Eksperimenti slabog i jakog skaliranja

- Algoritam nad kojim se vrše eksperimenti: Kanonovo množenje matrica

- Hardver upotrebljen za vršenje eksperimenta: Procesor AMD Ryzen 7 3700X @4.1GHz, 16 Threads, L1/L2/L3: 512KB, 4MB, 32MB

Objašnjenje algoritma

Kanonov algoritam je distribuiran algoritam za množenje dvodimenzionalnih nizova.

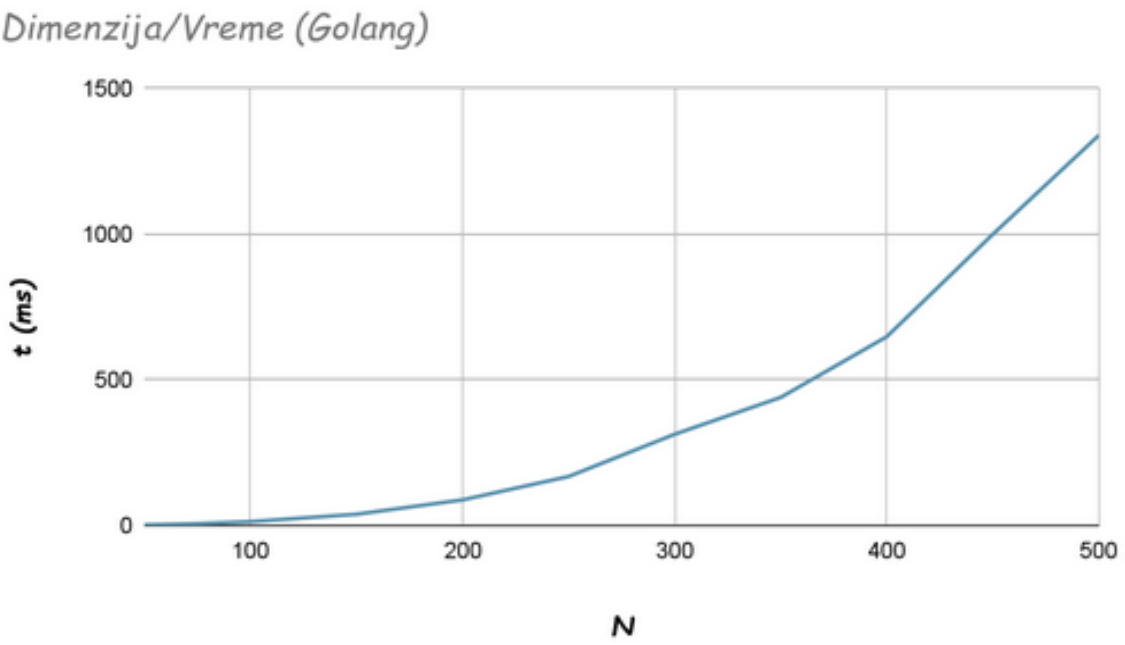
- Uzmimo dve N x N matrice A i B particionisane u P blokova
- $A(i, j)$ i $B(i, j)$ $(0 \leq i, j \leq \sqrt{P})$ veličine $(N / \sqrt{P}) \times (N / \sqrt{P})$
- Proces $P(i, j)$ inicijalno čuva $A(i, j)$ i $B(i, j)$ i računa blok $C(i, j)$ rezultujuće matrice
- Inicijalni korak algoritma "pomera" matrice A i B tako da svaki proces može krenuti sa računom odvojeno
- Ovo se izvršava tako što se sve vrste matrice $A(i)$ šiftuju za i koraka i kolone matrice $B(:, j)$ za j koraka
- Lokalni blokovi se množe
- Ponovo se vrše šiftovi za 1 korak
- Izvršava se množenje narednog bloka
- Ponavlja se dok svi blokovi nisu umnoženi i dodati u rezultujuću matricu C

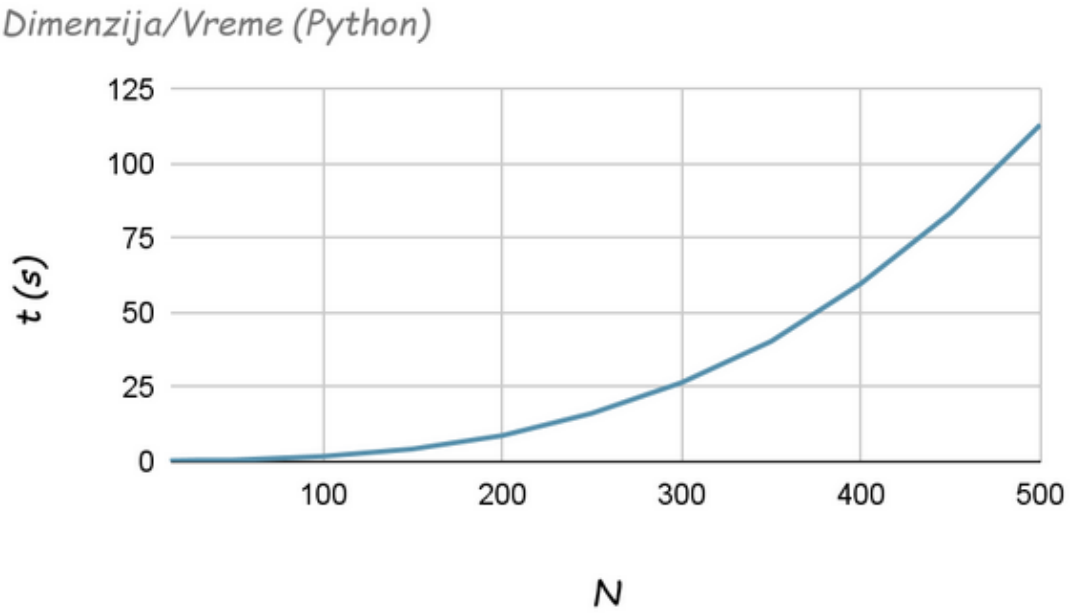
Implementirana je sekvencijalna i paralelna verzija algoritma, u jeziku Python i Golang.

U jeziku Python upotrebljena je biblioteka “multiprocessing” za paralelizaciju algoritma.

Sekvencijalna implementacija

Ispod su prikazani grafovi eksperimenata izvršenih nad sekvencijalnom implementacijom algoritma. X-osa predstavlja dimenzije množenih matrica, a y-osa totalno izvršno vreme algoritma.

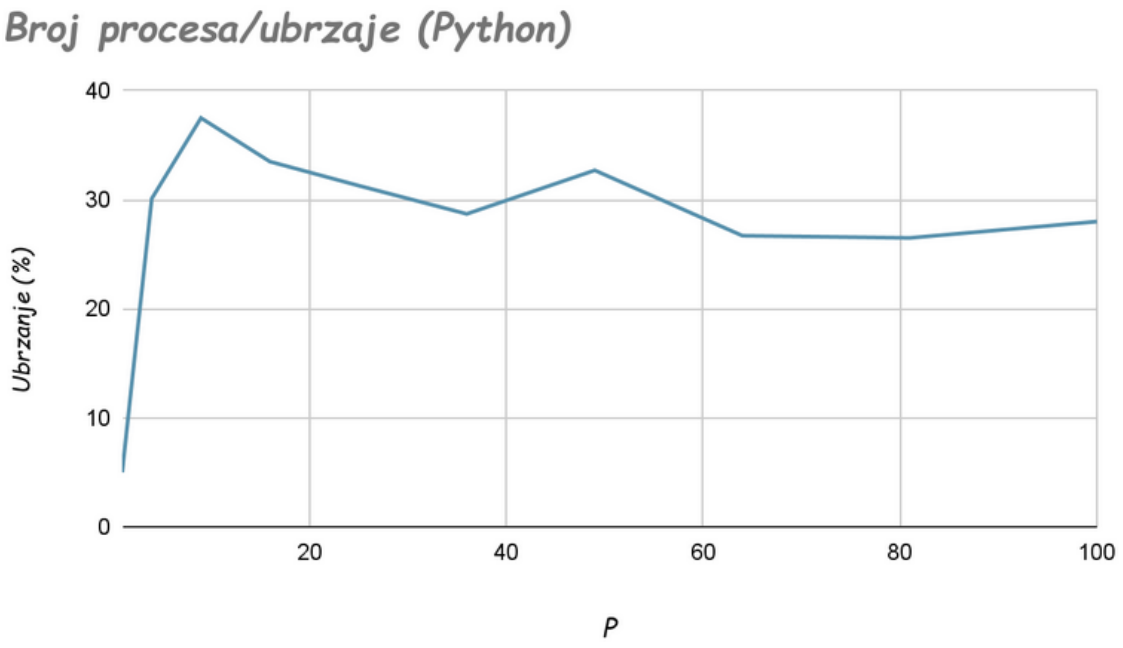


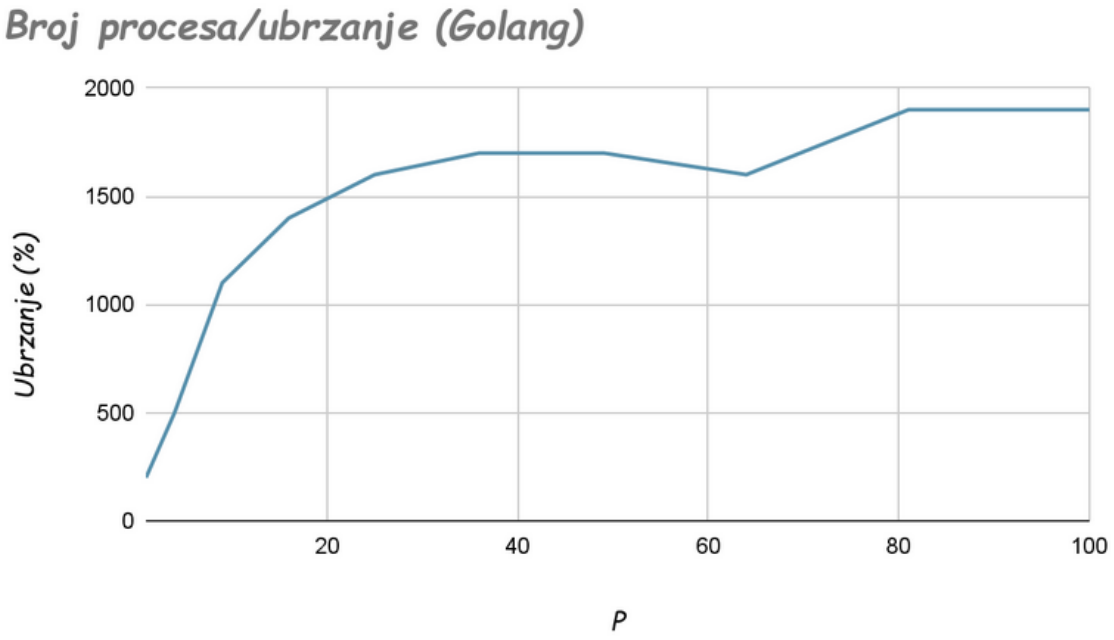


Eksperiment slabog skaliranja

Pre nego što navedemo rezultate ovog eksperimenta, bitno je reći da za paralelnu implementaciju algoritma broj procesa *P* mora biti kvadrat celog broja, i dimenzija matrice *N* mora biti deljiva sa korenom broja *P*.

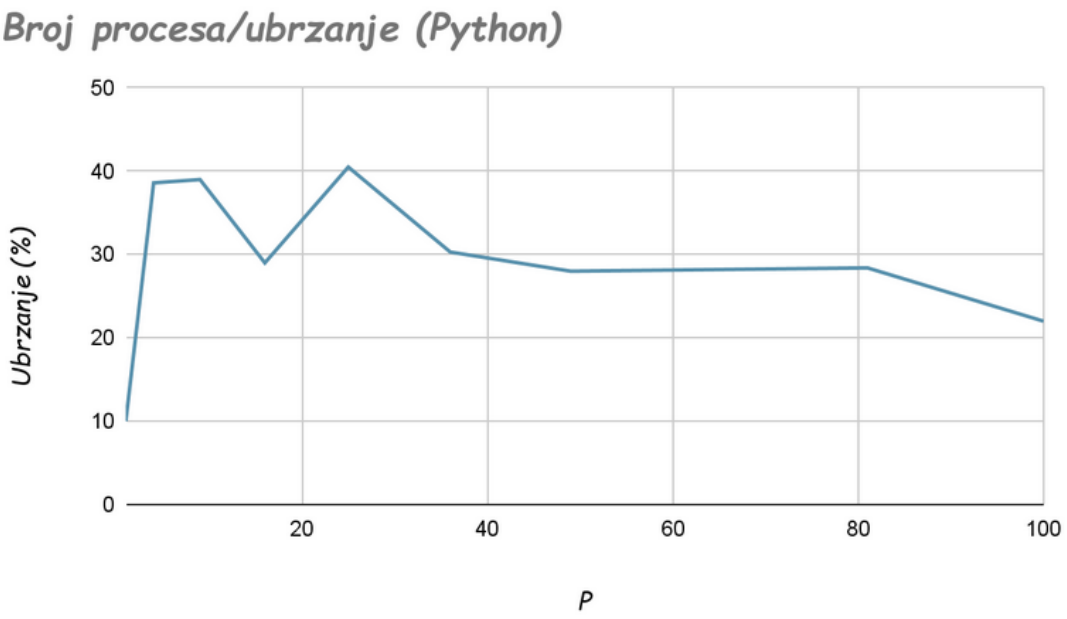
Za “baznu matricu” uzimamo matricu dimenzija 50x50, za *P* uzimamo brojeve 4, 9, 16, 25, 36, 49, 64, 81, 100 i u svakom pokretanju uzimamo matrice dimenzija (*N* * P_{sqrt}). Na graficima je prikazano ubrzanje paralelnog algoritma u odnosu na sekvencijalno rešenje.

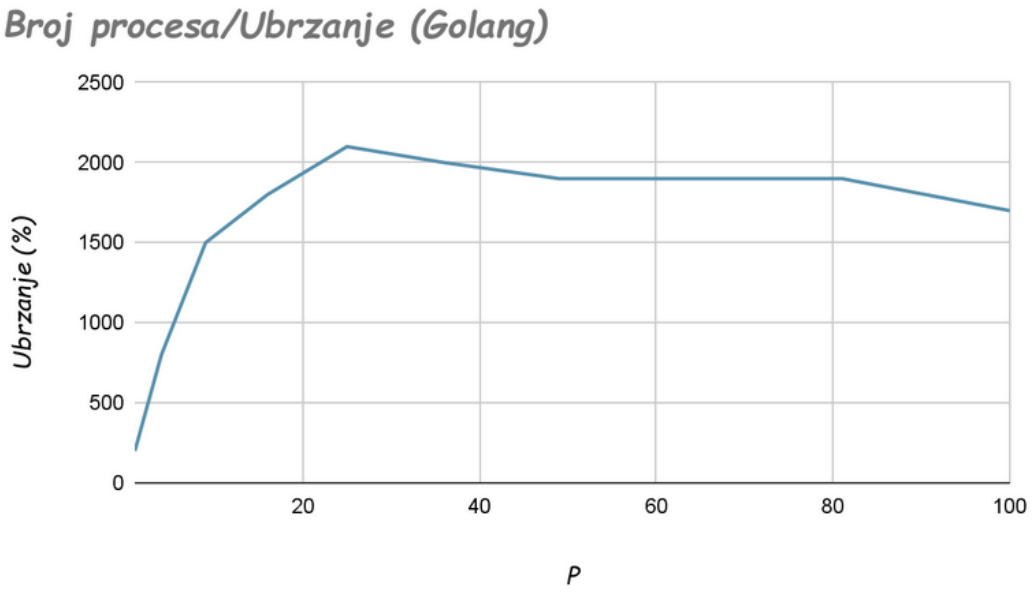




Eksperiment jakog skaliranja

Za dimenziju svih matrica u eksperimentu jakog skaliranja uzimamo $N = 420$, za P uzimamo brojeve 4, 9, 16, 25, 36, 49, 81, 100 i u svakom pokretanju uzimamo matrice dimenzija
($N * P_{sqrt}$). Na graficima je prikazano ubrzanje paralelnog algoritma u odnosu na sekvencijalno rešenje.





Zaključak

Kao što je i očekivano, najsporija implementacija algoritma je sekvencijalna implementacija, jer samo jedan procesor obavlja sve aritmetičke operacije u sekvencijalnom redosledu.

Vreme izvršavanja programa raste eksponencijalno povećavanjem dimenzija matrica i kod jezika Golang i kod jezika Python. Golang implementacija se izvršava znatno brže gde se kod dimenzija matrice 500 Golang program izvrši u 1.5s dok se Python program izvrši u 110s.

Kod eksperimenata slabog i jakog skaliranja implementiramo paralelizaciju algoritma sa određenim brojem procesa na osnovu kojeg se poslovi računanja dele među procesorima.

U jeziku Python glavni proces šalje i dobija odgovore putem metode ***apply_async.get.*** Po završetku svakog procesa sračunata podmatrica se dodaje u glavnu, finalnu matricu.

U jeziku Golang glavni (master) i drugi procesi (slaves) komuniciraju preko kanala i otvaraju paralelizovanu funkciju putem ***go func*** mehanizma. Na posletku se prikupljaju rezultati master kanala i rezultat se dodaje u finalnu matricu.

Najveće ubrzanje u odnosu na sekvencijalnu implementaciju se ostvarilo za sledeće brojeve procesa:

Jezik	Tip eksperimenta	Broj procesa	Ubrzanje
Python	Jak	36	41%
Golang	Jak	25	2100%
Python	Slab	9	38%
Golang	Slab	81	1800%