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»

ОТЧЕТ

По лабораторной работе №5

По курсу: «Анализ алгоритмов»

Тема: «Многопоточная реализация конвейера»

Студент: Ле Ни Куанг

Группа: ИУ7и-56Б

Преподаватель: Волкова Л. Л.

Строганов Ю. В.

Москва

2021

Оглавление

Введение

Параллельные вычисления - способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно (одновременно).

Конвейерная обработка улучшает использование аппаратных ресурсов для заданного набора процессов, каждый из которых применяет эти ресурсы заранее предусмотренным способом. Хорошим примером конвейерной организации является сборочный транспортер на производстве, на котором изделие последовательно проходит все стадии вплоть до готового продукта.

Целью работы: Реализация конвейера с использованием параллельных вычислений.

Задачи работы:

1. изучение основ конвейерной обработки данных;
2. получение практических навыков конвейерных вычислений;
3. экспериментальное подтверждение различий во временной эффективности реализаций на материале замеров процессорного времени выполнения;

1 Аналитический раздел

В данном разделе будет приведено описание конвейерной обработки и параллельных вычислений.

1.1 Конвейерная обработка

Конвейер - машина непрерывного транспорта, предназначенная для перемещения сыпучих, кусковых или штучных грузов.

Конвейеризация - это техника, в результате которой задача или команда разбивается на некоторое числоподзадач, которые выполняются последовательно. Каждая подкоманда выполняется на своем логическом устройстве. Все логические устройства (ступени) соединяются последовательно таким образом, что выход i-ой ступени связан свходом (i+1)-ой ступени,все ступени работают одновременно. Множество ступеней называется конвейером.Выигрыш вовремени достигается привыполнении нескольких задачзасчет параллельной работы ступеней, вовлекая на каждом такте новую задачу или команду.

1.2 Оценка производительности конвейера

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

$$\tau_e = \sum_{i=1}^{n} \tau_i \tag{1.1}$$

где

n - количество последовательных этапов;

τ_i - время выполнения i-го этапа;

Быстродействие одного процессора, выполняющего только эту операцию, составит

$$S_e = \frac{1}{\tau_e} = \frac{1}{\sum\limits_{i=1}^{n} \tau_i} \tag{1.2}$$

где

τ_e - время выполнения одной операции;

n - количество последовательных этапов;

τ_i - время выполнения i-го этапа;

Выберем время такта - величину $t_T = max\sum\limits_{i=1}^{n}(\tau_i)$ и потребуем при разбиении на этапы, чтобы для любого i = 1, ... , n выполнялось условие $(\tau_i + \tau_{i+1})modn = \tau_T$. То есть

чтобы никакие два последовательных этапа (включая конец и новое начало операции) не могли быть выполнены за время одного такта.

Максимальное быстродействие процессора при полной загрузке конвейера составляет

$$S_{max} = \frac{1}{\tau_T} \tag{1.3}$$

где

τ_T - выбранное нами время такта;

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

$$\frac{n}{n/2} \leq \frac{S_{max}}{S_e} \leq n \tag{1.4}$$

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 $n/2$ 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.

1.3 Многопоточность

К достоинствам многопоточной реализации той или иной системы перед многозадачной можно отнести следующее:

- Упрощение программы в некоторых случаях за счет использования общего адресного пространства.

- Меньшие относительно процесса временные затраты на создание потока.

К достоинствам многопоточной реализации той или иной системы перед однопоточной можно отнести следующее:

- Упрощение программы в некоторых случаях, за счет вынесения механизмов чередования выполнения различных слабо взаимосвязанных подзадач, требующих одновременного выполнения, в отдельную подсистему многопоточности.
- Повышение производительности процесса за счет распараллеливания процессорных вычислений и операций ввода-вывода.

Существует два виды параллелизма в алгоритмах и программах:

- Конечный параллелизм определяется информационной независимостью некоторых фрагментов в тексте программы.
- Массовый параллелизм определяется информационной независимостью итераций циклов программы.

1.4 Вывод

В данном разделе были приведено описание описание конвейерной обработки и параллельных вычислений.

2 Конструкторский раздел

В данном разделе будет приведено описание схемы конвейера.

2.1 Схема конвейера

Этот конвейер представляет собой простой конвейер обработки изображений. Идея была взята из конвейера обработки изображений машинного обучения (сверточной сети). На рисунке показана схема конвейера. В конвейере рабочие одной стадии делят входные и выходные каналы.

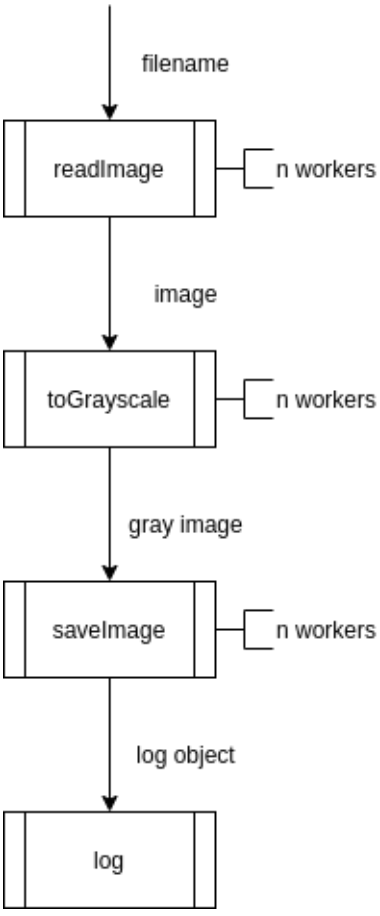

Рис. 2.1: Схема конвейера

2.2 Вывод

В данном разделе было приведено описание схем простого конвейера обработки изображений.

3 Технологический раздел

В данном разделе будет приведены требования к программу и листинг кода.

3.1 Требования к программному обеспечению

Программа должна принимать в качестве входных данных имя входного файла изображения PNG и имя выходного файла.

Результатом программы являются файлы PNG в оттенках серого с соответствующими именами.

3.2 Средства реализации

Язык программирования: Go

Редактор: VS Code

Go - это новый мощный язык программирования, в котором легко писать параллельные программы или конвейере (серия стадии, соединенных каналами).

3.3 Листинг кода

Листинг 3.1: Основная часть кода

```
// ...
const (
    NCores  = 4
    NTasks  = 10
    InFile  = "data/in.png"
    OutFile = "data/out.png"
)

type Task struct {
    id     int
    in     string
    out    string
    img    image.Image
    start  []time.Time
    finish []time.Time
}

type Stage func(in, out chan Task)

```

```
20 func createTask(id int, in, out string) Task {
21     return Task{id: id, in: in, out: out}
22 }
23
24 func genTask(n int, out chan Task) {
25     for i := 0; i < n; i++ {
26         out <- createTask(i, InFile, OutFile)
27     }
28     close(out)
29 }
30
31 func Stage1(in, out chan Task) {
32     for task := range in {
33         task.start = append(task.start, time.Now())
34         task.img = readImage(task.in)
35         task.finish = append(task.finish, time.Now())
36         out <- task
37     }
38 }
39
40 func Stage2(in, out chan Task) {
41     for task := range in {
42         task.start = append(task.start, time.Now())
43         task.img = toGrayscale(task.img)
44         task.finish = append(task.finish, time.Now())
45         out <- task
46     }
47 }
48
49 func Stage3(in, out chan Task) {
50     for task := range in {
51         task.start = append(task.start, time.Now())
52         saveImage(task.img, task.out)
53         task.finish = append(task.finish, time.Now())
54         out <- task
55     }
56 }
57
58 func StageLog(in, _ chan Task) {
59     for task := range in {
60         // ...
61     }
62 }
63
64 func mainPipeline(nTasks int, stages []Stage, nNodes []int) {
65     wg := &sync.WaitGroup{}
66     in := make(chan Task, nTasks)
67
```

```
68      genTask(nTasks, in)
69
70      for i := range stages {
71          out := make(chan Task, nTasks)
72          wg.Add(1)
73          if nNodes[i] > 1 {
74              go func(st Stage, in, out chan Task) {
75                  wg1 := &sync.WaitGroup{}
76                  for j := 0; j < nNodes[i]; j++ {
77                      wg1.Add(1)
78                      go func(st Stage, in, out chan Task) {
79                          defer wg1.Done()
80                          st(in, out)
81                      }(st, in, out)
82                  }
83                  defer wg.Done()
84                  defer close(out)
85                  wg1.Wait()
86              }(stages[i], in, out)
87          } else {
88              go func(st Stage, in, out chan Task) {
89                  defer wg.Done()
90                  defer close(out)
91                  st(in, out)
92              }(stages[i], in, out)
93          }
94          in = out
95      }
96
97      wg.Wait()
98  }
99
100 func readImage(path string) image.Image {
101     infile, err := os.Open(path)
102     if err != nil {
103         panic(err.Error())
104     }
105     defer infile.Close()
106
107     src, _, err := image.Decode(infile)
108     if err != nil {
109         panic(err.Error())
110     }
111     return src
112 }
113
114 func toGrayscale(src image.Image) image.Image {
115     bounds := src.Bounds()
```

```
116     w, h := bounds.Max.X, bounds.Max.Y
117     gray := image.NewGray(image.Rect(0, 0, w, h))
118     for x := 0; x < w; x++ {
119         for y := 0; y < h; y++ {
120             gray.Set(x, y, src.At(x, y))
121         }
122     }
123     return gray
124 }
125 
126 func saveImage(img image.Image, path string) {
127     outfile, err := os.Create(path)
128     if err != nil {
129         panic(err.Error())
130     }
131     defer outfile.Close()
132     png.Encode(outfile, img)
133 }
134 
135 func serialWorker(in, out chan Task) {
136     for t := range in {
137         t.start = append(t.start, time.Now())
138         t.img = readImage(t.in)
139         t.finish = append(t.finish, time.Now())
140 
141         t.start = append(t.start, time.Now())
142         t.img = toGrayscale(t.img)
143         t.finish = append(t.finish, time.Now())
144 
145         t.start = append(t.start, time.Now())
146         saveImage(t.img, t.out)
147         t.finish = append(t.finish, time.Now())
148 
149         out <- t
150     }
151 }
152 
153 func mainSerial(nTasks int) {
154     wg := &sync.WaitGroup{}
155     in := make(chan Task, nTasks)
156     out := make(chan Task, nTasks)
157 
158     genTask(nTasks, in)
159 
160     // go StageLog(out, out)
161 
162     for i := 0; i < NCores; i++ {
163         wg.Add(1)
```

```
164             go func(in, out chan Task) {
165                 serialWorker(in, out)
166                 defer wg.Done()
167             }(in, out)
168         }
169
170         defer close(out)
171         wg.Wait()
172 }
173
174 // ...
175
176 func main() {
177         stages := []Stage{Stage1, Stage2, Stage3, StageLog}
178         nNodes := []int{4, 4, 4, 1}
179         mainPipeline(NTasks, stages, nNodes)
180 }
```

3.4 Вывод

В этом разделе было рассмотрено требования к программу и кода программы.

4 Экспериментальный раздел

В данном разделе будет приведено пример работы программы, результаты тестирования и сравнение времени работы программы.

4.1 Примеры работы

На рисунке 4.1 приведен пример работы программы.

```
→ go run main.go
Id      Stage1 start    Duration        Stage2 start    Duration        Stage3 start    Duration
0       01:56:38.075    32.16ms         01:56:38.107    50.62ms         01:56:38.158    64.45ms
1       01:56:38.107    40.77ms         01:56:38.158    41.53ms         01:56:38.222    90.55ms
2       01:56:38.148    31.81ms         01:56:38.199    43.83ms         01:56:38.313    78.85ms
3       01:56:38.179    31.06ms         01:56:38.243    47.34ms         01:56:38.392    79.7ms
4       01:56:38.21     30.99ms         01:56:38.29     45.13ms         01:56:38.471    63.62ms
5       01:56:38.241    39.08ms         01:56:38.336    43.88ms         01:56:38.535    73.91ms
6       01:56:38.281    33.81ms         01:56:38.379    51.93ms         01:56:38.609    71.37ms
7       01:56:38.314    30.86ms         01:56:38.431    38.01ms         01:56:38.68     66.12ms
8       01:56:38.345    43.28ms         01:56:38.469    38.63ms         01:56:38.746    69.65ms
9       01:56:38.389    34.88ms         01:56:38.508    46.93ms         01:56:38.816    69.03ms
Total time (with log): 810.557989ms
```

Рис. 4.1: Примеры работы программы

4.2 Результат тестирования

На рисунке 4.2 приведен результат тестирования на изображении 1576x890 писель. (Тестирование в этом случае - это сравнение исходного изображения и результирующего изображения.)

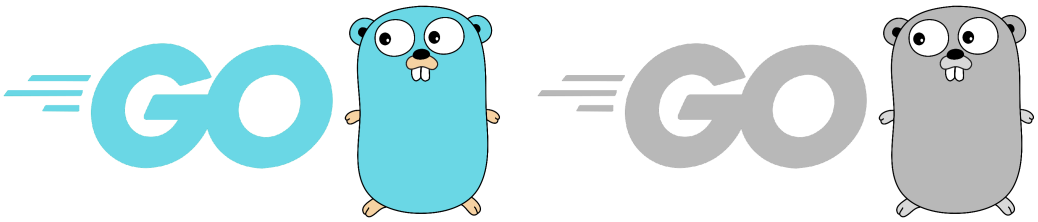

Рис. 4.2: Результат тестирования

4.3 Сравнение времени работы

Операционная система - Ubuntu 20.04.1 LTS

Процессор - Intel® CoreTM i5-7300HQ CPU @ 2.50GHz × 4 (ЦП 4 ядра 4 потока)

В таблице 4.1 приведены замеры времени работы последовательной параллельной реализации (работает на 3 ядрах), конвейера (3 стадии, каждый запускается в одном потоке, работают на 3 ядрах) и параллельного конвейера (3 стадии по три рабочих в каждом, работающих на 3 ядра). Чтобы быть справедливым, тесты будет использовать до 3-х ядер.

Кол-во задач	**Последо.**	**Конвейер**	**Паралл. кон.**
10	480	883	446
20	856	1701	845
30	1218	2549	1255
40	1689	3364	1679
50	2064	4326	2046
60	2432	5006	2469
70	2865	5872	2883
80	3245	6613	3282
90	3595	7463	3672
100	4049	8275	3999

Таблица 4.1: Времени работы (мс)

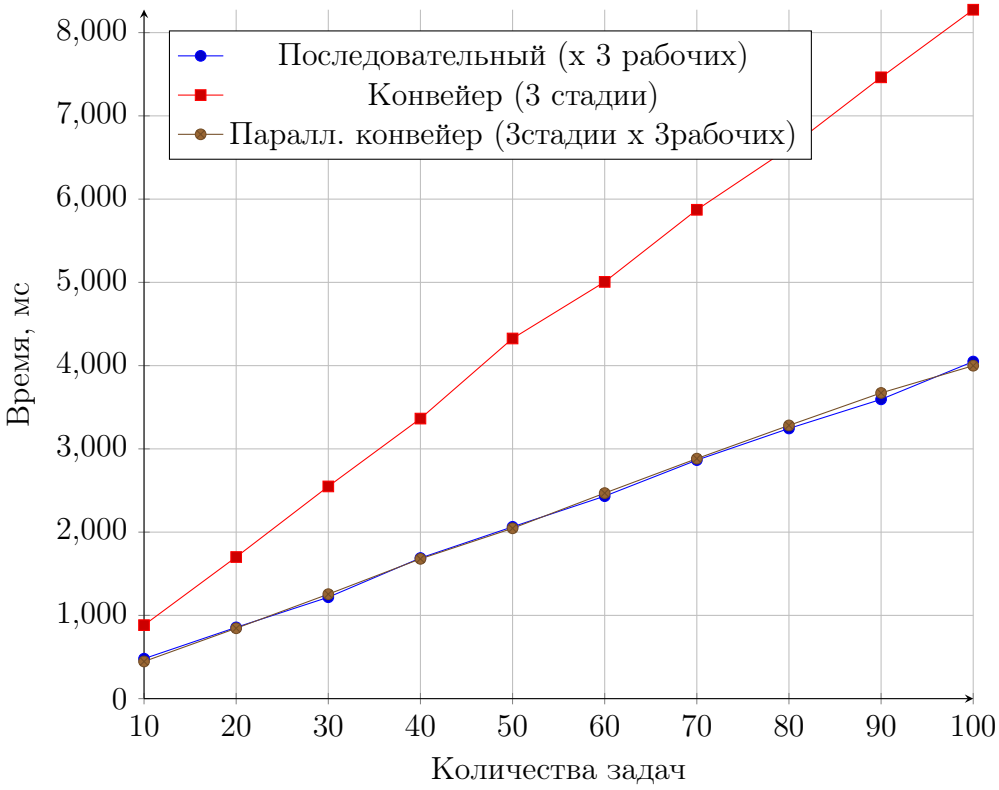

Рис. 4.3: Зависимость времени работы последовательного реализации, конвейера и параллельного конвейера от количества задач

4.4 Вывод

График показывает, что параллельный конвейер в 2 раз быстрее, чем конвейер, который использует один поток для каждого этапа (все два используют 3 ядра процессора). Причина в том, что последний этап требует больше времени для обработки, поэтому предыдущие этапы должны ждать его. Поскольку конвейер недостаточно длинный и для его реализации требуются дополнительные каналы и передача данных, поэтому мы не видим здесь разницы между параллельным конвейером и параллельно-последовательной реализацией (возможно также ОС не равномерно разделила рабочих конвейера на ядра процессора).

Заключение

В ходе лабораторной работы было изучено основы конвейерной обработки, реализованна конвейера с использованием параллельных вычислений. Было сравнить временные характеристики параллельная безконвейерная версия, конвейер и параллельный конвейер и сделаны следующие выводы:

- параллельный конвейер в 2 раз быстрее, чем конвейер, который использует один поток для каждого этапа;
- конвейер недостаточно длинный и для его реализации требуются дополнительные каналы и передача данных, поэтому не видим здесь разницы между параллельным конвейером и параллельно-безконвейерной реализацией.

Литература

[1] Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. — СПб: БХВ-Петербург, 2002. — 608 с.

[2] Конвейерные вычисления
`https://studylib.ru/doc/4736512/konvejernye-vychisleniya` [Электронный ресурс] (дата обращения: 25.12.20)

[3] Effective Go
`https://golang.org/doc/effective_go.html` [Электронный ресурс] (дата обращения: 25.12.20)