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 Костев Д.И.

Группа ИУ7-51Б

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

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

1.1 Конвейерная обработка данных

Конвейер - система поточного производства. В терминах программирования ленты конвейера представлены функциями, выполняющими над неким набором данных операции и предающие 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, была использована аналогия с этапами сборки автомобилей. Сопоставление этапов приведено ниже.

1. Сборка двигателя (возведение числа в степень).
2. Сборка корпуса (проверка числа на простоту).
3. Сборка колёс (вычисление числа Фибоначчи).

Вывод

В данном разделе были рассмотрены особенности построения конвейерных вычислений.

2 Конструкторская часть

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 аз аналитического раздела, были построенна 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++.

3.3 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 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      //
58      tm start_1;
59      size_t str1;
60      tm stop_1;
61      size_t stp1;
62
63      tm start_2;
```

```
64      size_t str2;
65      tm stop_2;
66      size_t stp2;
67
68      tm start_3;
69      size_t str3;
70      tm stop_3;
71      size_t stp3;
72
73
74      for (size_t i = 0; i < cars_cnt; i++)
75      {
76          std::shared_ptr<Car> car = q1.front();
77          set_time(&start_1, &str1);
78
79          car->create_carcass(i + 1);
80
81          set_time(&stop_1, &stp1);
82          q2.push(car);
83          q1.pop();
84
85          car = q2.front();
86          set_time(&start_2, &str2);
87          car->create_engine(i + 1);
88          set_time(&stop_2, &stp2);
89
90
91          q3.push(car);
92          q2.pop();
93
94          car = q3.front();
95          set_time(&start_3, &str3);
96          car->create_wheels(i + 1);
97          set_time(&stop_3, &stp3);
98          q3.pop();
99
100         print_my(i + 1, 1, str1, start_1);
101         print_my(i + 1, 1, stp1, stop_1);
102
103         print_my(i + 1, 2, str2, start_1);
104         print_my(i + 1, 2, stp2, stop_2);
105
106         print_my(i + 1, 3, str3, start_1);
107         print_my(i + 1, 3, stp3, stop_3);
108     }
109 }
110
111 void Conveyor::create_carcass()
112 {
113   size_t task_num = 0;
```

```
114
115   while (!this->q1.empty())
116   {
117     std::shared_ptr<Car> car = q1.front();
118     car->create_carcass(++task_num);
119
120     q2.push(car);
121     q1.pop();
122   }
123 }
124
125 void Conveyor::create_engine()
126 {
127   size_t task_num = 0;
128
129   do
130   {
131     if (!this->q2.empty())
132     {
133       std::shared_ptr<Car> car = q2.front();
134       car->create_engine(++task_num);
135
136       q3.push(car);
137       q2.pop();
138     }
139   } while(!q1.empty() || !q2.empty());
140 }
141
142 void Conveyor::create_wheels()
143 {
144   size_t task_num = 0;
145
146   do
147   {
148     if (!this->q3.empty())
149     {
150       std::shared_ptr<Car> car = q3.front();
151       car->create_wheels(++task_num);
152       q3.pop();
153     }
154   } while (!q1.empty() || !q2.empty() || !q3.empty());
155 }
```

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 (0 == num % i)
55             return;
```

```
56
57      this->is_freight = true;
58  }
59
60  //                                                          (a^x)
61  Engine::Engine(int a, int x)
62  {
63      this->engine_power = a;
64
65      for (int i = 0; i < x; i++)
66          this->engine_power *= a;
67  }
68
69  //                                                  (n-
                          )
70  Wheels::Wheels(int n)
71  {
72      size_t f1 = 1, f2 = 1;
73      this->wheels_cnt = f1;
74
75      for (int i = 2; i < n; i++)
76      {
77          this->wheels_cnt = f1 + f2;
78          f1 = f2;
79          f2 = this->wheels_cnt;
80      }
81  }
82
83  void Car::create_engine(size_t task_num)
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
91  void Car::create_carcass(size_t task_num)
92  {
93      this->carcass = std::unique_ptr<Carcass>(new Carcass(27644437));
94  }
95
96  void Car::create_wheels(size_t task_num)
97  {
98      this->wheels = std::unique_ptr<Wheels>(new Wheels(this->engine->
            engine_power));
99  }
100 }
```

Листинг 3.3: Реализация класса логирования

```cpp
#include <iostream>
#include <chrono>

using namespace std::chrono;

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

  static void set_time(tm *ti, size_t *ms)
}

void set_time(tm *ti, size_t *ms)
{
    system_clock::time_point now = system_clock::now();
    system_clock::duration cast = now.time_since_epoch();

    cast -= duration_cast<seconds>(cast);
    *ms = cast.count();

    time_t tt = system_clock::to_time_t(now);
    *ti = *gmtime(&tt);
}
```

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

На рисунке 4.1 приведна часть вывода программы.

```
Task №1 | Part1 | Start | 00:07:39.663534800
Task №1 | Part1 | Start | 00:07:39.663534800
Task №1 | Part2 | Start | 00:07:39.663534800
Task №1 | Part2 | Start | 00:07:39.663534800
Task №1 | Part3 | Start | 00:07:39.663534800
Task №1 | Part3 | Start | 00:07:39.663534800
Task №2 | Part1 | Start | 00:07:39.685896500
Task №2 | Part1 | Start | 00:07:39.685896500
Task №2 | Part2 | Start | 00:07:39.685896500
Task №2 | Part2 | Start | 00:07:39.685896500
Task №2 | Part3 | Start | 00:07:39.685896500
Task №2 | Part3 | Start | 00:07:39.685896500
Task №3 | Part1 | Start | 00:07:39.708836300
Task №3 | Part1 | Start | 00:07:39.708836300
Task №3 | Part2 | Start | 00:07:39.708836300
Task №3 | Part2 | Start | 00:07:39.709793400
Task №3 | Part3 | Start | 00:07:39.709793400
Task №3 | Part3 | Start | 00:07:39.709793400
Task №4 | Part1 | Start | 00:07:39.777612800
Task №4 | Part1 | Start | 00:07:39.777612800
Task №4 | Part2 | Start | 00:07:39.777612800
Task №4 | Part2 | Start | 00:07:39.778610800
Task №4 | Part3 | Start | 00:07:39.778610800
Task №4 | Part3 | Start | 00:07:39.778610800
Task №5 | Part1 | Start | 00:07:39.808700100
Task №5 | Part1 | Start | 00:07:39.808700100
Task №5 | Part2 | Start | 00:07:39.808700100
Task №5 | Part2 | Start | 00:07:39.808700100
Task №5 | Part3 | Start | 00:07:39.808700100
Task №5 | Part3 | Start | 00:07:39.808700100
Task №6 | Part1 | Start | 00:07:39.829646800
Task №6 | Part1 | Start | 00:07:39.829646800
Task №6 | Part2 | Start | 00:07:39.829646800
Task №6 | Part2 | Start | 00:07:39.829646800
Task №6 | Part3 | Start | 00:07:39.829646800
Task №6 | Part3 | Start | 00:07:39.829646800
```

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: Windows 11 64-bit.
- Оперативная память: 8 ГБ.
- Процессор: Intel(R) Core(TM) i5-8250 CPU @ 1.60 ГГц.

4.3 Время выполнения реализаций алгоритмов

Время выполнения алгоритма замерялось с помощью применения технологии профайлинга. Данный инструмент даёт детальное описание количество вызовов и количества времени CPU, затраченного на выполнение каждой функции. На вход функциям подаются обычные числа ввиду их простоты. 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

Вывод

В данном разделе приведено время исполнения 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.

Заключение

В рамках данной 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: 11.12.2021.

[2] Вычислительный конвейер, Режим доступа: https://ru.wikipedia.org/wiki/Вычислительный_конвейе
Дата обращения: 11.12.2021.