

Kauno technologijos universitetas

Informatikos fakultetas

Lygiagretusis programavimas

Lygiagreti „bitonic“ rikiavimo algoritmo implementacija „Go“ programavimo kalboje

Ataskaita

Atliko

IFF-6/10 grupės studentas

Margiris Burakauskas

Priėmė

doc. Romas Marcinkevičius,

lekt. Mindaugas Jančiukas

Kaunas, 2018

Turinys

[Užduotis 2](#_Toc532879468)

[Užduoties analizė 2](#_Toc532879469)

[Programos aprašymas 2](#_Toc532879470)

[Programos tekstas su komentarais 3](#_Toc532879471)

[Testavimas 6](#_Toc532879472)

[Vykdymo laiko kitimo tyrimas 6](#_Toc532879473)

[Išvados 7](#_Toc532879474)

[Literatūra 7](#_Toc532879475)

# 

# Užduotis

Parašyti lygiagrečią bitoninio rūšiavimo algoritmo implementaciją Go kalboje. Programos veikimas turi būti nepriklausomas nuo duomenų, turi būti galimybė keisti gijų skaičių bei duomenų kiekį.

# Užduoties analizė

**Bitoninis rūšiavimas** – vienas iš anksčiausiai (1968 m.) sukurtų lygiagrečių rūšiavimo algoritmų. Tai vienas iš greičiausių rūšiavimo tinklų (angl. sorting network) algoritmų. Rūšiavimo tinklai yra speciali algoritmų rūšis, kai sulyginimų seka nepriklauso nuo duomenų.

Bitoninę seką galima apibrėžti taip: vadinama bitonine seka, jeigu egzistuoja toks , kad . Kitaip dar galima būtų pasakyti, kad bitoninė seka yra mažėjančių ir didėjančių skaičių sekų (arba atvirkščiai) susiejimas. Pavyzdžiui, yra bitoninė seka.

Bitoninis rūšiavimas yra sulygink – sukeisk (angl. compare - exchange) tipo. Pirmiausia yra sudaroma bitoninė seka, o tada ji padalijama per pusę ir lyginami tik abiejų pusių atitinkami elementai – pirmas su pirmu, antras su antru ir t.t.; jie sukeičiami, jei pirmosios pusės elementas didesnis. Tada abi sekos pusės yra vėl dalinamos pusiau, ir veiksmai kartojami tol, kol gaunama seka iš 1 elemento – tada masyvas jau surikiuotas.

Lygiagretumas naudojamas pasitelkiant „Go“ kalbos ko-rutinas *go*. Naudojant globalų kintamąjį ir jį saugantį *mutex* užraktą yra kuriamos gijos, jei jau sukurtų gijų skaičius neviršija nurodyto. Apie baigtą darbą gijos savo tėvinėms gijoms praneša kanalais.

# Programos aprašymas

*// grąžina, kiek laiko praėjo nuo start laiko  
func* timeTrack(start time.Time, name string)

*// palygina i ir j elementus a masyve ir sukeičia vietomis jei neatitinka ascending reikšmės  
func* compareAndSwap(a []int, i, j int, ascending bool)

*// surikiuoja ir sulieja dvi bitoninės eiles a masyve  
func* bitonicMerge(a []int, low, length int, ascending bool)

*// surikiuoja ir sulieja dvi bitoninės eiles a masyve  
// jei galima, visa tai daro lygiagrečiai  
func* bitonicMergeAsync(a []int, low, length int, ascending bool, guardChan *chan*<- *struct*{}, wasConcurrent bool)

*// padalina a masyvą į dvi dalis ir joms iškviečia bitonicMerge  
func* bitonicSort(a []int, low, length int, ascending bool)

*// padalina a masyvą į dvi dalis ir joms iškviečia bitonicMerge  
// jei galima, visa tai daro lygiagrečiai  
func* bitonicSortAsync(a []int, low, length int, ascending bool, guardChan *chan*<- *struct*{}, wasConcurrent bool)

*// pagrindinė rikiavimo funkcija, pradeda rikiavimą ir matuoja bei į konsolę išveda vykdymo laiką  
func* sortB(a []int, ascending bool)

*// grąžina size dydžio masyvą su pseudo-atsitiktinai sugeneruotais skaičiais  
func* generateRandomIntegerArray(size int) []int

# Programos tekstas su komentarais

*package* main  
  
*import* (  
 "fmt"  
 "math"  
 "math/rand"   
 "os"  
 "sort"  
 "sync"  
 "time"  
)  
  
*var* size int  
*var* threadCount int  
*var* threadCountCurrent = 0  
*var* mutex = &sync.Mutex{}  
  
*// grąžina, kiek laiko praėjo nuo start laiko  
func* timeTrack(start time.Time, name string) {  
 elapsed := time.Since(start)  
 fmt.Printf("%s took %s\n", name, elapsed)  
}  
  
*// palygina i ir j elementus a masyve ir sukeičia vietomis jei neatitinka ascending reikšmės  
func* compareAndSwap(a []int, i, j int, ascending bool) {  
 *if* ascending == (a[i] > a[j]) {  
 a[i], a[j] = a[j], a[i]  
 }  
}  
 *// surikiuoja ir sulieja dvi bitoninės eiles a masyve  
func* bitonicMerge(a []int, low, length int, ascending bool) {  
 *if* length > 1 {  
 *var* k = length / 2  
   
 *for* i := low; i < low+k; i++ {  
 compareAndSwap(a, i, i+k, ascending)  
 }  
   
 bitonicMerge(a, low, k, ascending)  
 bitonicMerge(a, low+k, k, ascending)  
 }  
}  
  
*// surikiuoja ir sulieja dvi bitoninės eiles a masyve  
// jei galima, visa tai daro lygiagrečiai  
func* bitonicMergeAsync(a []int, low, length int, ascending bool, guardChan *chan*<- *struct*{}, wasConcurrent bool) {  
 *if* length > 1 {  
 *var* k = length / 2  
   
 *for* i := low; i < low+k; i++ {  
 compareAndSwap(a, i, i+k, ascending)  
 }  
   
 mutex.Lock()  
 *if* wasConcurrent && threadCountCurrent+2 <= threadCount {  
 threadCountCurrent += 2  
 mutex.Unlock()  
   
 *var* chan1 = make(*chan struct*{})  
 *var* chan2 = make(*chan struct*{})  
   
 *go* bitonicMergeAsync(a, low, k, ascending, chan1, true)  
 *go* bitonicMergeAsync(a, low+k, k, ascending, chan2, true)  
   
 <-chan1  
 <-chan2  
 } *else* {  
 mutex.Unlock()  
 bitonicMerge(a, low, k, ascending)  
 bitonicMerge(a, low+k, k, ascending)  
 }  
 }  
   
 *if* wasConcurrent {  
 guardChan <- *struct*{}{}  
 }  
}  
  
*// padalina a masyvą į dvi dalis ir joms iškviečia bitonicMerge  
func* bitonicSort(a []int, low, length int, ascending bool) {  
 *if* length > 1 {  
 *var* k = length / 2  
   
 bitonicSort(a, low, k, ascending)  
 bitonicSort(a, low+k, k, !ascending)  
   
 bitonicMerge(a, low, length, ascending)  
 }  
}  
  
*// padalina a masyvą į dvi dalis ir joms iškviečia bitonicMerge  
// jei galima, visa tai daro lygiagrečiai  
func* bitonicSortAsync(a []int, low, length int, ascending bool, guardChan *chan*<- *struct*{}, wasConcurrent bool) {  
 *if* length > 1 {  
 *var* k = length / 2  
   
 mutex.Lock()  
 *if* wasConcurrent && threadCountCurrent+2 <= threadCount {  
 threadCountCurrent += 2  
 mutex.Unlock()  
   
 *var* chan1 = make(*chan struct*{})  
 *var* chan2 = make(*chan struct*{})  
   
 *go* bitonicSortAsync(a, low, k, ascending, chan1, true)  
 *go* bitonicSortAsync(a, low+k, k, !ascending, chan2, true)  
   
 <-chan1  
 <-chan2  
   
 bitonicMergeAsync(a, low, length, ascending, guardChan, true)  
 } *else* {  
 mutex.Unlock()  
 bitonicSort(a, low, k, ascending)  
 bitonicSort(a, low+k, k, !ascending)  
   
 bitonicMerge(a, low, length, ascending)  
 }  
 }  
   
 *if* wasConcurrent {  
 guardChan <- *struct*{}{}  
 }  
}  
  
*// pagrindinė rikiavimo funkcija, pradeda rikiavimą ir matuoja bei į konsolę išveda vykdymo laiką  
func* sortB(a []int, ascending bool) {  
 *// kanalas, kuriuo gaunamas pranešimas apie rikiavimo pabaigą  
 var* guardChan = make(*chan struct*{})  
 *defer* timeTrack(time.Now(), fmt.Sprintf("Bitonic sort of size %d using %d threads took", len(a), threadCount))  
   
 *go* bitonicSortAsync(a, 0, len(a), ascending, guardChan, true)  
 <-guardChan  
 threadCountCurrent = 0  
}  
  
*// grąžina size dydžio masyvą su pseudo-atsitiktinai sugeneruotais skaičiais  
func* generateRandomIntegerArray(size int) []int {  
 arr := make([]int, size)  
   
 rand.Seed(time.Now().Unix())  
   
 *for* i := 0; i < size; i++ {  
 arr[i] = rand.Intn(size \* 3)  
   
 *if* rand.Intn(2) > 0 {  
 arr[i] = 0 - arr[i]  
 }  
 }  
   
 *return* arr  
}  
  
*func* main() {  
 *// patikrinamas argumentų skaičius  
 if* len(os.Args) != 3 {  
 fmt.Println("Netinkami argumentai. Naudojimas: [programos vardas].exe [gijų skaičius] [duomenų masyvo dydis]. Teisingam rikiavimui duomenų masyvo dydis turi būti dvejeto laipsnis.")  
 *return* }

*// priskiriamos pradinės reikšmės iš argumentų  
 var* err error  
 threadCount, err = strconv.Atoi(os.Args[1])  
 size, err = strconv.Atoi(os.Args[2])  
   
 *// patikrinama, ar buvo pateikti teisingi argumentai  
 if* err != nil {  
 fmt.Println("Blogi argumentai. Naudojimas: [programos vardas].exe [gijų skaičius] [duomenų masyvo dydis]. Teisingam rikiavimui duomenų masyvo dydis turi būti dvejeto laipsnis.")  
 *return* }

*// sukuriami duomenų masyvai  
 var* a = generateRandomIntegerArray(size)  
 *var* a0 = make([]int, size)  
 *var* a1 = make([]int, size)  
   
 copy(a0, a)  
 copy(a1, a)  
   
 *// paleidžiamas rikiavimas* sortB(a0, true)  
 sortB(a1, false)  
   
 *// patikrinama, ar buvo surikiuota teisingai, naudojant standartinę Go kalbos rikiavimo biblioteką  
 if* !sort.IsSorted(sort.IntSlice(a0)) || !sort.IsSorted(sort.Reverse(sort.IntSlice(a1))) {  
 fmt.Println("sorting failed")  
 }  
}

# Testavimas

Naudota įranga:

* **Procesorius:** AMD Ryzen 5 1600 (6 branduoliai, 12 loginių procesorių, taktinis dažnis – 3,2 GHz).
* **Operatyvinė atmintis:** 16 GB DDR4 3200 MHz.
* **Operacinė sistema:** Windows 10 (1809 versija, 17763.194 kompiliavimo versija).
* **Integruota programavimo aplinka (IDE):** Jetbrains GoLand 2018.3.

Programos veikimo teisingumas yra patikrinamas kiekvieną kartą vykdant programą naudojant standartinę „Go“ kalbos biblioteką ir, jei buvo surikiuota neteisingai, išvedamas klaidos pranešimas.

Programos instaliuoti nereikia.

Naudojimas: [programos vardas].exe [gijų skaičius] [duomenų masyvo dydis]. Teisingam rikiavimui duomenų masyvo dydis turi būti dvejeto laipsnis.

# Vykdymo laiko kitimo tyrimas

# Išvados

Užduotį pavyko įgyvendinti. Programa veikia teisingai, gaunami rezultatai tokie, kokių tikėtasi.

Bitoninis rikiavimas gana gerai greitėja jį išskirsčius procesams. Tą atspindi vykdymo laiko kitimo diagrama. Duomenų kiekiui padidėjus dvigubai maždaug dvigubai padidėja ir vykdymo laikas – didėjimas linijinis. Gijų skaičių apsimoka didinti tik iki tiek, kiek yra procesoriaus branduolių – naudojant 12 gijų (tiek, kiek yra loginių procesorių) pagreitėjimo nėra arba jis labai nežymus. Esant pakankamai mažam duomenų kiekiui lygiagretinimas nėra optimalus – gijų kūrimas kainuoja daugiau laiko nei yra sutaupoma lygiagrečiai vykdant programą.

# Literatūra

[The Go Programming Language](https://golang.org/)

[Bitonic Sort - GeeksforGeeks](https://www.geeksforgeeks.org/bitonic-sort/)

[Stack Overflow - Where Developers Learn, Share, & Build Careers](https://stackoverflow.com/)

[Generating a Random Number · GolangCode](https://golangcode.com/generate-random-numbers/)