Технічні науки

УДК 004.051

Бєлєвцов Віталій Вікторович
студент
Харківського національного університету радіоелектроніки
Белевцов Виталий Викторович
студент
Харьковского национального университета радиоэлектроники
Bielievtsov Vitalii
Student of the
Kharkiv National University of Radio Electronics

Шмельов Олег Борисович
студент
Харківського національного університету радіоелектроніки
Шмелёв Олег Борисович
студент
Харьковского национального университета радиоэлектроники
Shmelev Oleh
Student of the
Kharkiv National University of Radio Electronics

Науковий керівник:
Олійник Олена Володимирівна
старший викладач кафедри ПІ
Харківський національний університет радіоелектроніки

РЕАЛІЗАЦІЯ ТА ВИКОРИСТАННЯ БАГАТОЯДЕРНИХ ПРОЦЕСІВ З ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ GO

РЕАЛИЗАЦИЯ И ИСПОЛЬЗОВАНИЕ МНОГОЯДЕРНЫХ ПРОЦЕССОРОВ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА ПРОГРАММИРОВАНИЯ GO
USEGA AND IMPLEMENTATION OF PARALELISM IN GO

Анотація. *Досліджено методи та реалізацію паралельного програмування у Go.*

Ключові слова: *конкурентність, багатоядерність, метод, функція, ефективність, потік.*

Аннотация. *Исследованы методы и реализация параллельного программирования в GO.*

Ключевые слова: *конкурентность, многоядерность, метод, функция, эффективность, поток.*

Summary. *Implementation of parallelism in Go.*

Key words: *concurrency, multithreading, method, function, efficiency, thread.*

З розвитком технологій розробки процесорів персональних комп'ютерів та підвищенням запиту на швидкі розрахунки зростає потреба у оптимізації процесів комп'ютерних обчислень. Задля зниження ціни підтримки програмного забезпечення та обробки одиниці інформації, замовники програмних систем справедливо вимагають від розробників, щоб найбільший можливий об'єм даних було оброблено за найменший проміжок часу. Враховуючи наявні у стейкхолдерів величезні обсяги даних про користувачів, які підлягають, наприклад, аналізу, лише ефективних послідовних алгоритмів недостатньо для пришвидшення процесів обробки. Багато алгоритмів базується на послідовному виконанні певних дій над списком сутностей. Оскільки дії, виконувані над кожним

елементом оброблюваної послідовності скоріш за все не змінюватимуться незалежно від номеру ітерації, такі алгоритми можна пришвидшити у декілька разів, використовуючи ресурси багатоядерних процесорів та засоби роботи з ними.

Однією з технологій, що дозволяє виконувати багатопоточні обчислення є мова програмування, що розробляється компанією Google з 2007 го та презентована 2010 го року - Go (або Golang). Мова є достатньо низькорівневою, що дозволяє керувати та взаємодіяти, наприклад, з потоками, але є набагато більш простою для вивчення та використання, аніж будь-які інші популярні на сьогоднішній день низькорівневі мови програмування (наприклад, C++). Також, до переваг Golang можна віднести такі аспекти як:

1. Статична типізація;
2. Збирач сміття - інструмент, що позбавляє необхідності розробника в управлінні пам'яттю та ресурсами, які використовує програмне забезпечення вручну;
3. Власні бінарні файли;
4. Швидка та ефективна компіляція;
5. Проста робота з багатопоточністю.

Основним поняттям, що розглядатиметься є конкурентність. Конкурентність - це можливість різних елементів програми, алгоритмів або задач виконуватися невпорядковано, або в частковому порядку не впливаючи при цьому на вихідний результат. Це значним чином впливає на ефективність програмного забезпечення та швидкість виконання об'ємних обчислень. Основними інструментами роботи з конкурентністю у мові програмування Golang є Goroutnies - легко виконувані методи, або функції, що виконуються незалежно від викликаючого їх методу, або функції. Ефективність Goroutines обумовлена, здебільш, завдяки їх поведінці, яка полягає у плануванні обробки заданої кількості системних

потоків, що, теоретично, дозволяє обробити будь-яку кількість Goroutines. Варто зазначити, що однією з переваг використання ґорутин у мові програмування Go є мінімізація зусиль з боку розробника для написання коду. Оскільки для того, щоб змусити метод або функцію виконуватися у ґорутині треба використати ключове слово go перед викликом методу. Це дозволить виконувати їх незалежно від викликаючого потоку, та один від одного конкуруючи між потоками ядер процесору.

Наступним базовим поняттям для роботи з конкурентністю у Go є канали - спосіб комунікації між окремеми ґорутинами. Канали є умовною "магістраллю" для зв'язку та синхронізації виконуваних ґорутин. За допомогою каналів зникає необхідність у створенні проміжкових умовних змінних, які б контролювали та перевіряли кожну з виконуваних Goroutines на предмет закінчення виконання певного етапу методу або ж виконання його цілком. Канали, як і будь-який контейнер мають бути попередньо створені перед використанням таким чином: make(chan int). Поданий код створює канал для передачі цілих чисел. Для обробки "подій" створених шляхом передачі даних до каналів з ґорутин можна скористатися оператором select, який блокує виконання ґорутини до моменту спрацьовування хоча б однієї з описаних ситуацій. У якості прикладу наведемо метод отримання n-ного числа Фібоначчі:

```
package main
import "fmt"
func fibonacci(c, quit chan int) {
	x, y := 0, 1
	for {
		select {
		case c <- x:
			x, y = y, x+y
```

```
            case <-quit:
                fmt.Println("quit")
                return
            }
        }
}

func main() {
        c := make(chan int)
        quit := make(chan int)
        go func() {
            for i := 0; i < 10; i++ {
                fmt.Println(<-c)
            }
            quit <- 0
        }()
        fibonacci(c, quit)
}
```

Метод отримання числа Фібоначчі викликається 10 разів у циклі for, який виконується у ґорутині. Після кожної ітерації з результуючого каналу c, який передається у якості параметра до методу розрахунку числа, отримується чергове значення за допомогою оператору <-, а по завершенню виконання ґорутини до каналу quit передається значення 0, що ініціює друк слова "quit" та завершує виконання методу fibonacci. У методі fibonacci використовується конструкція select { case }, що очікує на спрацювання однієї з двох подій:

1. Внесення значення до каналу c;
2. Отримання значення з каналу quit.

Щойно отримання значення з каналу quit стає можливим, виконання ґорутини завершується, на екран виводиться слово "quit". Якщо ж спрацьовує подія внесення значення до каналу c, то виконується розрахунок наступних двох чисел послідовності Фібоначчі та продовжується виконання нескінченного циклу. Таким чином, розробник позбавлений необхідності створювати проміжкові змінні, для перевірки готовності виконання ґорутини. Як результат - алгоритм виконується набагато більш ефективно, ніж аналогічний даному послідовний.

Для демонстрації ефективності використання конкурентних методів над послідовними було розроблено послідовний та конкурентний алгоритми підрахунку суми чисел від 1 до заданого ліміту (у якості прикладу було підраховано суму від 1 до 1000000000), випробувано їх та виміряно час виконання. Результати дослідження та коди алгоритмів наведено нижче.

Код послідовного підрахунку суми:

```
func SerialSum() int {
  sum := 0
  for i := 0; i < limit; i++ {
     sum += i
  }
  return sum
}
```

Код конкурентного підрахунку суми:

```
func ConcurrentSum() int {
  n := runtime.GOMAXPROCS(0)

  res := make(chan int)

  for i := 0; i < n; i++ {
     go func(i int, r chan<- int){
        sum := 0
        start := (limit / n) * i
```

```
            end := start + (limit / n)
            if i == n - 1 {
                end = limit
            }


            for j := start; j < end; j += 1{
                sum += j
            }
            r <- sum
        }(i, res)
    }
    sum := 0

    for i := 0; i < n; i++ {
        sum += <-res
    }
    return sum
}
```

В обох наведених прикладах значення константи limit задано рівним 1000000000. Після запуску програми, виконуються заміри часу виконання конкурентного методу та послідовного. Результати наведено на рисунку 1.

```
Результат підрахунку суми конкурентним методом:  4999999995000000000
Витрачено часу:  156.0072ms
Результат підрахунку суми послідовним методом:  4999999995000000000
Витрачено часу:  448.3107ms
```

Рис. 1. Результати виконання програми

Результат наявно відображає ефективність використання всіх доступних ядер процесору для виконання масивних обчисленнь, пришвидшивши виконання програми у приблизно 2.8 разів порівняно з послідовним виконанням.

Таким чином, мова програмування Golang є досить перспективним інструментом для розробки масивних систем, засобами якого можна пришвидшити виконання коду програмного забезпечення у декілька разів, правильно конвертувавши послідовні алгоритми до конкурентних.

Література

1. Head First Go - O'Reilly: Jay McGavren, 2019. 560 с.
2. Go documentation. 2020. URL: https://golang.org/doc/