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 Конвейерная обработка

Студент Саркисов А.С.

Группа ИУ7-53Б

Оценка (баллы) _____

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

Москва — 2020 г.

Оглавление

Введение **2**

1 Аналитическая часть **3**

1.1 Общие сведения о конвейерной обработке 3

1.2 Параллельное программирование 3

1.2.1 Организация взаимодействия параллельных потоков 4

1.3 Вывод 5

2 Конструкторская часть **6**

2.1 Организация обработки 6

2.2 Вывод 7

3 Технологическая часть **8**

3.1 Выбор ЯП 8

3.2 Листинг кода алгоритмов 8

3.3 Вывод 11

4 Исследовательская часть **12**

4.1 Пример работы программы 12

4.2 Тестирование 13

4.3 Вывод 13

Заключение **14**

Список литературы **14**

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, предназначенная для перемещения сыпучих, кусковых или штучных грузов.[1]

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 и произвело революцию сначала в автомобилестроении, а потом и во всей промышленности.[3]

1.2 Параллельное программирование

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

При использовании многопроцессорных вычислительных систем с общей памятью обычно предполагается, что имеющиеся в составе системы процессоры обладают равной произ-

водительностью, 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 функционирующее 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.[4]

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 вводу: Количество конвейеров должно быть больше 0.

Требования к программе при параллельной обработке:

- Объекты должны последовательно проходить конвейеры в заданном подядке;
- конвейеры должны работать каждый в своем потоке;
- до завершения работы конвейер должен ожидать поступления новых элементов.

2.1 Организация обработки

На рисунке 2.1 представлена схема организации конвейерных вычислений.

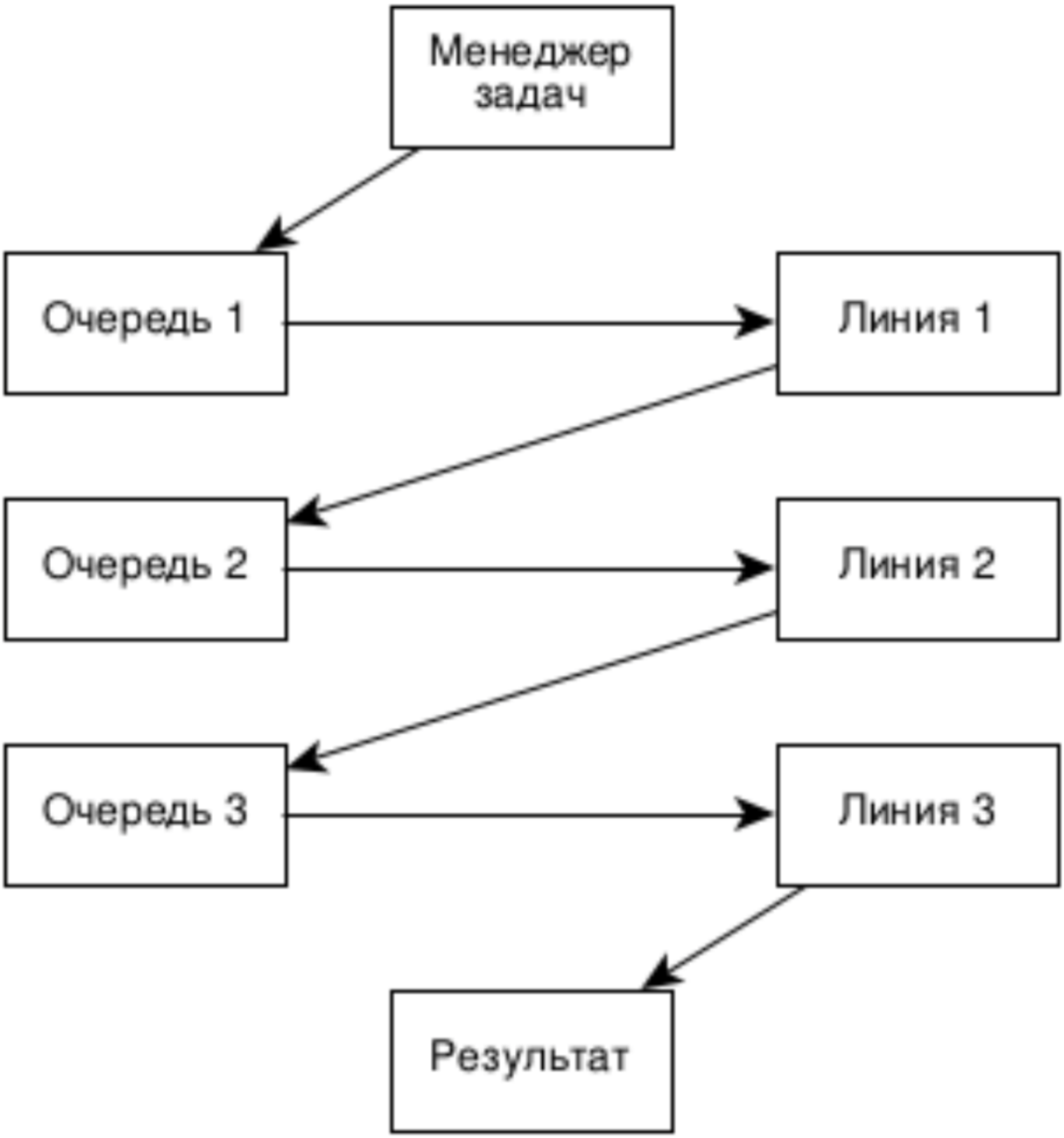

Рис. 2.1: Схема организации конвейерных вычислений

2.2 Вывод

В данном разделе была рассмотрена схема организации конвейерной обработки.

3 | Технологическая часть

Замеры времени были произведены на: Intel(R) Core(TM) i5 с тактовой частотой 1.4GHz, 4 ядра, 8 логических процессоров.

3.1 Выбор ЯП

В качестве языка программирования был выбран Golang так как этот язык поддерживает управление потоками на уровне ОС(незеленые потоки).[5] Средой разработки Visual Studio Code. Время работы алгоритмов было замерено с помощью класса Time. Многопоточное программирование было реализовано с помощью встроенной функции языка Golang "goroutine".

3.2 Листинг кода алгоритмов

В листинге 3.1 представлен модуль программы, в котором происходит формирование конвейеров с последующем запуском.

Листинг 3.1: Модуль создания конвейеров

```
1 func host(philos []*Philo, seatsNum int, eatTimes int) {
2   defer eatWG.Done()
3   eating := make(chan *Philo, 2)
4
5   for i := 0; i < seatsNum; i++ {
6     eatWG.Add(1)
7     go philos[i].eat(eatTimes, eating)
8   }
9 }
```

В листинге 3.2 представлен модуль, в котором задаются операции для каждого конвейера.

Листинг 3.2: Модуль операций каждого конвейера.

```
1  func (philo *Philo) eat(eatTimes int, eating chan *Philo) {
2    defer eatWG.Done()
3    sliceCS := make([]*ChopStick, 2)
4    sliceCS[0] = philo.leftCS
5    sliceCS[1] = philo.rightCS
6    randI1, randI2 := genTwoRandInd() // Random index of first and second chop
         sticks to pick
7
8    for i := 0; i < eatTimes; i++ {
9      select {
10     case eating <- philo:
11       //fmt.Printf("Host allows philosopher with index - %d to eat\n", philo
             .id)
12       //randomPause(3)
13       sliceCS[randI1].Lock()
14       sliceCS[randI2].Lock()
15
16       fmt.Printf("Starting to eat philospher with index - %d\n", philo.id)
17       randomPause(3)
18       fmt.Printf("Finishing to eat philospher with index - %d\n", philo.id)
19       <-eating
20
21       sliceCS[randI2].Unlock()
22       sliceCS[randI1].Unlock()
23     default:
24       //fmt.Printf("Host denied philosopher with index - %d to eat\n", philo
             .id)
25       //randomPause(3)
26       i--
27     }
28   }
```

```
}
```

В листинге 3.2 представлен модуль входа в программу.

Листинг 3.3: Модуль входа в программу

```
func main() {
  seatsNum, err := strconv.Atoi(os.Args[1]) // Number of philosophers and
     chop sticks as well
  if err != nil {
    return
  }
  eatTimes, err := strconv.Atoi(os.Args[2]) // Number of times every
     philosopher eats
  if err != nil {
    return
  }
  // Initialize chop sticks
  chopSticks := make([]*ChopStick, seatsNum)
  for i := 0; i < seatsNum; i++ {
    chopSticks[i] = new(ChopStick)
  }
  // Initialize chop philosophers
  philos := make([]*Philo, seatsNum)
  for i := 0; i < seatsNum; i++ {
    philos[i] = &Philo{i + 1, chopSticks[i], chopSticks[(i+1)%seatsNum]}
  }

  eatWG.Add(1)
  go host(philos, seatsNum, eatTimes)
  eatWG.Wait()
}
```

3.3 Вывод

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

4 | Исследовательская часть

4.1 Пример работы программы

На рисунке 4.1 представлена работа программы (лог операций конвейера):

```
(base) temasarkisov@MacBook-Pro-Artem src % go run philo.go 5 2
Starting to eat philospher with index - 5
Starting to eat philospher with index - 2
Finishing to eat philospher with index - 2
Starting to eat philospher with index - 2
Finishing to eat philospher with index - 5
Starting to eat philospher with index - 4
Finishing to eat philospher with index - 2
Starting to eat philospher with index - 1
Finishing to eat philospher with index - 4
Finishing to eat philospher with index - 1
Starting to eat philospher with index - 5
Finishing to eat philospher with index - 5
Starting to eat philospher with index - 4
Finishing to eat philospher with index - 4
Starting to eat philospher with index - 1
Starting to eat philospher with index - 3
Finishing to eat philospher with index - 3
Starting to eat philospher with index - 3
Finishing to eat philospher with index - 1
Finishing to eat philospher with index - 3
```

Рис. 4.1: Лог работы конвейерной обработки

На графиках видно, что конвейерная обработка с параллельными потоками в 2.5 раза быстрее чем такая же линейная с захватом переменных. Если же убрать ненужные захваты переменных (т.к. все действия происходят линейно, не может быть одновременного обращения к одной области памяти), то это будет работать в 2.5 раза быстрее, чем параллельный.

4.2 Тестирование

Тестирование произовдилось с помощью встроенной системы тестирования языка Golang "Benchmark".

На рисунке 4.2 представлены результаты работы бенчмарков:

```
❯ go test -bench=.
goos: darwin
goarch: amd64
BenchmarkConveyor1-8                8584            130761 ns/op
BenchmarkConveyor5-8                1855            647156 ns/op
BenchmarkConveyor10-8                932           1290879 ns/op
BenchmarkConveyor20-8                463           2585464 ns/op
BenchmarkConveyor100-8                90          12926973 ns/op
PASS
```

Рис. 4.2: Лог работы конвейерной обработки

4.3 Вывод

По результатам исследования, ассинхронные конвейерные вычисления позволяют сэкономить время в задачах, где необходима обработка большого объема данных за малый промежуток времени. Конвейерную обработку нет смысла применять для задач, занимающих мало времени, т.к. в этом случее большая часть времени потратится на ожидание доступа к переменной, дополнительных проверок. Тестирование показало, что конвейерная обработка реализована правильно.

Заключение

В ходе лабораторной работы я изучил возможности применения параллельных вычислений и конвейерной обработки и использовал такой подход на практике.

Конвейерная обработка позволяет сильно ускорить программу, если требуется обработать набор из однотипных данных, причем алгоритм обработки должен быть разбиваем на стадии. Однако от конвейерной обработки не будет смысла, если одна из стадий намного более трудоемкая, чем остальные, так как производительность всей программы будет упираться в производительность этой самой стадии, и разницы между обычной обработкой и конвейерной не будет, только добавятся накладные вычисления, связанные с диспетчеризацией потоков. В таком случае можно либо разбить трудоемкую стадию на набор менее трудоемких, либо выбрать другой алгоритм, либо отказаться от конвейерной обработки.

Литература

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

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

[5] Руководство по языку Golang[Электронный ресурс], - режим доступа: https://golang.org