UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Miloš Mitrović

KONKURENTNO PROGRAMIRANJE U PROGRAMSKOM JEZIKU GO

master rad

Beograd, 2017.

Mentor:

dr Milena Vujošević Janičić
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Ana Anić
Univerzitet u Beogradu, Matematički fakultet

dr Laza Lazić
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Mojoj sestri Ivoni

Naslov master rada: Konkurentno programiranje u programskom jeziku Go

Rezime: text

Ključne reči: programski jezik Go, konkurentno programiranje

Sadržaj

1 Uvod **1**

2 Karakteristike programskog jezika Go **2**
2.1 Tipovi podataka 2
2.2 Funkcije i metodi 3
2.3 Interfejsi 3
2.4 Refleksija 3
2.5 Testiranje 3
2.6 Paketi 3

3 Konkurentno programiranje u programskom jeziku Go **4**
3.1 Osnovni pojmovi konkurentnog programiranja 4
3.2 Go 4

4 Primeri i poređenje sa drugim jezicima **5**
4.1 Osnovni primeri sa poređenjem 5
4.2 Quicksort 6
4.3 Množenje matrica 10
4.4 Eratostenovo sito 10
4.5 Složeniji primeri u programskom jeziku Go 10

5 Zaključak **11**

Literatura **12**

Glava 1

Uvod

Glava 2

Karakteristike programskog jezika Go

Go je imperativni programski jezik otvorenog koda koji razvija kompanija Google od 2007. godine. Napravljen je kao kompilirani jezik opšte namene sa statičkim tipovima koji podseća na interpretirane jezike sa dinamičkim tipovima. Podržava konkurentno programiranje, automatsko upravljanje memorijom kao i refleksiju tokom izvršavanja programa. Pogodan je za rešavanje svih vrsta problema a najviše se koristi za izgradnju servera, skriptove i samostalne aplikacije za komandnu liniju a može se korititi i za grafičke i mobilne aplikacije.

2.1 Tipovi podataka

Go je statički tipiziran jezik što znači da se varijabli dodeljuje tip prilikom njene deklaracije i on se ne može menjati tokom izvršavanja programa. Za razliku od C-a Go ne podržava automatsku konverziju tipova već se konverzija mora navesti eksplicitno, u suprotnom se prijavljuje greška.

Od osnovnih ugrađenih tipova podataka Go podržava:

- Numeričke - celobrojne označene (`int8`, `int16`, `int32`, `int64`) i neoznačene (`uint8`, `uint16`, `uint32`, `uint64`), u pokretnom zarezu (`float32`, `float64`) i kompleksne (`complex64`, `complex128`)
- Bulovske (`bool`)

- Tekstualne (`string`)

Od drugih vrsta podataka, Go podržava nizove, mape i slajseve - nizove sa promenljivom dužinom. Postoje i paketi koji omogućavaju rad sa listama.

2.2 Funkcije i metodi

2.3 Interfejsi

2.4 Refleksija

2.5 Testiranje

2.6 Paketi

Glava 3

Konkurentno programiranje u programskom jeziku Go

3.1 Osnovni pojmovi konkurentnog programiranja

3.2 Go

Go-rutine

Kanali

Sinhronizacija

Data race i data race detektor

Select naredba

Glava 4

Primeri i poređenje sa drugim jezicima

U ovom poglavlju su izloženi primeri konkurentnih Go programa kao i poređenje implementacija nekoliko jednostavnih algoritama u jezicima C, C++ i Python. Primeri ilustruju kako se može koristiti konkurentnost u jeziku Go i kakva je njegova efikasnost i udobnost programiranja u odnosu na druge pomenute jezike.

4.1 Osnovni primeri sa poređenjem

Prvo sledi kratak pregled programskih jezika C, C++ i Python i njihove konkurentnosti a zatim poređenje implementacija algoritama quicksort, množenje matrica i Eratostenovo sito. Kao kriterijumi poređenja se koriste brzina izvršavanja, maksimalna upotreba memorije, broj linija kôda kao i kvalitet dostupnih implementacija i subjektivna razumljivost kôda. Programi su testirani na hardveru sa procesorom Intel Core i3-3210 sa dva jezgra/četiri niti pod Linux-om Ubuntu 17.04. Implementacije u drugim jezicima su preuzete sa interneta i biće kratko prodiskutovane bez detaljnog obrazlaganja kôda. Poređenje je čisto ilustrativnog tipa jer zavisi od kvaliteta i efikasnosti dostupnih implementacija i ne treba ga uzeti kao definitivne procene efikasnosti samih jezika.

C

C++

Python

4.2 Quicksort

Opis algoritma

Go

Implementacija 4.1 koristi koncept višestrukog semafora za ograničavanje broja aktivnih go-rutina. Semafor je realizovan pomoću kanala sa baferom i select naredbe gde kapacitet kanala označava maksimalan broj aktivnih go-rutina. Program pokreće po jednu go-rutinu za svaki rekurzivni poziv, odnosno za levi i desni podniz dokle god je to moguće. U select naredbi se pokušava „dobijanje tokena" odnosno slanje poruke kroz kanal sa baferom. Kanal je definisan nad tipom prazne strukture jer nam nije bitna sama poruka već samo trenutno zauzeće kanala. U slučaju da možemo da dobijemo token odnosno uspešno pošaljemo praznu strukturu kroz kanal pokrećemo go-rutinu za rekurzivni poziv, u suprotnom ukoliko nema slodbnog mesta u baferu, rekurzivni poziv se izvršava sekvencijalno. Na kraju svake go-rutine je potrebno pročitati poruku iz kanala odnosno osloboditi jedno mesto. Da bismo bili sigurni da su sve go-rutine završile sa svojim radom, za sinhronizaciju koristimo wait grupe. Svaki konkurentni poziv funkcije kreira svoju wait grupu kojoj postavlja brojač na dva, a zatim, na kraju, čeka da oba rekurzivna poziva završe sa radom. S obzirom da se ne zna kada će moći da se izvrši konkurentni a kada sekvencijalni poziv funkcije, potrebno je u oba slučaja signalizirati wait grupi da je završeno sa radom.

C

Za razliku od Go implementacije, ovde[1] je upotrebljena dubina rekurzije za ograničavanje broja niti koje program kreira. Kada se dostigne zadata dubina rekurzije program više ne kreira nove niti već prelazi u sekvencijalni režim rada. Svaki konkurentni poziv funkcije kreira po dve nove niti ukoliko maksimalna dubina nije dostignuta, nakon čega se join funkcijom čeka na njihov završetak sa radom.

Listing 4.1: Go implementacija konkurentne quicksort funkcije

```
var semaphore = make(chan struct{}, 100)

func QuickSortConcurrent(a *[]int, low, hi int) {
    if hi < low {
        return
    }

    p := partition(a, low, hi)

    wg := sync.WaitGroup{}
    wg.Add(2)

    select{
    case semaphore <- struct{}{}:
        go func(){
            QuickSortConcurrent(a, low, p-1)
            <- semaphore
            wg.Done()
        }()
    default:
        QuickSortSequential(a,low, p-1)
        wg.Done()
    }

    select{
    case semaphore <- struct{}{}:
        go func(){
            QuickSortConcurrent(a, p+1, hi)
            <- semaphore
            wg.Done()
        }()
    default:
        QuickSortSequential(a, p+1, hi)
        wg.Done()
    }

    wg.Wait()
}
```

C++

Za C++ su razmatrane dve implementacije: prva[2], u kojoj je niz reprezentovan strukturom vector i koristi standardnu biblioteku, i druga[4], koja koristi običan niz int-ova i OpenMP biblioteku. Prva implementacija ima koncept dubine rekurzije za restrikciju broja niti na isti način kao što je realizovano u C-u. Druga, u kojoj je upotrebljena OpenMP biblioteka, ima mogućnost da postavi maksimalni broj aktivnih niti u jednom trenutku i za razliku od ostalih implementacija, kreira nit samo za jedan rekurzivni poziv dok se drugi izvršava sekvencijalno. U ovom slučaju nije potrebno imati dve funkcije, konkurentnu i sekvencijalnu već se konkurentno izvršavanje funkcije postiže instrukcijama same biblioteke.

Python

Ovde se takođe razmatraju dve implementacije koje koriste različite pakete za realizaciju konkurentnosti. Prva implementacija[3] koristi multiprocessing paket u kojoj se postavlja maksimalni broj aktivnih niti. Algoritam je realizovan tako što svaka nit sortira jedan deo niza a drugi stavlja u red da bude dostupan nitima koje još nisu krenule sa radom. U trenutku kada su sve niti aktivne, prestaju sa deljenjem niza i sekvencijalno sortiraju ostatak. U drugoj verziji[5] je iskorišćen Parallel Python paket sa već pomenutim konceptom dubine rekurzije za kontrolu broja niti i izvršnim serverom.

Rezultati

Nakon testiranja navedenih implementacija nad slučajno generisanim nizom od milion brojeva dobijeni su rezultati koji se mogu videti u tabeli 4.1. Kod konkurentnog izvršavanja C radi najbrže, dok Go ne zaostaje značajno za njim. U C++-u, verzija sa standardnom bibliotekom dobija najveće ubrzanje od svih implementacija ali ipak radi sporije od C-a i Go-a, dok verzija sa omp bibliotekom ne dobija nikakavo ubrzanje u odnosu na sekvencijalnu. Kada je reč o upotrebi memorije, Go i C++ koriste približno istu količinu kao i sekvencijane verzije, dok je C-u potrebno čak 4 puta više memorije. Što se tiče Pythona, očekivano, potrebno mu je značajno više vremena i veća količina memorija od svih ostalih. Interesantno je da je konkurentna verzija sa multiprocessing paketom sporija od sekvencijalne što pokazuje često pojavu u Pythonu kao i u drugim jezicima da konkurentnost može da uspori program ako se ne koristi na odgovarajući način.

Tabela 4.1: Performanse quicksort implementacija za niz od milion brojeva

Sekvencijalno izvršavanje		
Implementacija	Brzina izvršavanja [s]	Maks. upotreba memorije [kB]
Go	0,200	5.248
C	0,191	5.152
C++ std	0,708	6.456
C++ omp	0,196	6344
Python mp	2,040	67.472
Python pp	8,750	39.296
Konkurentno izvršavanje		
Implementacija	Brzina izvršavanja [s]	Maks. upotreba memorije [kB]
Go	0,149	6.540
C	0,096	21.420
C++ std	0,297	6.776
C++ omp	0,197	6.264
Python mp	5,260	59.616
Python pp	4,050	49.664

Tabela 4.2: Dužine kôda quicksort implementacija

	Go	C	C++ std	C++ omp	Python mp	Python pp
Br. linija koda	92	150	85	80	55	39

Sve preuzete implementacije su dobro iskomentarisane, prilično razumljive i jednostavne za korišćenje. Najveći broj dostupnih implementacija postoji za C i C++ među kojima je najzastupljenija OpenMP biblioteka, dok je za Python dostupan samo mali broj. Napomena da postoje C++ programi koji se mogu izvršavati kao C programi jer ne sadrže ništa specifično za C++ što važi i za pomenutu C++ OpenMP implementaciju.

Dužine programskih kodova se mogu pogledati u tabeli 4.1. Primećujemo da C ima najveći broj linija kôda usled neophodne alokacije memorije i provera greške. Sa druge strane, Pythonu je potreban najmanji broj linija za realizaciju algoritma, međutim, iako je njegov kôd koncizan, manje je razumljiv. Konkurentnost je najjednostavnije realizovati u C++-u i Go-u, bez neophodnih alokacija memorije i komplikovanih poziva funkcije, za razliku od C-a i intuitivno je jasnije koji delovi kôda se izvršavaju konkurentno i na koji način, za razliku od Python-a.

4.3 Množenje matrica

Opis algoritma

Go

C

C++

Python

Rezultati

4.4 Eratostenovo sito

Opis algoritma

Go

C

C++

Python

Rezultati

4.5 Složeniji primeri u programskom jeziku Go

Glava 5

Zaključak

Literatura

[1] David Chisnall. *Parallel quicksort example in C*. URL: http://cs.swan.ac.uk/~csdavec/HPC/sort.c.html (posećeno 09/05/2017).

[2] Alexander Demin. *Multi-threaded QuickSort*. 2013. URL: http://demin.ws/blog/english/2012/04/28/multithreaded-quicksort/ (posećeno 09/05/2017).

[3] Norman Matloff. *Programming on Parallel Machines*. University of California, Davis, 2010, str. 73–74. URL: http://heather.cs.ucdavis.edu/~matloff/158/PLN/ParProcBookS2010.pdf.

[4] pHag. *quickSort-openmp*. 2015. URL: https://github.com/pHag/id1217/blob/master/hw2/ex2/quickSort-openmp.c (posećeno 09/05/2017).

[5] Vitalii. *Quick Sort Example*. 2008. URL: http://www.parallelpython.com/component/option,com_smf/Itemid,1/topic,138.0 (posećeno 09/05/2017).