Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 681.324.06
DOI: https://doi.org/10.24867/25BE26Tanovic

IMPLEMENTACIJA PARALELNOG K-MEANS ALGORITMA ZA KLASIFIKACIJU CIFARA UPOTREBOM OPENMP I MPI BIBLIOTEKA

IMPLEMENTATION OF PARALLEL K-MEANS ALGORITHM FOR DIGIT CLASSIFICATION USING OPENMP AND MPI LIBRARIES

Anja Tanović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu prikazan je razvoj paralelnog k-means algoritma za klasifikaciju cifara. Opisan je osnovni k-means algoritam klasterizacije, kao i njegovo prilagođenje na algoritam za klasifikaciju skupa slika. Analizirane su mogućnosti paralelizacije algoritma i implementirane su različite verzije paralelnog k-means algoritma korišćenjem OpenMP i MPI biblioteka.*

Ključne reči: *Paralelni algoritmi, K-means algoritam, OpenMP, MPI, Klasifikacija cifara*

Abstract – *This paper presents the development of the parallel k-means algorithm for digit classification. The basic k-means clustering algorithm is described, as well as its adaptation to the image set classification algorithm. The possibilities of paralellizing the algorithm were analyzed and different versions of the parallel k-means algorithm were implemented using OpenMP and MPI libraries.*

Keywords: *Parallel algorithms, K-means algorithm, OpenMP, MPI, Digit Classification*

1. UVOD

Paralelni algoritam u računarstvu je algoritam podeljen na određen broj delova koji se izvršavaju na različitim procesorskim uređajima. Kombinovanjem rezultata pojedinačnih segmenata na kraju izvršavanja dobija se rezultat algoritma. Transformacija programa iz serijskog u paralelni oblik izvršavanja zahteva detaljnu analizu samog algoritma. Neki algoritmi se mogu jednostavnije podeliti na više nazavisnih zadataka, dok neke algoritme karakteriše izuzetna zavisnost između koraka izračunavanja, te je njihova paralelizacija kompleksnija.

K-means algoritam klasterizacije, poznat i pod nazivom klasterizacija metodom k-srednjih vrednosti, je algoritam čiji cilj je particionisanje skupa podataka u *k* klastera, gde svaki klaster predstavlja grupu podataka koji imaju slične osobine ili značenje. Ovaj problem je računski težak (NP-težak problem), tako da je njegovo izvršavanje vremenski zahtevno za veće grupe podataka, kao što su na primer skupovi podataka korišćeni u oblasti mašinskog učenja. Kako bi se omogućila efikasna klasterizacija velikih skupova podataka primenom k-means algoritma, potrebna je paralelizacija algoritma.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Vuk Vranjković, vanr. prof.

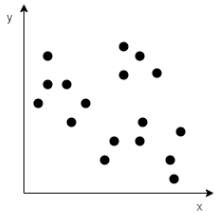
U drugom poglavlju rada biće ukratko opisani osnovni koraci k-means algoritma i implementacija k-means algoritma koji se koristi za klasifikaciju cifara. U trećoj celini biće opisana paralelizacija k-means algoritma korišćenjem OpenMP i MPI biblioteka za paralelno programiranje. U četvrtom poglavlju dati su rezultati i ubrzanje programa nakon testiranja različitih metoda paralelizacije algoritma. Na samom kraju, u petoj glavi, biće dati kratak pregled rezultata celokupnog rada i diskusija o mogućim unapređenjima algorima.

2. K-MEANS ALGORITAM ZA KLASIFIKACIJU CIFARA

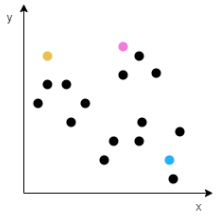
K-means algoritam je jedan od najstarijih i najšire korišćenih algoritama za klasterizaciju u oblasti istraživanja podataka (engl. *data mining*). Cilj algoritma je da se pronalaskom *k* klastera koji se ne preklapaju, podeli skup podataka na *k* grupa [1].

2.1. Osnovni koraci k-means algoritma

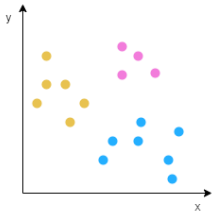
K-means algoritam može se ukratko opisati kroz sledeće korake. Prvo je potrebno izabrati *k* tačaka iz skupa podataka koje će predstavljati početne centroide, gde je *k* pozitivan ceo broj koji označava željeni broj klastera. Centroidi predstavljaju težišta klastera oko kojih se nalaze ostale tačke, i njihova pozicija se menja kroz iteracije algoritma. Nakon izbora početnih centroida, svaka tačka iz skupa podataka dodeljuje se najbližem centroidu.
Grupa tačaka koja je dodeljena jednom centroidu predstavlja jedan klaster.
Težište svakog klastera se zatim ažurira izračunavanjem srednjeg rastojanja između svih pripadnika datog klastera. Nakon promene težišta može se desiti da su neke od tačaka više udaljene od centorida klastera kojem pripadaju nego od centroida nekog od susednih klastera. Kako bi se dobila najbolja podela skupa na klastere potrebno je opisani proces dodele tačaka najbližim centoridima klastera i izračunavanje pozicija novih centroida ponoviti sve dok se ne desi da se centri klastera više ne menjaju [1].
Ilustrativni primer izvršavanja k-means algoritma nad skupom tačaka, za vrednost $k = 3$, prikazan je na slikama 1-3. Opisani koraci k-means algoritma mogu se implementirati korišćenjem nekog od standardnih programskih jezika. Za opisivanje k-means algoritma u ovom radu korišćen je C++ jezik.



Slika 1. *Početni skup tačaka*



Slika 2. *Izbor početnih centroida*



Slika 3. *Konačni rezultat klasterizacije*

2.2. Klasifikacija cifara upotrebom k-means algoritma

K-means algoritam često se koristi za klasifikaciju slika u oblasti mašinskog učenja. Prepoznavanje cifara pisanih rukom je važan problem u oblasti optičkog prepoznavanja karaktera i često je korišćeno kao test za različite algoritme mašinskog učenja. MNIST baza podataka rukom pisanih cifara predstavlja standard za testiranje algoritama mašinskog učenja namenjenih za ovu svrhu. Sastoji se iz 60000 cifara namenjenih za trening i 10000 za testiranje. Svaka slika u bazi je veličine 28 × 28 piksela. Baza podataka data je formi CSV datoteke [2].

K-means je nenadgledani (engl. *unsupervised*) algoritam mašinskog učenja čiji je cilj da particioniše *n* tačaka podataka u *k* klastera. Svaki podatak koji predstavlja odgovarajuću cifru ima dimenzionalnost 28·28=784. Reprezentacija cifara u programu urađena je pomoću klase *Image*. Polja klase definišu koordinate tačke u 784-dimenzionalnom sistemu (niz *coor[dim]*, gde je *dim* dimenzionalnost), labelu koja opisuje cifru predstavljenu na slici (*label*), klaster kojem tačka pripada (*cluster*) i rastojanje između tačke i najbližeg klastera (*minDist*). U klasi je definisana i metoda za izračunavanje rastojanja između date tačke i tačke prosleđene kao parametar.

K-means algoritam za treniranje i testiranje sastoji se iz sledećih koraka:

1. Učitavanje skupa podataka iz CSV datoteke
2. Izbor početnih centroida iz skupa tačaka
3. Izračunavanje rastojanja između tačaka i centara klastera i dodela tačaka najbližim klasterima
4. Izračunavanje koordinata novih centara klastera
5. Provera da li je bar jedan od centroida promenio svoje koordinate. Ukoliko jeste, potrebno je ponoviti korake 3, 4 i 5.
6. Izračunavanje tačnosti treniranja
7. Testiranje nad test skupom podataka
8. Izračunavanje tačnosti testiranja
9. Upis rezultata u CSV datoteku

Rezultati treniranja i testiranja ispisuju se na terminalu, a koordinate centroida upisuju se u posebnu CSV datoteku kako bi se u budućnosti mogli koristiti za predviđanje nad novim podacima cifara. Dodatno, meri se i vreme potrebno za treniranje i testiranje, koje će se koristiti za poređenja između serijske i paralelnih verzija algoritma.

3. PARALELIZACIJA K-MEANS ALGORITMA UPOTREBOM OPENMP I MPI BIBLIOTEKA

Korisno bi bilo omogućiti brže treniranje prethodno opisanog algoritma iz više razloga. Prvo, algoritam daje bolje rezultate za veće vrednosti parametra *k*, ali on tada postaje i značajno sporiji. Osim toga, algoritam može generisati različite rezultate pri istim parametrima zbog izbora početnih centroida. To znači da bi bilo neophodno izvršavati algoritam više puta, za velike vrednosti *k*, kako bi se dobili kvalitetni rezultati klasifikacije. Ovakvo treniranje bi moglo oduzeti mnogo vremena. Takođe, implementirani algoritam za klasifikaciju može se koristiti i za druge skupove slika a ne samo za cifre. Potencijalno neki od skupova mogu imati još veći broj slika, i same slike mogu biti značajno većih dimenzija, što bi dodatno usporilo algoritam. Jedno od mogućih rešenja za ubrzanje programa je paralelizacija algoritma.

3.1. Analiza mogućnosti paralelizacije k-means algoritma

Neophodan korak u paralelizaciji svakog algoritma je njegova detaljna analiza. Potpuno razumevanje algoritma olakšava podelu algoritma na odvojene zadatke i implementaciju efikasnog paralelnog algoritma. Korisno bi bilo usresrediti se na delove algoritma koji najduže traju i pokušati njih paralelizovati. U prethodnom poglavlju dati su koraci algoritma i može se zaključiti da se koraci 3, 4 i 5 ponavljaju potencijalno veliki broj puta, te bi njihova paralelizacija značajno pomogla pri ubrzanju programa. S obzirom na to da su ovi koraci međusobno zavisni, nije moguće istovremeno ih obavljati, već je potrebno paralelizaciju uraditi unutar svakog od koraka.

Posao koji je potrebno obaviti u koraku 3 je izračunavanje rastojanja između različitih centroida i tačaka. Ovo računanje se može podeliti na nezavisne taskove, s obzirom na to da se koordinate tačaka i centroida ne menjaju tokom celog koraka. Izvršavanje koraka 4 zahteva izračunavanja novih koordinata centroida koja se takođe mogu nezavisno obaviti. Provera da li je došlo do promene odgovarajućeg centra klastera nije zavisna od provere nad drugim centrom, tako da je i korak 5 moguće paralelizovati.

Analizom logike algoritma i programskog koda utvrđeno je da je moguće paralelizovati određene korake k-means algoritma i obezbediti kraće trajanje programa. Kako bi se algoritam efikasno implementirao, koristiće se biblioteke za paralelno programiranje, OpenMP i MPI.

3.2. Implementacija paralelnog k-means algoritma upotrebom OpenMP biblioteke

K-means algoritam moguće je paralelizovati korišćenjem modela deljene memorije pomoću OpenMP biblioteke. Paralelizacija se postiže dodavanjem *pragma* direktiva u kodu, vodeći računa o međuzavisnosti iteracija i balansiranju opterećenja. Prethodno analizirani koraci implementirani su u programskom kodu u obliku *for* petlji. Za paralelizaciju petlji korišćene su *pragma omp parallel for num_threads(t) schedule(OMP_SCHEDULE, chunksize)*

direktive, gde parametar *t* predstavlja broj niti koje se koriste za obavljanje zadataka, a *OMP_SCHEDULE* predstavlja strategiju raspodele posla između niti. Pokazuje se da je za petlje sa približno podjednakom količinom posla najbolje koristiti *static* raspodelu jer su iteracije podeljene nitima pre izvršavanja, dok je za petlje sa različitim trajanjem bolje koristiti *dynamic* raspodelu. Dodatno, parametar *chunksize* utiče na brzinu izvršavanja tako što se njegovim podešavanjem prave različite kombinovane raspodele [3].

Kako bi se u dobijeno ubrzanje programa moglo uporediti sa originalnim serijskim programom, neophodno je izmeriti vreme izvršavanja programa. Za merenje vremena korišćena je OpenMP funkcija *omp_get_wtime()*.

3.3. Implementacija paralelnog k-means algoritma upotrebom OpenMP i MPI biblioteka

Paralelizacija k-means algoritma pomoću OpenMP biblioteke omogućava veću brzinu izvršavanja programa na računaru sa više jezgara. Međutim, ako se za treniranje algoritma koriste računari koji su ograničeni po broju procesorskih jezgara, može se desiti da postignuto ubrzanje nije dovoljno dobro. U tom slučaju bi bilo korisno ako bi više računara moglo da radi zajedno na treniranju, doprinoseći većem ukupnom broju jezgara. Ovakav scenario omogućava MPI biblioteka koja koristi model razmene poruka između procesa na različitim računarima.

Empirijski je pokazano da najveći deo vremena u k-means algoritmu oduzima korak u kojem se izračunavaju rastojanja između tačaka i centara klastera. Ovaj posao je već podeljen na različita jezgra računara, ali bi bilo korisno kada bi još jezgara moglo da učestvuje u računanju. Paralelizacija je urađena tako što je svakoj niti dodeljen određeni broj iteracija petlje u kojima se računaju rastojanja između klastera i odgovarajućih tačaka, što se može posmatrati kao da je svaka nit dobila određenu grupu tačaka nad kojima se računaju rastojanja. S obzirom na to da je broj tačaka u skupu podataka veoma veliki, ima smisla dodatno podeliti ovaj posao na više delova, tako što se kreira više procesa koji imaju svoju grupu tačaka. Uz to, svaki proces ima mogućnost da podeli svoju grupu tačaka na svoja procesorska jezgra, korišćenjem *pragma* direktiva i niti, kao i ranije.

Problem koji nastaje nakon izračunavanja svih rastojanja, je to što sledeći korak izračunavanja centara klastera zahteva znanje o svim tačkama i njihovim pripadnostima klasterima nakon proračuna. To znači da je neophodno da svi procesi pošalju ostalima svoje rezultate o tačkama. Prilikom procene vremena potrebnog za slanje podataka između računara, izračunato je da slanje slika u obliku klase *Image* oduzima previše vremena. Kako bi se ovo rešilo neophodno je rekonstruisati kod tako da se minimizuje količina podataka koja se šalje. Kada se bolje pogledaju polja klase *Image* može se zaključiti da nakon izračunavanja nije potrebno slati sva polja jer se samo polja *cluster* i *minDist* menjaju nakon izračunavanja rastojanja između tačaka i centroida. Stoga, klasa *Image* može se podeliti na 2 strukture koje zajedno predstavljaju jednu tačku, ali se posmatraju odvojeno. Nove strukture nazvane su *Image_part1* i *Image_part2*. Struktura *Image_part1* sadrži polja *coor[dim]* i *label*, a struktura *Image_part2* sadrži polja *cluster* i *minDist* koja se menjaju tokom izvršavanja algoritma. Slanjem samo strukture *Image_part2* u svakoj iteraciji algoritma, umesto prethodne strukture *Image*, značajno se skraćuje vreme potrebno za slanje poruka između procesa.

Dakle, nakon izračunavanja rastojanja, svaki proces prosleđuje svim ostalim procesima svoje rezultate o rastojanjima i pripadnosti klasterima, koji se nalaze unutar struktura *Image_part2*. U ovu svrhu koristi se funkcija *MPI_Allgatherv*, koja je proširena verzija često korišćene MPI funkcije *MPI_Allgather*.

Kako bi se omogućilo pokretanje paralelnog k-means programa na više računara, potrebno je grupisati računare u takozvane MPI klastere, unutar lokalne računarske mreže. Komunikacija između računara je omogućena preko mreže, putem SSH protokola (engl. *Secure Socket Shell*), dok su podaci koji se koriste deljeni pomoću NFS protokola (engl. *Network File System*) koji obezbeđuje deljenje direktorijuma među računarima.

4. REZULTATI TESTIRANJA PARALELNIH IMPLEMENTACIJA K-MEANS ALGORTIMA

Cilj predstavljenih paralelnih implementacija je skraćenje vremena izvršavanja algoritma. U nastavku su dati rezultati testiranja paralelnih implementacija korišćenjem OpenMP i MPI biblioteke, kao i poređenje brzine izvršavanja u odnosu na serijsku implementaciju. Za pokretanje programa korišćena su dva računara istih karakteristika, koja poseduju po šest procesorskih jezgara.

U tabeli 1 dati su rezultati merenja vremena izvršavanja serijskog programa sa različitim parametrom *k*. Ovi podaci korišćeni su kasnije pri proračunu ubrzanja paralelnih implementacija.

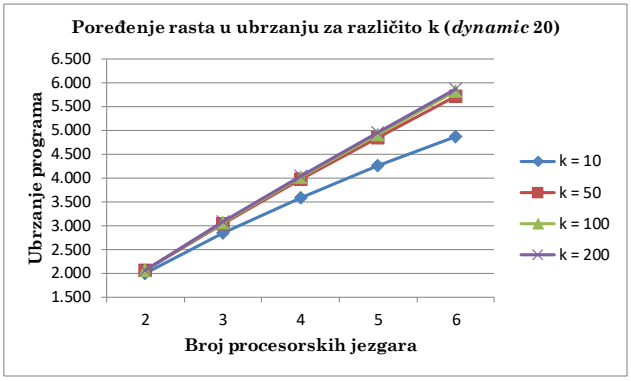
Tabela 1. *Rezultati testiranja serijske implementacije algoritma*

Serijska implementacija programa	**Vreme izvršavanja programa [s]**
k = 10 **broj iteracija: 45** **tačnost treniranja: 59.2217%** **tačnost testiranja: 59.32%**	115.518
k = 50 **broj iteracija: 194** **tačnost treniranja: 82.51%** **tačnost testiranja: 83.14%**	2343.61
k = 100 **broj iteracija: 120** **tačnost treniranja: 87.6933 %** **tačnost testiranja: 88.55%**	2911.29
k = 200 **broj iteracija: 79** **tačnost treniranja: 90.6933 %** **tačnost testiranja: 91 %**	3846.89

Iz datih rezultata može se uočiti da trajanje algoritma značajno raste sa rastom parametra *k*, što je i očekivano s obzirom na to da sa rastom broja centroida se povećava i broj prolazaka kroz petlje koda. Osim toga, tačnost treniranja i testiranja raste za veće vrednosti *k*, i dostiže 91% prilikom korišćenja 200 klastera.

Prva paralelna implementacija k-means algoritma koristi OpenMP biblioteku i deli algoritam na poslove koje obavljaju različite niti na dostupnim procesorskim

jezgrima računara. Strategija raspodele posla među nitima može biti različita u zavisnosti od konkretnog slučaja. Algoritam je testiran za različite strategije i parametre *chunksize* kako bi se pronašla najbrža implementacija. Rezultati ukazuju na to da je efikasnije koristiti *dynamic* strategiju nego *static*, ali da nema velike razlike u izboru *chunksize* parametra. Najbolji rezultati dobijeni su za strategije *dynamic, 10* i *dynamic, 20*, a nadalje će se koristiti strategija *dynamic, 20*. Kada je u pitanju dobijeno ubrzanje programa, primećuje se sličan rast u ubrzanju programa prilikom rasta vrednosti parametra *k*, ali se značajno bolji rezultati dobijaju za veće vrednosti *k*. Važan pokazatelj uspešne paralelizacije je činjenica da je dobijeno ubrzanje veoma blisko broju jezgara koja se koriste u izvršavanju (slika 4).



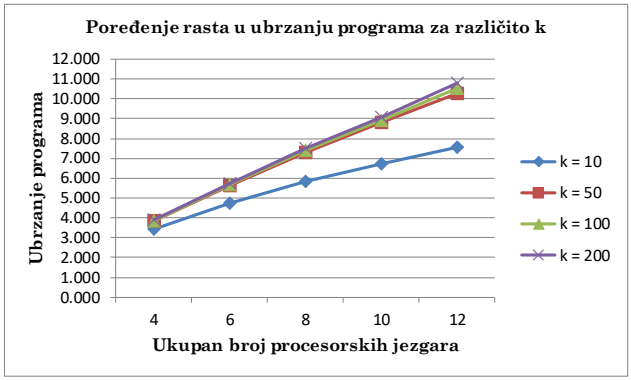
Slika 4. *Poređenje rasta u ubrzanju programa*

Testiranje je zatim obavljeno za implementaciju koja koristi OpenMP i MPI biblioteku, na dva računara. U tabeli 2 dato je dobijeno ubrzanje, a na slici 5 prikazan je rast u ubrzanju za različite vrednosti *k*. Treba naglasiti da se dati rezultati odnose na implementaciju programa u kojoj oba računara imaju pristup skupu podataka i nije potrebno prosleđivanje skupa ostalim procesorima.

Tabela 2. *Rezultati testiranja paralelne implementacije algoritma upotrebom OpenMP i MPI biblioteka*

Ubrzanje programa	broj procesorskih jezgara korišćeno od strane jednog procesa	Dva računara, 2 procesa	
		ukupan broj jezgara za izračunavanje	Ubrzanje programa *(broadcast nije potreban)*
k = 10	2	4	3.440
	3	6	4.762
	4	8	5.813
	5	10	6.741
	6	12	7.553
k = 50	2	4	3.841
	3	6	5.625
	4	8	7.300
	5	10	8.791
	6	12	10.280
k = 100	2	4	3.869
	3	6	5.683
	4	8	7.381
	5	10	8.933
	6	12	10.561
k = 200	2	4	3.903
	3	6	5.747
	4	8	7.529
	5	10	9.076
	6	12	10.775

Dati rezultati pokazuju da je ubrzanje korišćenjem dva računara značajno i da je rast ubrzanja sličan kao i u prethodnoj implementaciji. Razmatranjem celokupnog testiranja, može se zaključiti da je paralelni k-means algoritam 10.775 puta brži od serijske implementacije algoritma, ako se koristi 12 procesorskih jezgara. Kraće vreme izvršavanja programa značajno olakšava svako dalje testiranje ili optimizovanje algoritma.



Slika 5. *Poređenje rasta u ubrzanju programa korišćenjem dva računara*

5. ZAKLJUČAK

U ovom radu prikazan je razvoj k-means algoritma za klasifikaciju cifara, i kreiranje njegovih različitih paralelnih implementacija. Za paralelizaciju korišćene su biblioteke OpenMP i MPI i omogućeno je pokretanje algoritma na jednom ili više računara. Paralelni k-means algoritam daje ispravne rezultate, a postignuto ubrzanje programa je blisko broju jezgara koja se koriste.

Paraleni k-means algoritam se može unaprediti kako bi se dobilo još veće ubrzanje programa. Iako paralelni algoritam daje dovoljno veliko ubrzanje, veliki deo koda je ostao serijski napisan. Moguće je dodatno analizirati i rekonstruisati serijske delove koda, i onda ih paralelizovati. Osim toga, upotrebom neke druge biblioteke za paralelno programiranje potencijalno se mogu ostvariti bolji rezultati. Kombinovanje odgovarajućih biblioteka i dobro definisanog koda pruža veće mogućnosti u paralelizaciji algoritma.

6. LITERATURA

[1] J. Wu, "Cluster Analysis and K-means Clustering: An Introduction. U: Advances in K-means Clustering." Springer Theses. Springer, Berlin, Heidelberg, 2012. doi: 10.1007/978-3-642-29807-3_1.

[2] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," in IEEE Signal Processing Magazine, vol. 29, no. 6, strane 141-142, Nov. 2012, doi: 10.1109/MSP.2012.2211477.

[3] https://610yilingliu.github.io/2020/07/15/ScheduleinOpenMP/ posećeno septembar 2023.

Kratka biografija:

Anja Tanović rođena je u Novom Sadu 1999. god. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Energetika, elektronika i telekomunikacije odbranila je 2022.god.
kontakt: anjatanovic99@gmail.com