Eksperimenti jakog i slabog skaliranja za problem n-tela

Tehničke karakteristike sistema

- Model procesora: Intel(R) Core(TM) i5-7200U CPU 2.50GHz
- Organizacija cache memorije:
  - L1 Data cache 2 x 32 KB (8-way, 64-byte line)
  - L1 Instruction cache 2 x 32 KB (8-way, 64-byte line)
  - L2 cache 2 x 256 KB (4-way, 64-byte line)
  - L3 cache 3 MB (12-way, 64-byte line)
- Broj fizičkih jezgara: 2
- Broj logičkih jezgara: 4
- RAM: DDR4, 8GB
- OS: Windows 10 64-bit
- Dodatne biblioteke i njihove verzije:
  - matplotlib, 3.1.1
  - numpy, 1.17.3

Eksperimenti u Python programskom jeziku

Analiza koda

Merenjem vremena izvršavanja delova koda sa fiksnim brojem tela n = 50, dobijen je procenat sekvencijalnog dela koda koji se može paralelizovati i iznosi 89%, dok je ostatak sekvencijalnog koda 11%. Na osnovu ovih vrednosti mogu se izračunati teorijski maksimumi ubrzanja po Amdalovom zakonu. Pri pokretanju simulacije za veće vrednosti n (do 200 tela), uzeta je srednja vrednost procenata, za sekvencijalni deo koda bilo je potrebno samo 2% ukupnog vremena izvršavanja. Za n > 200 program je previše sporo radio zbog implementiranog brute-force algoritma pa za te vrednosti nisu rađeni eksperimenti.

Amdalov zakon i jako skaliranje

Po Amdalovom zakonu (1967.) maksimalno ubrzanje je ograničeno delom sekvencijalnog koda koji se ne može paralelizovati i definiše se na sledeći način:

$$speedup = \frac{1}{s + \frac{p}{N}}$$

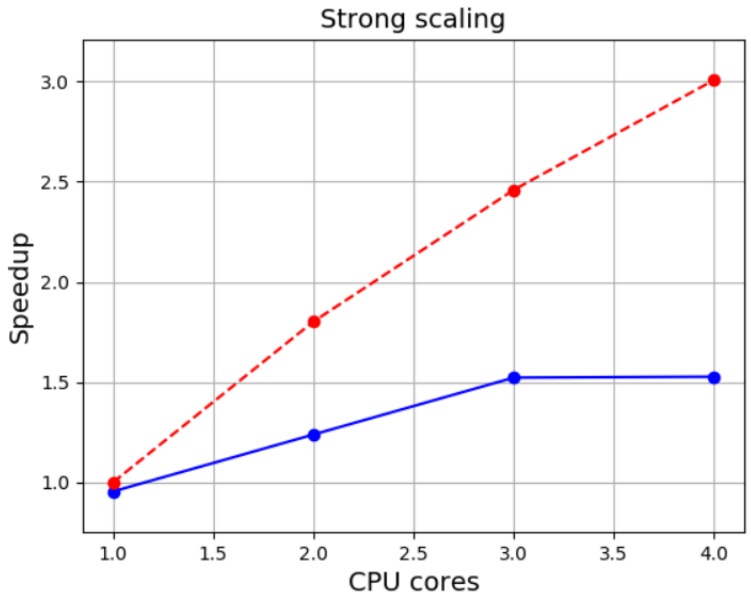
gde je *s* procenat vremena za izvršavanje sekvencijalnog dela, *p* je procenat vremena za izvršavanje dela koda koji može biti paralelizovan, a *N* je broj procesora. Amdalov zakon kaže da, za fiksni problem, gornja granica ubrzanja je ograničena sekvencijalnim delom koda i to predstavlja jako skaliranje.

Na primeru projekta, po Amdalovom zakonu jednačina bi izgledala ovako:

$$nbody\ speedup = \frac{1}{0{,}11 + \frac{0{,}89}{N}}$$

Na primer, ukoliko imamo 4 procesorska jezgra, maksimalno ubrzanje koje bi u teoriji moglo da se postigne je 3 puta.

Nad simulacijom Sunčevog sistema sa 10 većih tela (Sunce i planete) i 40 manjih tela (asteroidi ili komete) izvršen je eksperiment jakog skaliranja. Sekvencijalna i paralelna verzija programa je pokrenuta nekoliko puta, s tim da je u paralelnoj verziji menjan broj procesorskih jezgara na kojima se izvršavaju paralelizovani delovi koda (od 1 do maksimalno 4 jezgra koliko je ograničeno hardverom). Na grafiku (Slika 1 Jako skaliranje u Python-u) je na x-osi prikazan broj procesora, a na y-osi srednja vrednost ubrzanja koje je postignuto u paralelnom programu u odnosu na sekvencijalni (ubrzanje je izračunato kao t1/tn, gde je t1 vreme izvršavanja sekvencijalnog, a tn paralelnog programa). Takođe, isprekidanom crvenom linijom je prikazan teorijski maksimum ubrzanja po Amdalovom zakonu. Možemo zaključiti gledajući grafik da paralelna verzija dostiže ubrzanje nešto više od 1.5 puta na 4 procesora. Detaljnije rezultate vidimo u tabeli ispod (Tabela 1 Jako skaliranje u Python-u). Vrednosti trajanja izvršavanja programa su izražene u sekundama, za simulacije od 1000 iteracija sa vremenskim korakom 0.01 godina u svakoj iteraciji, što ukupno predstavlja kretanje tela u periodu od 10 godina.



Slika 1 Jako skaliranje u Python-u

CPU count	Serial mean	Parallel mean	Serial st.dev.	Parallel st.dev.	Speedup
1	17.49	18.33	0.33	0.21	0.95
2	-	14.13	-	0.55	1.23
3	-	11.50	-	0.49	1.52
4	-	11.37	-	0.42	1.54

Tabela 1 Jako skaliranje u Python-u

Gustafsonov zakon i slabo skaliranje

Gustafsonov zakon (1988.) je zasnovan na aproksimacijama da se paralelni deo koda linearno skalira sa količinom resursa, i da se serijski deo ne povećava s obzirom na veličinu problema. Prema tome dobijamo jednačinu za skalirano ubrzanje:

$$scaled\ speedup = s + pN$$

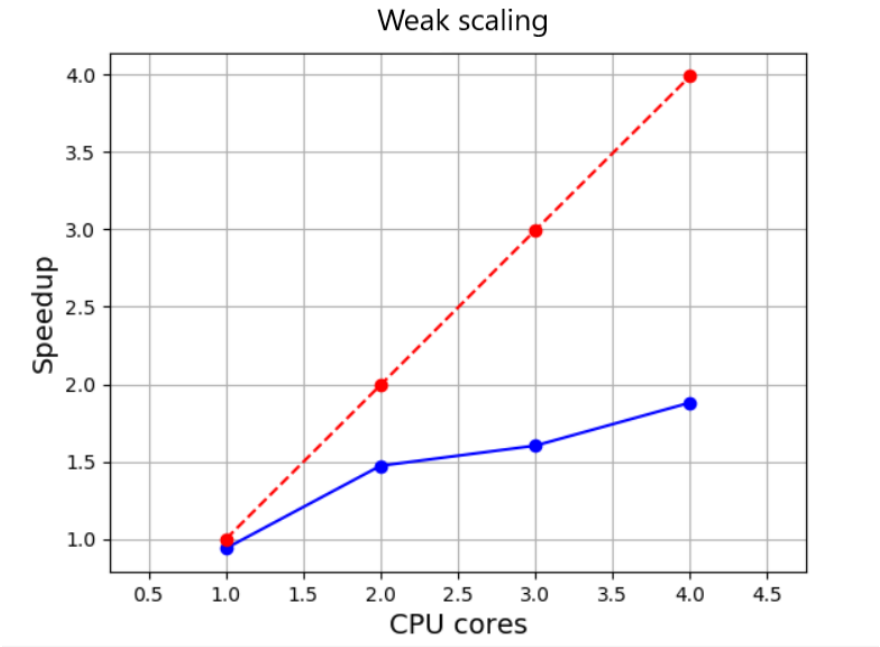
gde oznake imaju isto značenje kao kod Amdalovog zakona. Sa Gustafsonovim zakonom ubrzanje se povećava linearno sa brojem procesora, i ne postoji gornja granica za skalirano ubrzanje. Ovo se naziva slabo skaliranje, gde se ubrzanje računa na osnovu količine posla (za razliku od Amdalovog zakona gde imamo fiksni problem).

Za primer projekta, formula bi bila:

$$nbody\ scaled\ speedup = 0{,}02 + 0{,}98N$$

gde bi za npr. N = 4, maksimalno skalirano ubrzanje bilo 3,94 puta.

Nad simulacijom Sunčevog sistema sa različitim brojem nebeskih tela izvršen je eksperiment slabog skaliranja. Sa povećanjem broja procesora, povećan je i broj nebeskih tela, tako da su uvek bili jednako zaposleni. Ponovo je nekoliko puta pokrenuta simulacija, s tim da je sa brojem procesora, n povećavan za 50 (početno n je 50, pa zatim 100, 150 i 200 tela). Na grafiku ispod (Slika 2 Slabo skaliranje u Python-u) se može videti postignuto prosečno skalirano ubrzanje u odnosu na granicu ubrzanja po Gustafsonovom zakonu. U tabeli (Tabela 2 Slabo skaliranje u Python-u) se mogu videti detaljniji rezultati kao i prosečna vremena izvršavanja u sekundama, za 100 iteracija.



Slika 2 Slabo skaliranje u Python-u

CPU count-n	Serial mean.	Parallel mean.	Serial st.dev.	Parallel st.dev.	Scaled speedup
1-50	7.66	8.13	0.97	0.45	0.94
2-100	15.40	10.42	1.86	0.74	1.47
3-150	23.14	14.41	2.50	1.11	1.60
4-200	30.33	16.11	3.86	0.72	1.88

Tabela 2 Slabo skaliranje u Python-u

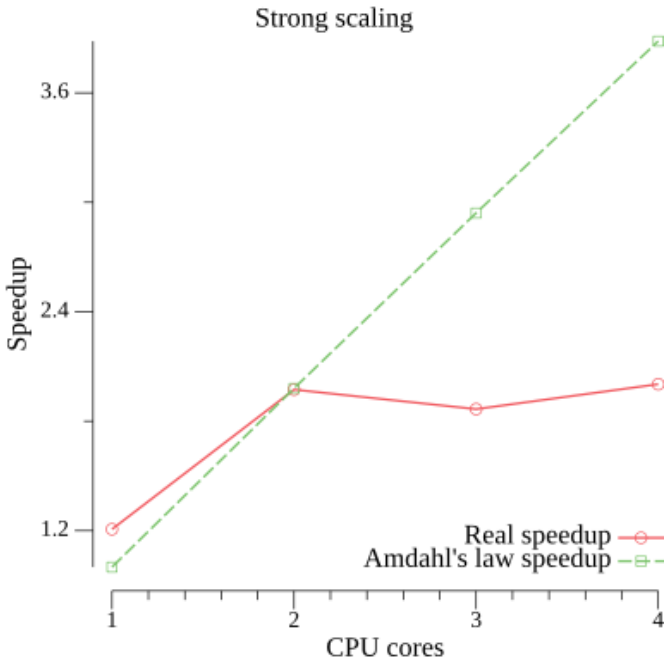
Eksperimenti u programskom jeziku Go

Analiza koda

Merenjem vremena izvršavanja delova koda sa fiksnim brojem tela n = 100 i fiksnim brojem iteracija 4000, dobijen je procenat ukupnog vremena izvršavanja sekvencijalnog dela koda koji se može paralelizovati i iznosi 99%, dok je ostatak izvršavanja sekvencijalnog koda samo 1%. Takođe, izvršavanje simulacije u Go programskom jeziku je znatno brže u odnosu na Python verziju programa, kao što je i očekivano.

Jako skaliranje

Nad simulacijom Sunčevog sistema sa 10 većih tela (Sunce i planete) i 90 manjih tela (asteroidi ili komete) izvršen je eksperiment jakog skaliranja. Sekvencijalna i paralelna verzija programa je pokrenuta tridesetak puta, s tim da je u paralelnoj verziji menjan broj procesorskih jezgara na kojima se izvršavaju paralelizovani delovi koda (od 1 do maksimalno 4 jezgra koliko je ograničeno hardverom). Na grafiku (Slika 3 Jako skaliranje u Go-u) je na x-osi prikazan broj procesora, a na y-osi srednja vrednost ubrzanja koje je postignuto u paralelnom programu u odnosu na sekvencijalni. Možemo zaključiti gledajući grafik da paralelna verzija dostiže ubrzanje nešto više od 2 puta puta na 4 procesora. Detaljnije rezultate vidimo u tabeli ispod (Tabela 3 Jako skaliranje u Go-u). Vrednosti trajanja izvršavanja programa su izražene u sekundama, za simulacije od 4000 iteracija sa vremenskim korakom 0.01 godina u svakoj iteraciji, što ukupno predstavlja kretanje tela u periodu od 40 godina.



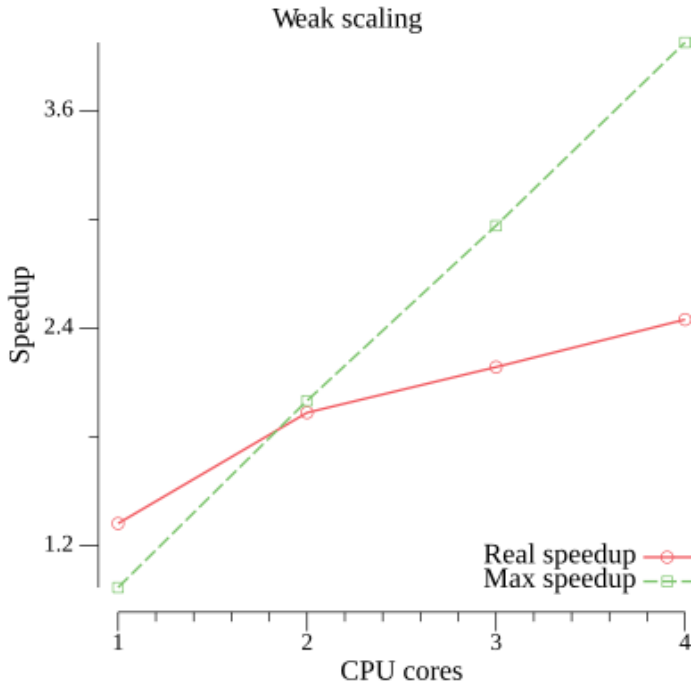
Slika 3 Jako skaliranje u Go-u

CPU count	Serial mean	Parallel mean	Serial st.dev.	Parallel st.dev.	Speedup
1	4.88	4.08	0.13	0.07	1.20
2	-	2.46	-	0.15	1.99
3	-	2.58	-	0.25	1.88
4	-	2.43	-	0.13	2.06

Tabela 3 Jako skaliranje u Go-u

Slabo skaliranje u programskom jeziku Go

Nad simulacijom Sunčevog sistema sa različitim brojem nebeskih tela izvršen je eksperiment slabog skaliranja. Sa povećanjem broja procesora, povećan je i broj nebeskih tela, tako da su uvek bili jednako zaposleni. Ponovo je tridesetak puta pokrenuta simulacija, s tim da je sa brojem procesora, n povećavan za 50 (početno n je 50, pa zatim 100, 150 i 200 tela). Na grafiku ispod (Slika 4 Slabo skaliranje u Go-u) se može videti postignuto prosečno skalirano ubrzanje u odnosu na granicu ubrzanja po Gustafsonovom zakonu. U tabeli (Tabela 4 Slabo skaliranje u Go-u) se mogu videti detaljniji rezultati kao i prosečna vremena izvršavanja u sekundama, za 100 iteracija.



Slika 4 Slabo skaliranje u Go-u

CPU count-n	Serial mean.	Parallel mean.	Serial st.dev.	Parallel st.dev.	Scaled speedup
1-50	0.034	0.028	0.003	0.001	1.31
2-100	0.126	0.068	0.002	0.002	1.93
3-150	0.278	0.128	0.005	0.005	2.18
4-200	0.497	0.205	0.010	0.001	2.44

Tabela 4 Slabo skaliranje u Go-u

U odnosu na Python, osim nekoliko puta bržeg izvršavanja programa, primetno je i da je ubrzanje veće na istom hardveru. Može se zaključiti da je Go bolji izbor u rešavanju problema u oblasti konkurentnog programiranja. Sadrži *goroutines* kao ugrađenu podršku za konkurentnost, i *channels* kao jednostavan način komunikacije između *goroutine*-a. One troše svega nekoliko kilobajta dodatne memorije, lake su za korišćenje, ne zahtevaju nikakvu komplikovanu implementaciju. *Go runtime* rukuje svom kompleksnošću.