МГТУ им. Баумана

Лабораторная работа №5

По курсу: "Анализ алгоритмов"

Конвеерная обработка

Работу выполнил: Мокеев Даниил, ИУ7-54

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

Москва, 2019

Оглавление

Введение

Цель работы: изучение возможности конвеерной обработки и использование такого подхода на практике. Необходимо сравнить времени работы алгоритма на нескольких потоков и линейную реализацию.

В ходе лабораторной работы предстоит:

- Реализовать конвейер на потоках;
- Реализовать линейную обработку;
- Провести сравнение времени работы;

1 Аналитическая часть

Ковейер - система поточного производства. В терминах программирования ленты конвейера представленны 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 ассинхронном режиме. В качестве предметной области я решил выбрать торты - на первой линии конвейера замешивается тесто, на второй наносят глазурь, на третьей декорируют.

1.1 Параллельное программирование

При использовании многопроцессорных вычислительных систем с общей памятью обычно предполагается, что имеющиеся в составе системы процессоры обладают равной производительностью, являются равноправными при доступе к общей памяти, и время доступа к памяти является одинаковым (при одновременном доступе нескольких процессоров к одному и тому же элементу памяти очередность и синхронизация доступа обеспечивается на аппаратном уровне). Многопроцессорные системы подобного типа обычно именуются симметричными мультипроцессорами (symmetric multiprocessors, SMP).

Перечисленному выше набору предположений удовлетворяют также активно развиваемые в последнее время многоядерные процессоры, в которых каждое ядро представляет практически независимо функциони рующее вычислительное устройство.

Обычный подход при организации вычислений для многопроцессорных вычислительных систем с общей памятью – создание новых парал-

лельных методов на основе обычных последовательных программ, в которых или автоматически компилятором, или непосредственно программистом выделяются участки независимых друг от друга вычислений. Возможности автоматического анализа программ для порождения параллельных вычислений достаточно ограничены, и второй подход является преобладающим. При этом для разработки параллельных программ могут применяться как новые алгоритмические языки, ориентированные на параллельное программирование, так и уже имеющиеся языки, расширенные некоторым набором операторов для параллельных вычислений.

Широко используемый подход состоит и в применении тех или иных библиотек, обеспечивающих определенный программный интерфейс (application programming interface, API) для разработки параллельных программ. В рамках такого подхода наиболее известны Windows Thread API. Однако первый способ применим только для ОС семейства Microsoft Windows, а второй вариант API является достаточно трудоемким для использования и имеет низкоуровневый характер [3].

1.1.1 Организация взаимодействия параллельных потоков

Потоки исполняются в общем адресном пространстве параллельной программы. Как результат, взаимодействие параллельных потоков можно организовать через использование общих данных, являющихся доступными для всех потоков. Наиболее простая ситуация состоит в использовании общих данных только для чтения. В случае же, когда общие данные могут изменяться несколькими потоками, необходимы специальные усилия для организации правильного взаимодействия.

1.2 Вывод

Была рассмотрена конвейерная обработка данных, технология параллельного программирования и организация взаимодействия параллельных потоков.

2 | Конструкторская часть

Требования к вводу: На ввод подается целое число - желаемое колличество изготовленных экземпляров
Требования к программе:

- вывод статистики обаботанных экземпляров;
- при матрицах неправильных размеров программа не должна аварийно завершаться.

2.1 Схемы алгоритмов

В данной части будет рассмотрена схема алгоритма.

2.2 Распараллеливание программы

Распараллеливание программы должно ускорять время работы. Это достигается за счет перенесения каждой из лент конвейера на отдельный поток.

2.3 Вывод

В данном разделе была рассмотрена схема алгоритма и способ ее распараллеливания.

3 Технологическая часть

3.1 Выбор ЯП

Я выбрал в качестве языка программирования Golang, потому как он достаточно удобен, быстр и успешно использует концепции мультипоточного программирования.

Время работы алгоритмов было замерено с помощью функции Now() из библиотеки time.

3.2 Описание структуры ПО

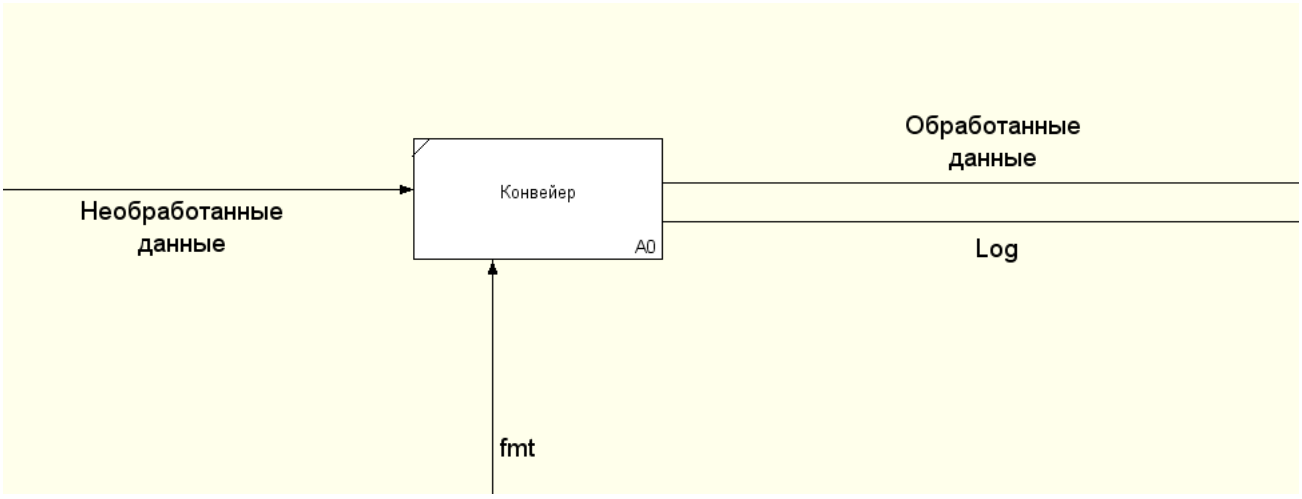

Рис. 3.1: Функциональная схема умножения матриц (IDEF0 диаграмма 1 уровня)

3.3 Сведения о модулях программы

Программа состоит из:

- lab05.go- главный файл программы, в котором располагается точка входа в программу.
- linear.go - файл, содержащий функцию линейной обработки данных.

3.4 Листинг кода алгоритмов

Листинг 3.1: Параллельный конвейер

```
1  func conv(amount int, wait chan int) *queue{
2    uno := make(chan *cake, 5)
3    dos := make(chan *cake, 5)
4    tres := make(chan *cake, 5)
5    line := new_queue(amount)
6    first := func(){
7      for{
8        select{
9        case a := <- uno:
10       a.dough = true
11
12       a.started_dough = time.Now()
13       took_dough := 200
14       time.Sleep(time.Duration(took_dough) * time.
             Millisecond)
15
16       a.finished_dough = time.Now()
17       dos <- a
18       }
19     }
20   }
21
22   second:= func(){
23   for{
24     select{
```

```
25        case a := <- dos:
26        //fmt.Printf("Cake num %d started topping\n", a.num)
27        a.topping = true
28
29        a.started_topping = time.Now()
30        took_topping := 200
31        time.Sleep(time.Duration(took_topping) * time.
              Millisecond)
32
33        a.finished_topping = time.Now()
34        tres <- a
35        }
36      }
37    }
38
39    third := func(){
40      for{
41        select{
42        case a := <- tres:
43        //fmt.Printf("Cake num %d started decor\n", a.num)
44        a.decor = true
45
46        a.started_decor = time.Now()
47        took_decor := 200
48        time.Sleep(time.Duration(took_decor) * time.
              Millisecond)
49
50        a.finished_decor = time.Now()
51        line.push(a)
52        if (a.num == amount){
53        wait <- 0 }
54
55        }
56      }
57    }
58
59    go first()
60    go second()
61    go third()
62    for i:=0; i<=amount; i++{
```

```
63      a := new(cake)
64      a.num = i
65      uno <- a
66    }
67    return line
68  }
```

Листинг 3.2: Линейный конвейер

```
1  func linear (amount int)*queue{
2    queue_for_topping := new_queue(amount)
3    queue_for_decor := new_queue(amount)
4    finished := new_queue(amount)
5    i:= 0
6    for ;i!=-1;{
7      a := new(cake)
8      a.num = i
9      first(a, queue_for_topping)
10     if queue_for_topping.last >= 0{
11       second(queue_for_topping.pop(), queue_for_decor)
12     }
13     if queue_for_decor.last >= 0{
14       third(queue_for_decor.pop(), finished)
15     }
16     if finished.waiting[len(finished.waiting)-1] != nil{
17       return finished}
18     i+=1
19   }
20   return finished
21 }
```

3.5 Вывод

В данном разделе была рассмотрена структура ПО и листинги кода программы.

4 | Исследовательская часть

Был проведен замер времени работы алгоритмов с использованием разного колличества изготавливаемых изделий. Исследования были проведены на процессоре Intel Core i5-6200U На каждом ковейре время производства занимало 0.2сек

4.1 Примеры работы

4.2 Постановка эксперемента

Проведем сравнение для каждого из алгоритмов. Для замера времени будем использовать функцию time.Now()

Сравним результаты для линейной обработки данных и распараллеленой:

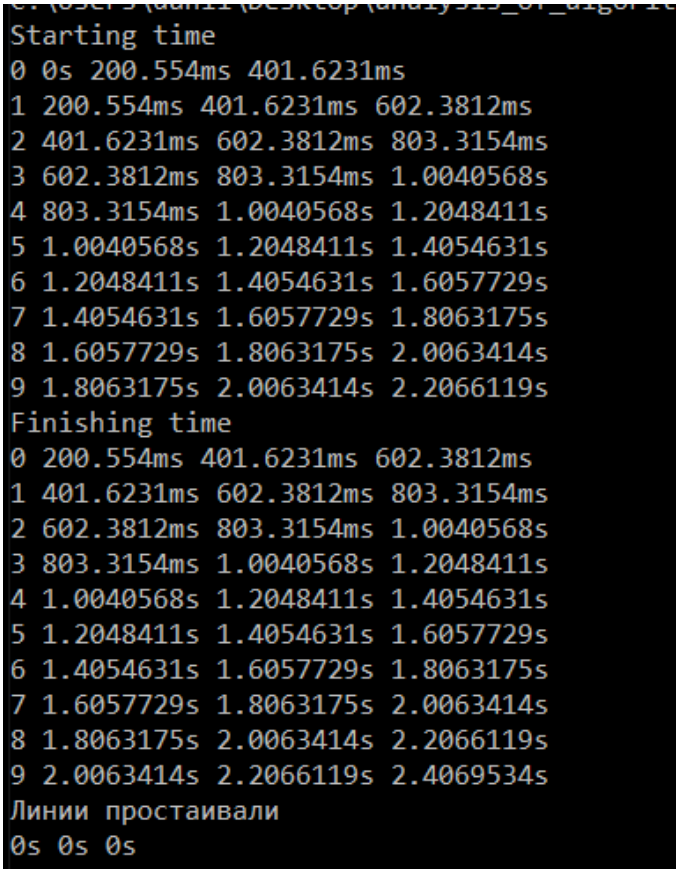

Рис. 4.1: Пример работы программы - конефейер на потоках

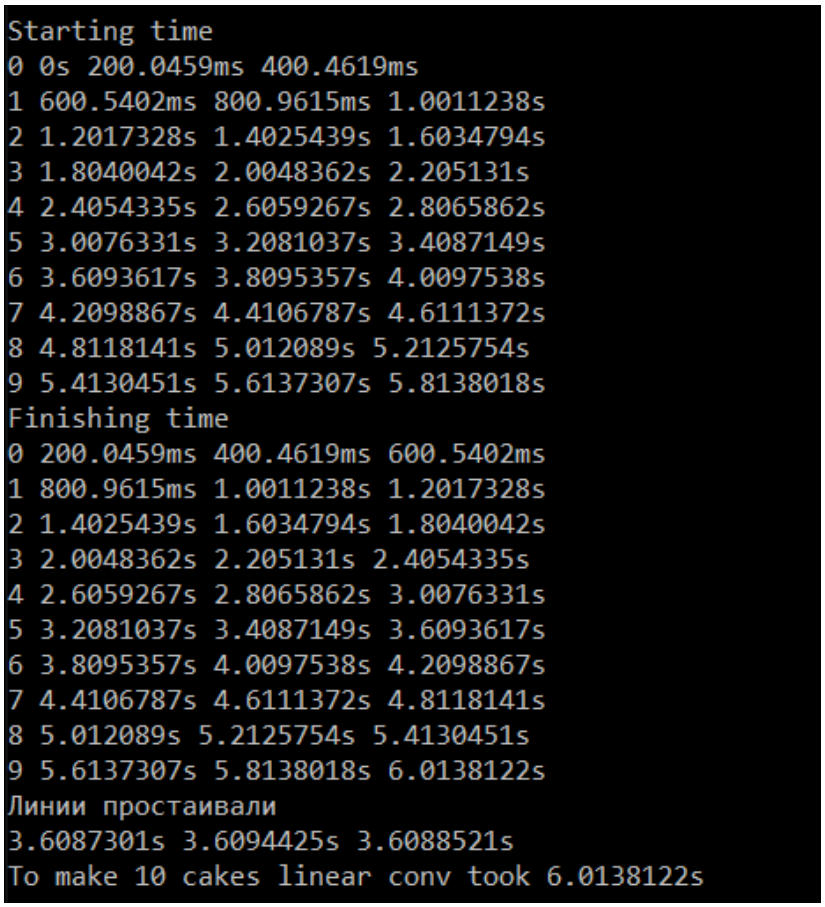

Рис. 4.2: Пример работы программы - линейная обработка данных и результат

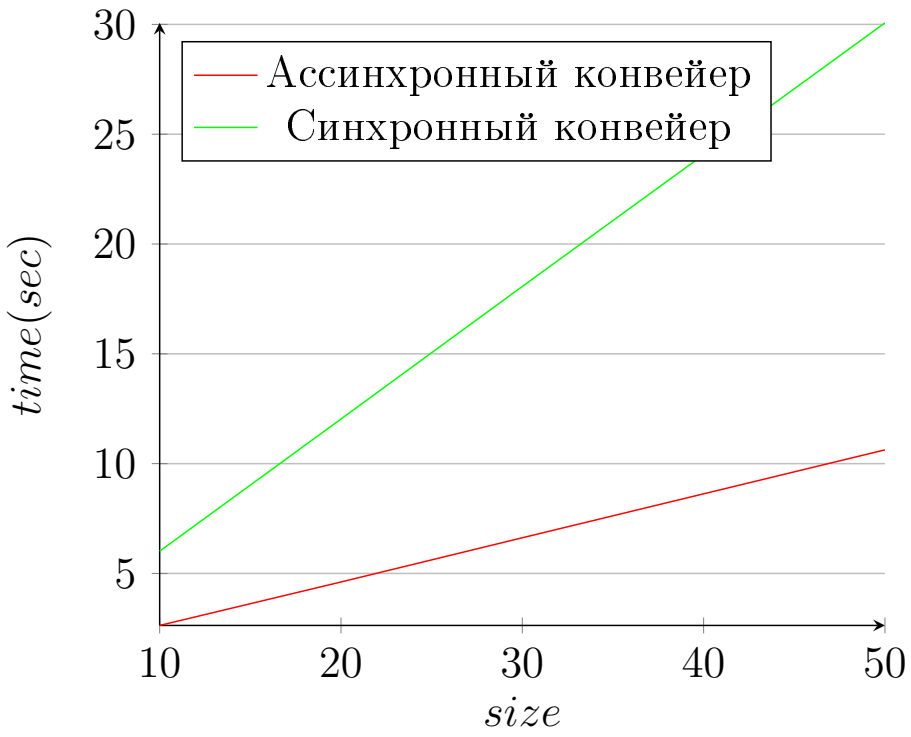

Рис. 4.3: Сравнение параллельного и обычного алгоритмов

4.2.1 Заключение эксперементальной части

Экперемент показывает, что использование нескольких потоков для реализации конвейерной обработки данных ускоряет алгоритм в несколько раз. При этом возникает ситуация при которой ленты не простаивают. Тратится лишь малое время для передачи данных на линию.

Заключение

В ходе лабораторной работы были изучены возможности параллельных вычислений, реализован алгоритм конвейерной обработки данных с помощью параллельных вычислений.

Было проведено сравнение синхронной версии того же алгоритма и ассинхронной. Выяснилось, что при использовании потоков, время работы алгоритма не просто сокращается, но и снижается скорость роста времени при увеличении числа изготавливаемых экземпляров.

Литература

[1] И. В. Белоусов(2006), Матрицы и определители, учебное пособие по линейной алгебре, с. 1 - 16

[2] Le Gall, F. (2012), "Faster algorithms for rectangular matrix multiplication Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012), pp. 514–523

[3] Константин Баркалов, Владимир Воеводин, Виктор Гергель. Intel Parallel Programming [Электронный ресурс], - режим доступа https://www.intuit.ru/studies/courses/4447/983/lecture/14925

[4] Руководство по языку C#[Электронный ресурс], - режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/