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»

Лабораторная работа №5
по дисциплине "Анализ Алгоритмов"

Тема Конвейерная обработка данных

Студент Рядинский К. В.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва

2021 г.

СОДЕРЖАНИЕ

Введение

Параллельные вычисления часто используются для увеличения скорости выполнения программ. Однако приемы, применяемые для однопоточных машин, для параллельных могут не подходить. Конвейерная обработка данных является популярным приемом при работе с параллельными машинами.

Целью данной работы является изучение и реализация метода конвейерных вычислений.

В рамках выполнения работы необходимо решить следующие задачи:

1. изучения основ конвейерной обработки данных;
2. применение изученных основ для реализации конвейерной обработки данных;
3. получения практических навыков;
4. получение статистики выполнения программы;
5. описание и обоснование полученных результатов;
6. выбор и обоснование языка программирования, для решения данной задачи.

1 Аналитическая часть

Конвейер - способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций), технология, используемая при разработке компьютеров и других цифровых электронных устройств.
Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд.

Для данной работы был выбран алгоритм сборки компьютера, состоящий из трех этапов: установка материнской платы (м.п.), установка центрального процессора (цп), установка графического процессора (гп).

1.1 Оценка производительности идеального конвейера

Пусть задана операция, выполнение которой разбито на n последовательных этапов. При последовательном их выполнении операция выполняется за время

$$\tau_e = \sum_{i=1}^{n} \tau_i \tag{1}$$

где

n – количество последовательных этапов;

τ_i – время выполнения i-го этапа;

Быстродействие одного процессора, выполняющего только эту операцию, составит

$$S_e = \frac{1}{\tau_e} = \frac{1}{\sum\limits_{i=1}^{n} \tau_i} \tag{2}$$

где

τ_e — время выполнения одной операции;

n — количество последовательных этапов;

τ_i — время выполнения i-го этапа;

Максимальное быстродействие процессора при полной загрузке конвейера составляет

Число n — количество уровней конвейера, или глубина перекрытия, так как каждый такт на конвейере параллельно выполняются n операций. Чем больше число уровней (станций), тем больший выигрыш в быстродействии может быть получен.

Известна оценка

$$\frac{n}{n/2} \leq \frac{S_{max}}{S_e} \leq n \tag{3}$$

где

S_{max} — максимальное быстродействие процессора при полной загрузке конвейера;

S_e — стандартное быстродействие процессора;

n — количество этапов.

то есть выигрыш в быстродействии получается от $\frac{n}{2}$ до n раз.

Реальный выигрыш в быстродействии оказывается всегда меньше, чем указанный выше, поскольку:

1) некоторые операции, например, над целыми, могут выполняться за меньшее количество этапов, чем другие арифметические операции. Тогда отдельные станции конвейера будут простаивать;
2) при выполнении некоторых операций на определённых этапах могут требоваться результаты более поздних, ещё не выполненных этапов предыдущих операций. Приходится приостанавливать конвейер;

3) поток команд(первая ступень) порождает недостаточное количество операций для полной загрузки конвейера.

1.2 Вывод

В данном разделе были рассмотрены основополагающие материалы, которые в дальнейшем потребуются при реализации конвейера.

В качестве входных данных программе будет подаваться количество компьютеров (целое положительное число). На выходе будет выдаваться лог работы программы в параллельном режиме и последовательном. Ограничением для работы программного продукта будет являться, что количество компьютеров строго положительное целое число.

Реализуемое программное обеспечение будет работать в экспериментальном режиме. В экспериментальном режиме будет проводиться сравнение временных характеристик последовательного и параллельного реализаций конвейерных вычислений.

Критерием, по которому данная реализиация будет сравниваться с другими реализациями, является время выполнения реализации алгоритма.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы работы конвейеров, используемые типы данных и структура программного обеспечения.

2.1 Схемы алгоритмов

На схеме 1 представлены этапы запуска конвейеров. На схеме 2 представлена работа одного из конвейеров.

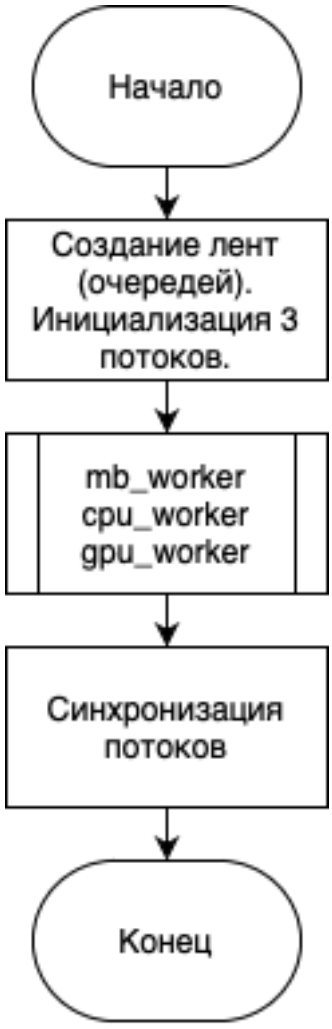

Рис. 1 — Схема запуска конвейеров

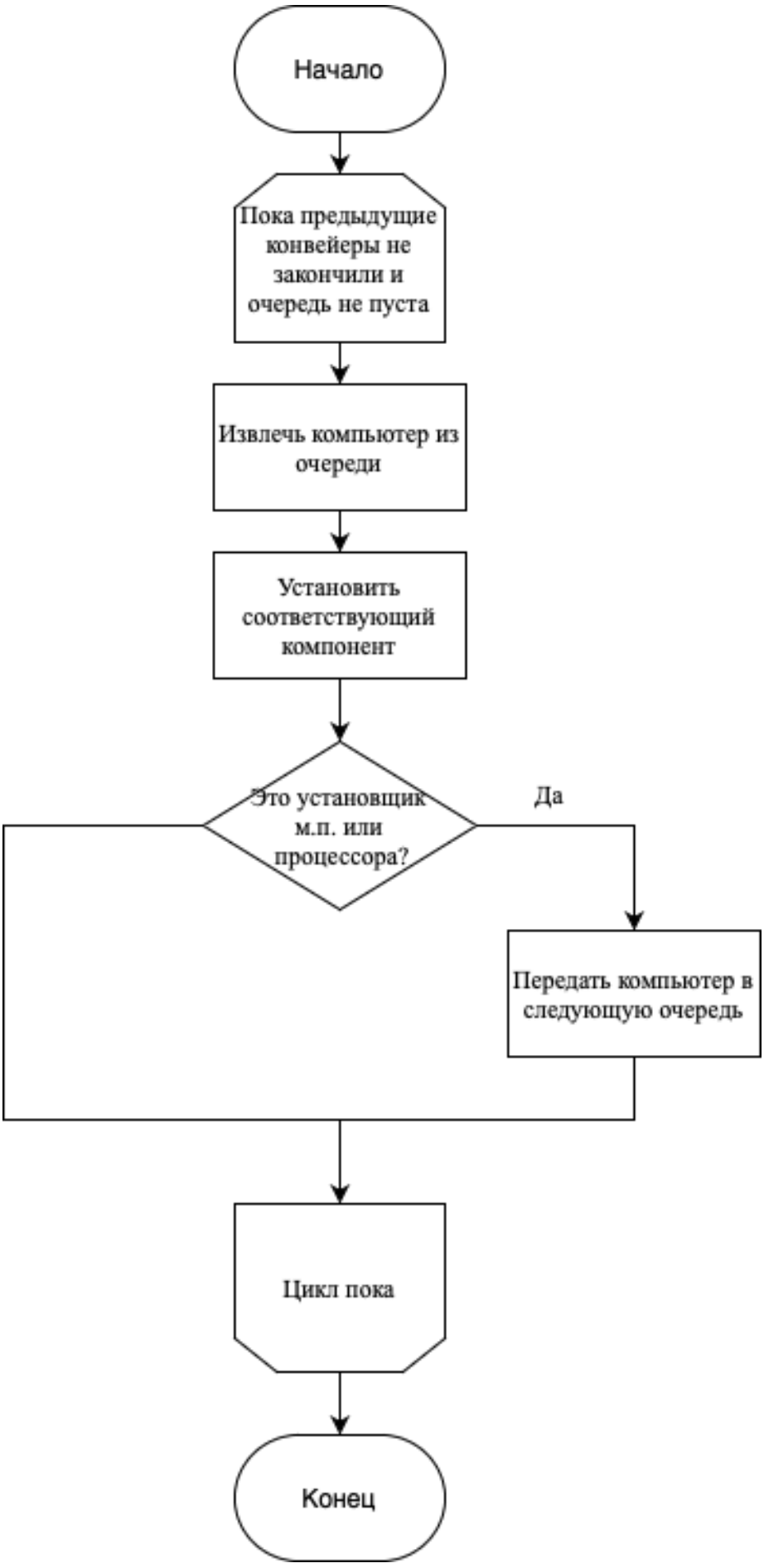

Рис. 2 — Схема работы одного конвейера

2.2 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

1. Конвейер — очередь.
2. Время установки компонента — целочисленный тип long long.
3. Номер компьютера — целочисленный тип long long.

2.3 Структура программного обеспечения

Программное обеспечение состоит из следующих модулей:

1. main.cpp — модуль, содержащий код точки входа.
2. queue.h — модуль, содержащий код реализации очереди.
3. dns.cpp — модуль, содержащий код конвейера.
4. computer.h — модуль, содержащий код класса компьютера.

2.4 Вывод

В данном разделе были рассмотрены схемы алгоритма, описаны используемые типы данных, структура программного обеспечения.

3 Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Средства реализации

К языку программирования выдвигаются следующие требования:

1. Возможность порождать системные потоки.
2. Возможность производить замер времени выполнения части программы.
3. Существуют среды разработки для этого языка.

По этим требованиям был выбран язык программирования C++.

3.2 Листинги кода

Листинг 1: Реализация очереди

```
#pragma once

#include <queue>
#include <mutex>
#include <condition_variable>

template <class T>
class SafeQueue
{
public:
    SafeQueue(void) : q(), m(), c() {}
    ~SafeQueue(void) {}

    bool empty() const {
        std::lock_guard<std::mutex> lock(m);
        return q.empty();
    }

    size_t size() const {
        std::lock_guard<std::mutex> lock(m);
        return q.size();
    }
```

```
23
24
25      void enqueue(T t)
26      {
27          std::lock_guard<std::mutex> lock(m);
28          q.push(t);
29          c.notify_one();
30      }
31
32      T dequeue(void)
33      {
34          std::unique_lock<std::mutex> lock(m);
35          while (q.empty()) {
36              c.wait(lock);
37          }
38
39          T val = q.front();
40          q.pop();
41          return val;
42      }
43  private:
44      std::queue<T> q;
45      mutable std::mutex m;
46      std::condition_variable c;
47  };
```

Листинг 2: Класс компьютера

```
1  #pragma once
2
3  #include <chrono>
4  #include <thread>
5
6
7  class Computer {
8  public:
9      Computer() = default;
10
11     bool is_built() const { return mother_board && cpu && gpu
       ; }
12
```

```
13      bool mb_installed() const { return mother_board; }
14      bool cpu_installed() const { return cpu; }
15      bool gpu_installed() const { return gpu; }
16
17      void set_id(size_t i) { id = i; }
18      size_t get_id() const { return id; }
19
20      void install_mb()  { std::this_thread::sleep_for(
        mb_installation_time);  mother_board = true; }
21      void install_cpu() { std::this_thread::sleep_for(
        cpu_installation_time); cpu = true; }
22      void install_gpu() { std::this_thread::sleep_for(
        gpu_installation_time); gpu = true; }
23
24
25      long long mb_install_time_s;
26      long long mb_install_time_e;
27      long long cpu_install_time_s;
28      long long cpu_install_time_e;
29      long long gpu_install_time_s;
30      long long gpu_install_time_e;
31
32  private:
33      bool mother_board;
34      bool cpu;
35      bool gpu;
36
37      size_t id;
38
39      const std::chrono::microseconds mb_installation_time
        {1500};
40      const std::chrono::microseconds cpu_installation_time
        {1000};
41      const std::chrono::microseconds gpu_installation_time
        {1100};
42  };
```

Листинг 3: Работа конвейеров

```
1  void dns::run_parallel() {
2      std::vector<Computer> comps(ncomps);
```

```
3
4       for (size_t i = 1; i <= comps.size(); i++) {
5           comps[i - 1].set_id(i);
6       }
7
8       SafeQueue<Computer*> mb_man;
9       SafeQueue<Computer*> cpu_man;
10      SafeQueue<Computer*> gpu_man;
11
12      std::atomic_bool mb_done  = false;
13      std::atomic_bool cpu_done = false;
14
15      auto qs = std::chrono::high_resolution_clock::now();
16
17
18      auto mb_worker = [qs, &mb_done, &m = mb_man, &c = cpu_man
    ]() {
19          while (!m.empty()) {
20              auto comp = m.dequeue();
21              auto s = std::chrono::high_resolution_clock::now
    ();
22              comp->install_mb();
23              auto e = std::chrono::high_resolution_clock::now
    ();
24
25              comp->mb_install_time_s = std::chrono::
    duration_cast<std::chrono::microseconds>(s - qs).count();
26              comp->mb_install_time_e = std::chrono::
    duration_cast<std::chrono::microseconds>(e - qs).count();
27
28              c.enqueue(comp);
29          }
30
31          mb_done = true;
32      };
33
34      auto cpu_worker = [qs, &cpu_done, &mb_done, &c = cpu_man,
     &g = gpu_man]() {
35          while (1) {
36              if (!c.empty()) {
```

```
37                  auto comp = c.dequeue();
38                  auto s = std::chrono::high_resolution_clock::
    now();
39                  comp->install_cpu();
40                  auto e = std::chrono::high_resolution_clock::
    now();
41
42                  comp->cpu_install_time_s = std::chrono::
    duration_cast<std::chrono::microseconds>(s - qs).count();
43                  comp->cpu_install_time_e = std::chrono::
    duration_cast<std::chrono::microseconds>(e - qs).count();
44
45                  g.enqueue(comp);
46              } else if (mb_done) {
47                  break;
48              }
49          }
50
51          cpu_done = true;
52      };
53
54      auto gpu_worker = [qs, &cpu_done, &mb_done, &g = gpu_man
    ]() {
55          while (1) {
56              if (!g.empty()) {
57                  auto comp = g.dequeue();
58                  auto s = std::chrono::high_resolution_clock::
    now();
59                  comp->install_gpu();
60                  auto e = std::chrono::high_resolution_clock::
    now();
61
62                  comp->gpu_install_time_s = std::chrono::
    duration_cast<std::chrono::microseconds>(s - qs).count();
63                  comp->gpu_install_time_e = std::chrono::
    duration_cast<std::chrono::microseconds>(e - qs).count();
64              } else if (cpu_done && mb_done){
65                  break;
66              }
67          }
```

```
68      };
69
70
71      for (auto &&c : comps) {
72          mb_man.enqueue(&c);
73      }
74
75      qs = std::chrono::high_resolution_clock::now();
76
77      auto mb_t  = std::thread(mb_worker);
78      auto cpu_t = std::thread(cpu_worker);
79      auto gpu_t = std::thread(gpu_worker);
80
81
82      mb_t.join();
83      cpu_t.join();
84      gpu_t.join();
85
86      auto qe = std::chrono::high_resolution_clock::now();
87
88      std::cout << "End time: " << std::chrono::duration_cast<
    std::chrono::microseconds>(qe - qs).count() << "\n";
89
90      print_comps(comps);
91  }
```

3.3 Вывод

В данном разделе был разработан конвейер.

4 Исследовательский раздел

В данном разделе будет проведен замер временных характеристик выполнения алгоритмов и пример работы программы.

4.1 Пример работы программы

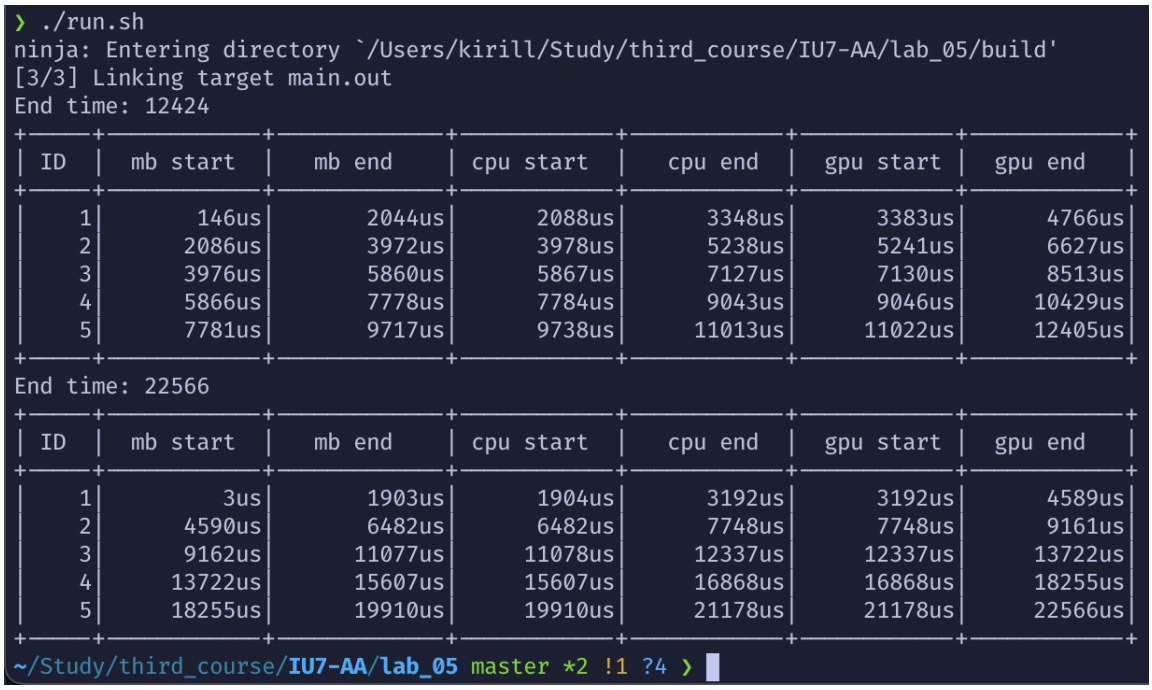

Рис. 3 — Пример работы программы

4.2 Технические характеристики

Технические характеристики электронно-вычислительной машины, на которой выполнялось тестирование:

— операционная система: macOS BigSur версия 11.4;

— оперативная память: 8 гигабайт LPDDR4;

— процессор: Apple M1.

4.3 Время работы программы

В таблице 1 представлен лог работы программы. Лента 1 занимается установкой материнской платы, лента 2 — установкой центрального процессора,

лента 3 — установка графического процессора. Время указано в микросекундах.

Таблица 1 — Лог работы конвейерной обработки программы

Лента №	Задача №	Время начала	Время конца
1	1	40	1951
2	1	1966	3221
3	1	3227	4403
1	2	1962	3865
2	2	3876	5132
3	2	5137	6516
1	3	3873	5781
2	3	5789	7044
3	3	7059	8439
1	4	5789	7669
2	4	7672	8927
3	4	8932	10314
1	5	7671	9554
2	5	9559	10816
3	5	10817	12200

На рисунке 4 представлен график зависимости времени работы реализации конвейеров от количества задач для линейной и параллельной обработки конвейера.

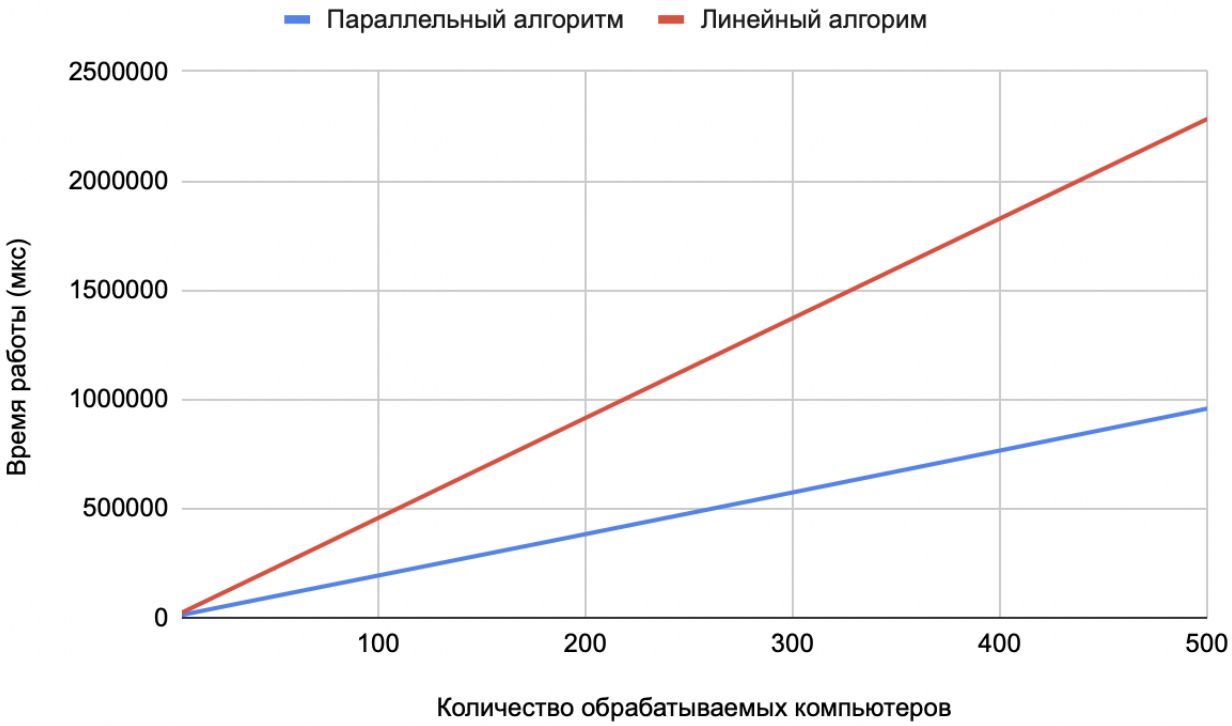

Рис. 4 — Зависимость времени работы реализации конвейеров от количества задач

4.4 Вывод

Параллельная реализация конвейерной обработки значительно выигрывает по времени относительно линейной реализации. Как видно из рисунка 4, линейная реализация примерно в 2 раза медленнее параллельной при 500 задачах.

Заключение

В результате выполнения лабораторной работы было экспериментально подтверждено различие временных характеристик последовательной и параллельной реализации конвейерной обработки данных.

В рамках выполнения работы была достигнута цель и решены следующие задачи:

1. изучили освоили конвейерную обработку данных;
2. применили изученные основы для реализации конвейерной обработки данных;
3. получили практические навыки;
4. получили статистику выполнения программы;
5. описали и обосновали полученные результаты;
6. выбрали и обосновали языка программирования, для решения данной задачи.

Список литературы

1. Visual Studio Code [Электронный ресурс], режим доступа: https://code.visualstudio.com/ (дата обращения: 30 ноября 2021 г.)
2. LPDDR4 [Электронный ресурс] https://ru.wikipedia.org/wiki/LPDDR#LPDDR4 (дата обращения: 30 ноября 2021 г.)
3. Ульянов М. В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и Анализ. - Наука Физматлит, 2007. - 376.
4. Библиотека Chrono [Электронный ресурс] Режим доступа: https://en.cppreference.com/w/cpp/chrono (дата обращения: 30 ноября 2021 г.).
5. Библиотека для работы с потоками thread [Электронный ресурс] Режим доступа: https://en.cppreference.com/w/cpp/thread (дата обращения: 30 ноября 2021 г.).