МГТУ им. Баумана

Лабораторная работа №5

По курсу: "Анализ алгоритмов"

Конвейер

Работу выполнил: Ковалев Дмитрий ИУ7-53Б

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

Москва, 2020

Оглавление

Введение

Цель работы: создать систему конвейерной обработки.

Задачи данной лабораторной работы:

1. спроектировать ПО, реализующего конвейерную обработку;
2. описать реализацию ПО;
3. провести тестирование ПО.

1 | Аналитическая часть

В данной части будут рассмотрены главные принципы конвейерной обработки и параллельных вычислений.

1.1 Общие сведения о конвейерной обработке

Конвейер – машина непрерывного транспорта [1], предназначенная для перемещения сыпучих, кусковых или штучных грузов.
Конвейерное производство - система поточной организации производства на основе конвейера, при которой оно разделено на простейшие короткие операции, а перемещение деталей осуществляется автоматически. Это такая организация выполнения операций над объектами, при которой весь процесс воздействия разделяется на последовательность стадий с целью повышения производительности путём одновременного независимого выполнения операций над несколькими объектами, проходящими различные стадии. Конвейером также называют средство продвижения объектов между стадиями при такой организации[2]. Появилось в 1914 году на производстве Модели-Т на заводе Генри Форда[3] и произвело революцию сначала в автомобилестроении, а потом и во всей промышленности.

1.2 Параллельное программирование

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

новременно).

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 параллельных 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 [8].

1.2.1 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.

1.3 Вывод

В данном разделе были рассмотрены основы конвейерной обработки, технология параллельного программирования и организация взаимодействия параллельных потоков.

2 | Конструкторская часть

Требования к программе: В этом задании реализован аналог unix pipeline

Когда STDOUT одной программы передаётся как STDIN в другую программу

Но в нашем случае эти роли выполняют каналы, которые мы передаём из одной функции в другую.

Само задание по сути состоит из двух частей
* Написание функции ExecutePipeline которая обеспечивает нам конвейерную обработку функций-воркеров, которые что-то делают.
* Написание нескольких функций, которые считают нам какую-то условную хеш-сумму от входных данных

Расчет хеш-суммы реализован следующей цепочкой:
* SingleHash считает значение crc32(data)+"-"+crc32(md5(data)) (конкатенация двух строк через), где data - то что пришло на вход (по сути - числа из первой функции)
* MultiHash считает значение crc32(th+data)) (конкатенация цифры, приведённой к строке и строки), где th=0..5 (т.е. 6 хешей на каждое входящее значение), потом берёт конкатенацию результатов в порядке расчета (0..5), где data - то что пришло на вход (и ушло на выход из SingleHash)
* CombineResults получает все результаты, сортирует (https://golang.org/pkg/sort/), объединяет отсортированный результат через _ (символ подчеркивания) в одну строку
* crc32 считается через функцию DataSignerCrc32
* md5 считается через DataSignerMd5

В чем сложность:
* DataSignerMd5 может одновременно вызываться только 1 раз, счита-

ется 10 мс. Если одновременно запустится несколько - будет перегрев на 1 сек * DataSignerCrc32, считается 1 сек
* На все расчеты у нас 3 сек.
* Если делать в лоб, линейно - для 7 элементов это займёт почти 57 секунд, следовательно надо это как-то распараллелить

2.1 Вывод

В данном разделе была рассмотрена схема организации конвейерной обработки.

3 | Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:
• на вход подаётся количество задач (функции для хэширования);
• на выходе — время, затраченное на обработку заявок;
• в процессе обработки задач необходимо контролировать перегревы.

3.2 Сведения о модулях программы

Программа состоит из:

- signer.go - главный файл программы, в котором располагается вся логика конвеера
- common.go - реализация хэш функций
- main_test.go - тесты конвеера
- extra_test.go - дополнительные тесты конвеера

3.3 Листинг кода алгоритмов

Листинг 3.1: Работа конвеера

```
package main

import (
  "fmt"
  "sort"
  "strconv"
  "strings"
  "sync"
)

const TH_SIZE = 6
const DEBUG_INFO = false


var Md5Defender sync.Mutex

func CalcMd5(data string) string {
  Md5Defender.Lock()
  result := DataSignerMd5(data)
  Md5Defender.Unlock()
  return result
}

func CalcSingleHash(out chan string, str string) {
  out <- DataSignerCrc32(str)
}

func AsyncSingleHash(out chan interface{}, str, strMd5
    string, SHash *sync.WaitGroup) {
  defer SHash.Done()

  ChanCrc32, ChanCrc32Md5 := make(chan string, 1), make(
      chan string, 1)
  go CalcSingleHash(ChanCrc32, str)
  go CalcSingleHash(ChanCrc32Md5, strMd5)

  strCrc32, strCrc32Md5 := <-ChanCrc32, <-ChanCrc32Md5
  out <- strCrc32 + "~" + strCrc32Md5
  if DEBUG_INFO {
```

```
38      fmt.Println(str, " SingleHash data ", str)
39      fmt.Println(str, " SingleHash md5(data) ", strMd5)
40      fmt.Println(str, " SingleHash crc32(md5(data)) ",
            strCrc32Md5)
41      fmt.Println(str, " SingleHash strCrc32 ", strCrc32)
42      fmt.Println(str, " SingleHash result ", strCrc32+"~"+
            strCrc32Md5)
43    }
44
45  }
46
47  func SingleHash(in, out chan interface{}) {
48    SHash := &sync.WaitGroup{}
49    for data := range in {
50      dataToStr := fmt.Sprintf("%v", data)
51      strMd5 := CalcMd5(dataToStr)
52      SHash.Add(1)
53      go AsyncSingleHash(out, dataToStr, strMd5, SHash)
54    }
55    SHash.Wait()
56  }
57
58  func CalcMultiHash(Arr *[]string, idx int, str string, wg *
        sync.WaitGroup) {
59    defer wg.Done()
60    (*Arr)[idx] = DataSignerCrc32(strconv.Itoa(idx) + str)
61  }
62
63  func AsyncMultiHash(out chan interface{}, data interface{},
        MHash *sync.WaitGroup) {
64    defer MHash.Done()
65
66    str := fmt.Sprintf("%v", data)
67    MHashInside := &sync.WaitGroup{}
68    ArrFinalResult := make([]string, TH_SIZE)
69    for TH := 0; TH < TH_SIZE; TH++ {
70      MHashInside.Add(1)
71      go CalcMultiHash(&ArrFinalResult, TH, str, MHashInside)
72    }
73    MHashInside.Wait()
```

```
74    out <- strings.Join(ArrFinalResult, "")
75    if DEBUG_INFO {
76      for TH := 0; TH < TH_SIZE; TH++ {
77        fmt.Println(str, " MultiHash result: crc32(th+step1)
              ", TH, ArrFinalResult[TH])
78      }
79      fmt.Println(str, " MultiHash result:", strings.Join(
            ArrFinalResult, ""))
80    }
81  }
82
83  func MultiHash(in, out chan interface{}) {
84    MHash := &sync.WaitGroup{}
85    for data := range in {
86      MHash.Add(1)
87      go AsyncMultiHash(out, data, MHash)
88    }
89    MHash.Wait()
90  }
91
92  func CombineResults(in, out chan interface{}) {
93    var result []string
94    for el := range in {
95      result = append(result, el.(string))
96    }
97    sort.Strings(result)
98    strResult := strings.Join(result, "_")
99    out <- strResult
100   if DEBUG_INFO {
101     fmt.Println("CombineResults", strResult)
102   }
103 }
104
105 func ExecutePipeline(jobs ...job) {
106   wg := &sync.WaitGroup{}
107   in := make(chan interface{})
108   close(in)
109   for _, jobF := range jobs {
110     wg.Add(1)
111     out := make(chan interface{})
```

```
112     go func(in, out chan interface{}, wg *sync.WaitGroup,
            jobFunc job) {
113       defer wg.Done()
114       defer close(out)
115       jobFunc(in, out)
116     }(in, out, wg, jobF)
117     in = out
118   }
119   wg.Wait()
120 }
```

3.4 Вывод

В данном разделе были рассмотрены основные сведения о модулях программы, листинг кода.

4 | Исследовательская часть

4.1 Сравнительный анализ на основе замеров времени

Последовательное исполнение конвеера - 55 сек
Параллельное исполнение конвеера - 2.2 сек

4.2 Тестирование

Результаты, которые выводятся если отправить 2 значения (закомментировано в тесте):

```
    0 SingleHash data 0
0 SingleHash md5(data) cfcd208495d565ef66e7dff9f98764da
0 SingleHash crc32(md5(data)) 502633748
0 SingleHash crc32(data) 4108050209
0 SingleHash result 4108050209 502633748
4108050209 502633748 MultiHash: crc32(th+step1)) 0 2956866606
4108050209 502633748 MultiHash: crc32(th+step1)) 1 803518384
4108050209 502633748 MultiHash: crc32(th+step1)) 2 1425683795
4108050209 502633748 MultiHash: crc32(th+step1)) 3 3407918797
4108050209 502633748 MultiHash: crc32(th+step1)) 4 2730963093
4108050209 502633748 MultiHash: crc32(th+step1)) 5 1025356555
4108050209 502633748 MultiHash result: 29568666068035183841425683795340791879727309630931

1 SingleHash data 1
1 SingleHash md5(data) c4ca4238a0b923820dcc509a6f75849b
1 SingleHash crc32(md5(data)) 709660146
```

1 SingleHash crc32(data) 2212294583
1 SingleHash result 2212294583 709660146
2212294583 709660146 MultiHash: crc32(th+step1)) 0 495804419
2212294583 709660146 MultiHash: crc32(th+step1)) 1 2186797981
2212294583 709660146 MultiHash: crc32(th+step1)) 2 4182335870
2212294583 709660146 MultiHash: crc32(th+step1)) 3 1720967904
2212294583 709660146 MultiHash: crc32(th+step1)) 4 259286200
2212294583 709660146 MultiHash: crc32(th+step1)) 5 2427381542
2212294583 709660146 MultiHash result: 4958044192186797981418233587017209679042592542

CombineResults 29568666068035183841425683795340791879727309630931025356555
_4958044192186797981418233587017209679042592862002427381542

4.3 Вывод

Асинхронные конвейерные вычисления позволяют организовать непрерывную обработку данных, что позволяет выиграть время в задачах, где требуется обработка больших объемов данных за малый промежуток времени.

Заключение

В ходе выполнения работы была достигнута цель выполнены все поставленные задачи:

• рассмотреть и изучить асинхронную конвейерную обработку данных;

• реализовать систему конвейерных вычислений с количеством линий не меньше трех;

• на основании проделанной работы сделать выводы.

Асинхронные 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] Меднов В.П., Бондаренко Е.П. Транспортные, распределительные и рабочие конвейеры. М., 1970.

[2] Конвейерное производство[Электронный ресурс] - режим доступа https:/ё/dic.academic.ru/dic.nsf/ruwiki/1526795

[3] Конвейерный метод производства Генри Форда[Электронный ресурс] - режим доступа https://popecon.ru/305-konveiernyi-metod-proizvodstva-genri-forda.html

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

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

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

[7] 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

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

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