + 1);
66     q3.push(car);
67     q2.pop();
68
69     car = q3.front();
70     car->create_wheels(i + 1);
71     q3.pop();
72 }
73 }
74
75 void Conveyor::create_carcass()
76 {
77     size_t task_num = 0;
78
79     while (!this->q1.empty())
80     {
81         std::shared_ptr<Car> car = q1.front();
82         car->create_carcass(++task_num);
83
84         q2.push(car);
85         q1.pop();
86     }
87 }
88
89 void Conveyor::create_engine()
90 {
91     size_t task_num = 0;
92
93     do
94     {
95         if (!this->q2.empty())
96         {
97             std::shared_ptr<Car> car = q2.front();
98             car->create_engine(++task_num);
99
100             q3.push(car);
101             q2.pop();
102         }
103     } while (!q1.empty() || !q2.empty());
104 }
105
106 void Conveyor::create_wheels()
107 {
108     size_t task_num = 0;
109
110     do
111     {

```



```

112     if (!this->q3.empty())
113     {
114         std::shared_ptr<Car> car = q3.front();
115         car->create_wheels(++task_num);
116         q3.pop();
117     }
118 } while (!q1.empty() || !q2.empty() || !q3.empty());
119 }

```

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     {
55         if (0 == num % i)
56         {
57             return;
58         }
59     }
60
61     this->is_freight = true;
62 }
63
64 Engine::Engine(int a, int x)
65 {
66     this->engine_power = a;
67
68     for (int i = 0; i < x; i++)
69     {
70         this->engine_power *= a;
71     }
72 }
73
74 Wheels::Wheels(int n)
75 {
76     size_t f1 = 1, f2 = 1;
77     this->wheels_cnt = f1;
78
79     for (int i = 2; i < n; i++)
80     {
81         this->wheels_cnt = f1 + f2;
82         f1 = f2;
83         f2 = this->wheels_cnt;
84     }
85 }
86
87 void Car::create_engine(size_t task_num)
88 {
89     Logger::log_current_event(task_num, "Part 2 | Start");

```

```

90
91     if (this->carcass->is_freight)
92     {
93         this->engine = std::unique_ptr<Engine>(new Engine(10, 150000));
94     }
95     else
96     {
97         this->engine = std::unique_ptr<Engine>(new Engine(5, 150000));
98     }
99
100     Logger::log_current_event(task_num, "Part 2 | End ");
101 }
102
103 void Car::create_carcass(size_t task_num)
104 {
105     Logger::log_current_event(task_num, "Part 1 | Start");
106     this->carcass = std::unique_ptr<Carcass>(new Carcass(27644437));
107     Logger::log_current_event(task_num, "Part 1 | End ");
108 }
109
110 void Car::create_wheels(size_t task_num)
111 {
112     Logger::log_current_event(task_num, "Part 3 | Start");
113     this->wheels = std::unique_ptr<Wheels>(new Wheels(this->engine->engine\
114 _power));
115     Logger::log_current_event(task_num, "Part 3 | End ");
116 }

```

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     "Task #%lu | %s | %02u:%02u:%02u.%3u\n",
27     task_num,
28     event,
29     t.tm_hour,
30     t.tm_min,
31     t.tm_sec,
32     static_cast<unsigned>(tp / milliseconds(1))
33 );
34 }

```

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

```
Task №1 | Part 1 | Start | 13:02:57.318
Task №1 | Part 1 | End   | 13:02:57.318
Task №2 | Part 1 | Start | 13:02:57.318
Task №1 | Part 2 | Start | 13:02:57.318
Task №2 | Part 1 | End   | 13:02:57.318
Task №3 | Part 1 | Start | 13:02:57.318
Task №3 | Part 1 | End   | 13:02:57.319
Task №4 | Part 1 | Start | 13:02:57.319
Task №4 | Part 1 | End   | 13:02:57.319
Task №5 | Part 1 | Start | 13:02:57.319
Task №5 | Part 1 | End   | 13:02:57.319
Task №6 | Part 1 | Start | 13:02:57.319
Task №6 | Part 1 | End   | 13:02:57.319
Task №7 | Part 1 | Start | 13:02:57.319
Task №1 | Part 2 | End   | 13:02:57.319
Task №2 | Part 2 | Start | 13:02:57.319
Task №1 | Part 3 | Start | 13:02:57.319
Task №1 | Part 3 | End   | 13:02:57.319
Task №7 | Part 1 | End   | 13:02:57.319
Task №8 | Part 1 | Start | 13:02:57.319
Task №8 | Part 1 | End   | 13:02:57.320
Task №9 | Part 1 | Start | 13:02:57.320
Task №2 | Part 2 | End   | 13:02:57.320
Task №3 | Part 2 | Start | 13:02:57.320
Task №2 | Part 3 | Start | 13:02:57.320
Task №9 | Part 1 | End   | 13:02:57.320
Task №2 | Part 3 | End   | 13:02:57.320
```

Рис. 4.1: Пример работы программы (параллельная обработка)

4.2 Технические характеристики

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

- Операционная система: Debian [2] Linux [3] 11 «bullseye» 64-bit.
- Оперативная память: 12 GB.
- Процессор: Intel(R) Core(TM) i5-3550 CPU @ 3.30GHz [4].

4.3 Время выполнения алгоритмов

Время выполнения алгоритма замерялось с помощью применения технологии профайлинга [5]. 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.

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

Количество задач	Линия №1	Линия №2	Линия №3	Общее время работы
50	0.03	0.16	0.01	0.27
100	0.06	0.34	0.02	0.47
200	0.13	0.63	0.06	0.9
400	0.3	1.32	0.15	1.86
800	0.63	2.45	0.31	3.45

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

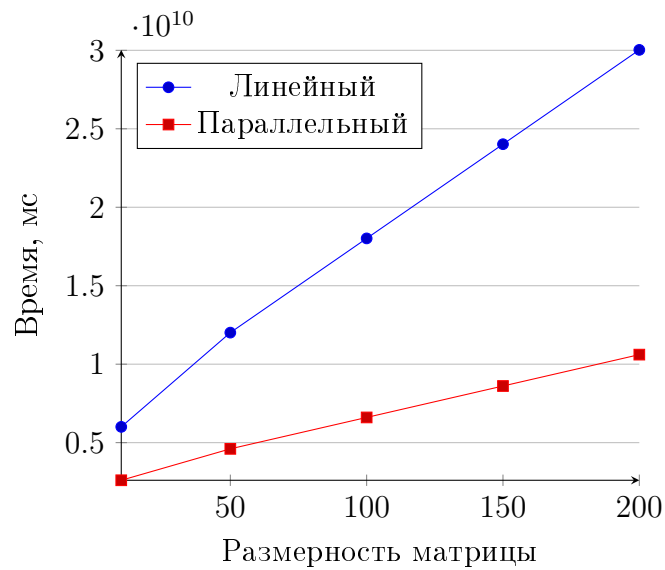


Рис. 4.2: Зависимость времени работы реализации конвейеров от количества задач

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 у линейной реализации. Как видно из рисунка 4.2, линейная реализация работает в 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 объемов данных за малый промежуток времени.

Литература

- [1] C++ Standard. Режим доступа: <https://isocpp.org/>. Дата обращения: 01.12.2020.
- [2] Debian – универсальная операционная система [Электронный ресурс]. Режим доступа: <https://www.debian.org/>. Дата обращения: 20.09.2020.
- [3] Linux – Getting Started [Электронный ресурс]. Режим доступа: <https://linux.org>. Дата обращения: 20.09.2020.
- [4] Процессор Intel® Core™ i5-3550 [Электронный ресурс]. Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/65516/intel-core-i5-3550-processor-6m-cache-up-to-3-70-ghz.html>. Дата обращения: 20.09.2020.
- [5] 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.