Paralelizacija postopka za učenje modela mešanice Gaussovih porazdelitev

Jasna Lenar, Janez Žibert

Univerza na Primorskem Fakulteta za matematiko, naravoslovje in informacijske tehnologije
Glagoljaška 8, 6000 Koper, Slovenia
E-pošta: jasna.lenar@gmail.com, janez.zibert@upr.si

Parallelization of Gaussian Mixture Model learning procedure

In speaker recognition systems the speech is modeled with Gaussian Mixture Model (GMM) that is build with EM algorithm. Due to amount of data to model that is increasing rapidly and the requests for more detailed results of modeling there is need to develop a parallel version of algorithm and implement it on a cluster of computers or a supercomputer. We propose a parallel version of the EM algorithm for modeling with GMM. We can analytically show that the speedup of the algorithm is linear for a big amount of data. We implemented our algorithm in C using the message passing paradigm. The experiments have been conducted on supercomputer Arctur-1. On the generated test data set of size 8,5 GB our implementation approached the linear speedup.

1 Uvod

Povod za razvoj vzporednega algoritma Expectation-Maximization (EM) za učenje modelov Gaussovih mešanic (GMM) je potreba po modeliranju količin podatkov, ki dosegajo razsežnosti od nekaj GB pa tja do nekaj TB. Vzporedni algoritem, ki smo ga razvili, se bo uporabljal za besedilno neodvisno razpoznavanje govorcev. V tej tehnologiji se danes za modeliranje govora uporablja od govorca neodvisni model, ki ga imenujemo splošni model govora (UBM) in temelji na modeliranju Gaussovih mešanic. Takšen način modeliranja govorcev je bil predstavljen v članku [1]. Težava gradnje modela UBM je velika količina vhodnih podatkov in računska zahtevnost modeliranja [2].

Na različnih področjih razpoznavanja vzorcev, npr. pri rekonstrukciji tomografskih slik [4] in senzoričnih omrežjih [5] so že razvili paralelne oblike algoritma, ki so učinkovite predvsem na ožjem področju. Splošnejša smer razvoja je paralelizacija za izvajanje na grafičnih procesorjih [6]. Dosežene so velike pohitritve, vendar zaradi omejenosti grafičnega pomnilnika ne moremo reševati poljubno velikih problemov. Po zgledu rešitve za GPU je razvit tudi paralelni sistem Distrim za izvajanje na multiprocesorjih [7], ki odpravlja pomanjkljivost premajhne skalabilnosti v [6]. Rešitev sloni na ustrezni preureditvi podatkov, da se lahko izkoristi več niti. S tem se zmanjša komunikacija, ki predstavlja problem pri paralelizaciji z message passing paradigmo.

Naš vzporedni EM algoritem smo razvili na podlagi paralelizacije postopka k-tih povprečij [8]. Paralelizacija sloni na message passing paradigmi, vendar ne predpostavlja dodatnih prilagoditev in uporabe niti, dobro pa rešuje problem skalabilnosti, ki je značilen za področje razpoznavanja govorcev in ga obstoječe enostavnejše paralelizacije ne rešujejo zadovoljivo.

2 EM algoritem za učenje modela mešanic Gaussovih porazdelitev

Pri razpoznavanju govorcev modeliramo podatke, ki so predstavljeni kot vektorji v *D*-dimenzionalnem prostoru. Vsaka komponenta vektorja predstavlja neko značilko. Modeliramo *N* vektorjev ***{X_1, … , X_N}***.

Podatke modeliramo z modelom GMM. Gaussovo porazdelitev definiramo s formulo

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\},$$

kjer je vektor x D-dimenzionalni vektor značilk, μ je D-dimenzionalni vektor povprečij, Σ pa je kovariančna matrika velikosti $D \times D$. Oznaka $|\Sigma|$ predstavlja determinanto matrike Σ. Mešanico Gaussovih porazdelitev zapišemo s formulo

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k),$$

kjer so vrednosti π_k uteži, za katere morata veljati pogoja $0 \leq \pi_k \leq 1$ in $\sum_{k=1}^{K} \pi_k = 1$. Število K predstavlja število mešanic Gaussovih porazdelitev.

Z modelom GMM modeliramo množico vektorjev tako, da iščemo K povprečij μ_k, K kovariančnih matrik Σ_k in K uteži π_k, ki maksimizirajo kriterijsko funkcijo logaritma verjetja

$$\ln p(X|\pi,\mu,\Sigma) = \sum_{n=1}^{N} \ln\left\{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k,\Sigma_k)\right\}.$$

Ta nam pove, kolikšna je verjetnost, da so bili podatki **X** generirani ravno iz parametrov $\boldsymbol{\mu}$, $\sum$ in $\boldsymbol{\pi}$. Hočemo, da je verjetnost čimvečja, zato funkcijo maksimiziramo. Problema maksimizacije ne moremo rešiti analitično, zato z iterativnim postopkom EM [3] poiščemo numerične rešitve.

Povzemimo po korakih postopek EM.

1. **Inicializacija** povprečij μ_k, kovarianc $\sum_k$ in uteži π_k ter določitev začetne vrednosti kriterijske funkcije logaritma verjetja.
2. **E korak:** Izračun pripadnostne funkcije $\gamma(z_{nk})$ s trenutnimi parametri modela.

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(x_n|\boldsymbol{\mu}_k, \sum_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\boldsymbol{\mu}_j, \sum_j)}$$

3. **M korak:** Ponoven izračun parametrov modela z upoštevanjem novih vrednosti pripadnostne funkcije.

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n$$

$$\sum_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^T$$

$$\pi_k = \frac{N_k}{N}$$

kjer je

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}) .$$

4. **Izračun kriterijske funkcije** logaritma verjetja.

$$\ln p(X|\mu, \Sigma, \pi) = \sum_{n=1}^{N} ln\left\{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right\}$$

Sledi preverjanje konvergenčnega kriterija. Če se vrednosti kriterijske funkcije ali vrednosti parametrov modela niso bistveno spremenili, končamo postopek, sicer se vrnemo na korak 2. Rešitev, ki jo najdemo s tem algoritmom ni nujno globalna. Globalno rešitev dobimo, če pravilno izberemo začetne parametre.

2.1 Predpriprava algoritma za paralelizacijo

Algoritem EM ni enostavno paralelizabilen, saj potrebujemo v vsakem koraku iteracije za izračun parametrov modela podatke o vseh vhodnih vektorjih $\boldsymbol{X_n}$. Zato ne moremo razdeliti vhodnih podatkov v začetku med procese in na koncu samo združiti rezultatov. Algoritem bi tudi radi uporabljali za modeliranje velikih količin podatkov, ki jih ne moremo shraniti v delovni pomnilnik enega samega procesorja.

Pri paralelizaciji takih statističnih algoritmov preurejamo algoritem v smislu zadostnih statistik. Če algoritem lahko zapišemo v sumacijski obliki, potem vhodne podatke razdelimo in na vsakem procesorju izračunamo delne vsote, ki jih na koncu seštejemo.

Formule za parametre modela preoblikujemo tako, da izrazimo ustrezne zadostne statistike.

$$\bar{\mu}_k = \sum_{n=1}^{N} \gamma(z_{nk}) x_n,$$

$$\bar{\Sigma}_k = \sum_{n=1}^{N} \gamma(z_{nk}) x_n \cdot x_n{}^{\mathrm{T}},$$

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}).$$

Končne vrednosti parametrov izrazimo z zadostnimi statistikami.

$$\mu_k = \frac{\bar{\mu}_k}{N_k}, \sum_k = \frac{\bar{\Sigma}_k - \bar{\mu}_k \cdot \mu_k{}^{\mathrm{T}}}{N_k}, \pi_k = \frac{N_k}{N}$$

Računanje zadostnih statistik bomo vključili v E korak algoritma. V M koraku bomo izračunali samo še končne vrednosti parametrov iz zadostnih statistik.

3 Vzporedni EM algoritem za učenje GMM

Vzporedno različico algoritma smo načrtovali po modelu Single Program Multiple Data (SPMD), po katerem vsi procesi izvajajo isti program, vendar vsak na svojem naboru podatkov. Uporabili smo paradigmo message-passing, ki se uporablja na sistemih s porazdeljenim pomnilnikom. Vsakemu procesu se dodeli zaporedna številka. Ko pišemo enoten program, ki ga izvajajo vsi procesi, z zaporedno številko procesa določimo, katere dele programa naj posamezen proces izvaja. Ker procesi ne dostopajo do istega delovnega pomnilnika, si med seboj pošiljajo sporočila.

```
1:  P  = MPI_Comm_size();
2:  pr = MPI_Comm_rank();
3:  LOGLIKE = MajhnoStevilo;
4:  if (pr == 0)
5:     Inicializiraj μ, ∑ in π;
6:  endif
7:  MPI_Bcast(μ, Σ, π, 0);
8:  do {
9:     OldLOGLIKE  = LOGLIKE;
10:    LOGLIKE = 0;
11:    for k = 1 to K
12:       N_k = 0; μ̄_k = 0; Σ̄_k = 0;
13:    endfor;
14:    for n = pr*(N/P)+1 to (pr+1)*(N/P)
15:       (N(x_n), p(x)) = loglike(x_n, μ, Σ, π);
16:       for k = 1 to K
17:          γ(z_nk) = π_k·N(x_nk) / p(x)
18:          N_k = N_k + γ(z_nk)
19:          μ̄_k = μ̄_k + γ(z_nk)x_n
20:          Σ̄_k = Σ̄_k + γ(z_nk)x_n · x_n^T
21:       endfor;
22:       LOGLIKE = LOGLIKE + p(x);
23:    endfor;
24:    for k = 1 to K
25:       MPI_AllReduce(N_k, μ̄_k, Σ̄_k, MPI_SUM);
26:       π_k = N_k / N
27:       μ_k = μ̄_k / N_k
28:       ∑_k = (Σ̄_k − μ̄_k·μ_k^T) / N_k
29:    endfor;
30:    MPI_AllReduce(LOGLIKE, MPI_SUM);
31: } while (LOGLIKE − OldLOGLIKE > ε)
```

Slika 1. Psevdokoda vzporednega algoritma

Vzporedni algoritem smo načrtovali tako, da eden od procesov izvaja vlogo koordinatorja. Ta v začetku prebere podatke iz vhodne datoteke ter jih razdeli ostalim procesom, nekaj pa jih obdrži zase.

Na sliki 1 je predstavljena psevdokoda vzporednega EM algoritma. Z ukazom *MPI_Comm_size()* določimo število procesov, ki jih imamo na voljo, z ukazom *MPI_Comm_rank()* pa vsakemu procesu dodelimo zaporedno številko. Koordinator postavi začetne vrednosti parametrov modela in jih z ukazom *MPI_Bcast()* posreduje ostalim procesom. Vrstice 9-23 predstavljajo E korak algoritma, vrstice 27-32 pa M korak algoritma. S funkcijo *loglike* vsak proces izračuna normalne porazdelitve s starimi parametri modela in tistim delom vektorjev X_n, ki mu je bil dodeljen. Sledi

izračun pripadnostne funkcije $\gamma(z_{nk})$. Nato si procesi z ukazom *MPI_Allreduce()* izmenjajo vrednosti zadostnih statistik in v koraku M zaključijo računanje novih parametrov modela. Vrednost kriterijske funkcije hranimo v spremenljivki *LOGLIKE*.

Preučimo še časovno zahtevnost algoritmov. Preoblikovani zaporedni algoritem iz poglavja 2.1 je skoraj enak vzporednemu algoritmu na sliki 1, le da teče glavna zanka od vrstice 14 do 23 po vseh vrednostih *n* in da nimamo komunikacije med procesi. Funkcija *loglike* je časovne zahtevnosti $O(K \cdot D)$. Naj I označuje število iteracij. Potem je časovna zahtevnost algoritma

$$T_{zaporedni} = O(I \cdot N \cdot K \cdot D).$$

Pri vzporednem algoritmu teče glavna zanka samo po delu vhodnih podatkov. Dodaten čas pri izvajanju vzporednega algoritma zahteva prenos podatkov med procesi. Časovno zahtevnost zapišemo s formulo

$$T_{vzporedni} = O\left(I \cdot \frac{N}{P} \cdot K \cdot D\right) + I \cdot K \cdot D \cdot T_{prenos},$$

kjer je $T_{prenos} = O(\log P)$, čas, ki je potreben za prenos številskega podatka vsem *P* procesom.

4 Rezultati meritev

Ko načrtujemo vzporedni algoritem, nas poleg pravilnosti rešitve zanima predvsem, koliko se zmanjša čas izvajanja pri enaki količini vhodnih podatkov in večjem številu procesov. To lastnost algoritma imenujemo pohitritev. Poleg tega nas zanima, za koliko lahko povečamo količino vhodnih podatkov, ki jih lahko obdelamo v enakem času. To lastnost vzporednega algoritma imenujemo skaliranje.

Vse meritve, ki so predstavljene v članku smo izvajali z vzporednim algoritmom. Ker predpostavljamo pravilnost delovanja algoritma, smo se ukvarjali samo z meritvami pohitritve delovanja vzporednega algoritma.

4.1 Metode

Meritve smo izvajali na superračunalniku Arctur-1 [9]. Sestavljen je iz 84 vozlišč IBM iDataPlex dx360 M3. Vsako vozlišče sestavljata dva procesorja Intel Xeon X5650 s šestimi jedri in hitrostjo 2,66 GHz. Velikost glavnega pomnilnika je 32 GB. Povezava med vozlišči je QLogic InfiniBand QDR 40 Gbit/s.

Program smo implementirali v programskem jeziku C z Intelovo knjižnico MPI. Časovne meritve smo izvajali s pomočjo funkcije *MPI_Wtime()*. Merili smo samo čas, ki smo ga porabili za učenje modela z EM algoritmom, torej čas za izvedbo iteracij v vrsticah 8-33 v psevdokodi na sliki 1, ne pa tudi čas za branje vhodnih podatkov iz vhodne datoteke in izpisovanje podatkov, ter računanje začetnih parametrov modela.

Za meritve smo s programom Octave generirali bazo podatkov velikosti 8,5 GB. Baza vsebuje 2^{25} vektorjev značilk dolžine 32. Pri poskusih z manjšim številom vzorcev *N* ali dimenzij *D* smo bazo ustrezno zmanjšali.

Začetna povprečja smo poljubno izbrali iz množice vhodnih podatkov, kovariančne matrike pa smo določili glede na razpršenost vhodnih podatkov. Modelirali smo z diagonalnimi kovariančnimi matrikami. Števila iteracij nismo določili vnaprej, ampak smo jih izvajali, dokler se je bistveno spreminjala vrednost kriterijske funkcije.

Vsako meritev smo pri istih začetnih parametrih ponovili petkrat, in iz meritev izračunali povprečno vrednost, da bi omilili vpliv zunanjih dejavnikov.

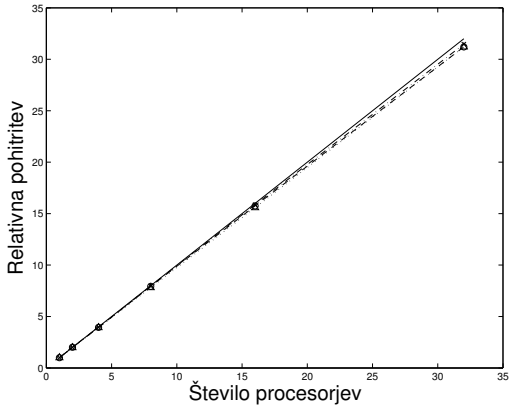
4.2 Pohitritev

Zanima nas, koliko hitrejše je računanje enako velikega problema, ko povečujemo število procesov. Rezultate bomo predstavili z grafi relativnih pohitritev, ki jih izračunamo po formuli

$$pohitritev = \frac{O(I \cdot N \cdot K \cdot D)}{O\left(I \cdot \frac{N}{P} \cdot K \cdot D\right) + I \cdot K \cdot D \cdot T_{prenos}}.$$

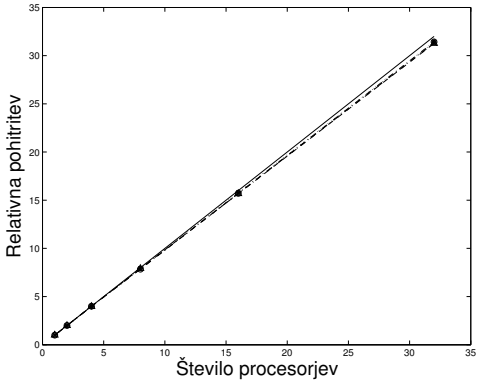
Iz formule lahko razberemo, da postane za velike vrednosti spremenljivke *N* čas prenosa podatkov zanemarljivo majhen. Zato se pohitritev bliža številu procesov *P*. Pričakujemo torej linearno pohitritev. Vidimo, da je linearna pohitritev zagotovljena pri vseh treh spremenljivkah *N*, *D* in *K*, če je le *N* dovolj velik.

Variiranje *N* (Slika 2): Merili smo pohitritve za velikosti vzorcev N = 2^{21}, 2^{22}, 2^{23}, 2^{24} pri vrednostih števila dimenzij D = 32 in števila komponent mešanice K = 8, ter številu procesorjev P = 1, 2, 4, 8, 16, 32.



Slika 2. Relativna pohitritev glede na število vzorcev

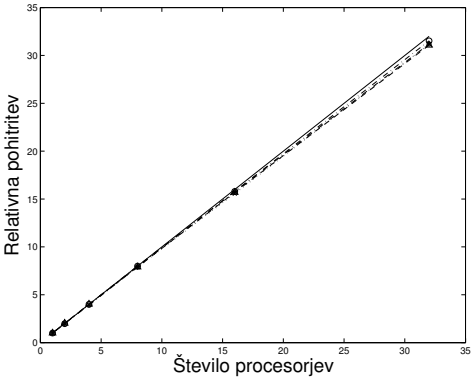
Polna črta na grafu predstavlja idealno krivuljo, prekinjene črte pa dejanske meritve. Za dane velikosti vzorcev so pohitritve skoraj linearne. Analitično smo pokazali, da je pohitritev linearna samo za dovolj velike vrednosti spremenljivke *N*. Izkazalo se je, da imamo pri vrednostih N = 2^{18}, 2^{19}, 2^{20} premajhno število vzorcev, zato algoritem ni dosegel linearne pohitritve, ker je prevladala komunikacija med procesi.



Slika 3. Relativna pohitritev glede na dimenzijo

Variiranje *D* (Slika 3): Merili smo pohitritve za število dimenzij D = 2, 4, 8, 16 pri vrednostih velikosti vzorcev N = 2^{25} in števila komponent mešanice K = 8. Ker je bil vzorec dovolj velik, so se tudi pohitritve za vsa števila dimenzij približale linearni.

Variiranje *K* (Slika 4): Merili smo pohitritve za število mešanic K = 2, 4, 8, 16 pri vrednostih velikosti vzorcev N = 2^{25} in števila dimenzij D = 32. Tudi tu smo zaradi dovolj velikega števila vhodnih podatkov dosegli skoraj linearne pohitritve.



Slika 4. Relativna pohitritev glede na število mešanic

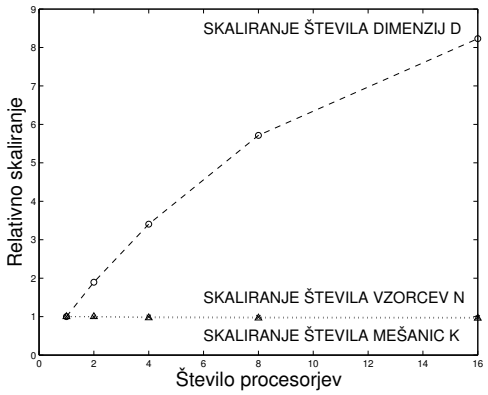
4.3 Skaliranje

S skaliranjem opišemo kolikokrat večji problem lahko rešimo v enakem času, če dodajamo nove procesorje.

Relativno skaliranje definiramo kot čas izvajanja ene iteracije na N točkah na enem procesu deljeno s časom izvajanja ene iteracije na $N \cdot P$ točkah na P procesorjih.

$$skaliranje = \frac{O(N \cdot K \cdot D)}{O(N \cdot P \cdot K \cdot D)/P + K \cdot D \cdot T_{prenos}}.$$

Za dovolj velike vrednosti N je skaliranje blizu konstanti 1.



Slika 5. Relativno skaliranje glede na vse tri spremenljivke

Merili smo skaliranje za velikosti vzorcev N = 2^{21}, 2^{22}, 2^{23}, 2^{24}, 2^{25} pri konstantnih vrednostih D = 2 in K = 32, število dimenzij D = 2, 4, 8, 16, 32 pri vrednostih K = 32 in N = 2^{21} in število mešanic K = 32, 64, 128, 256, 512 pri vrednostih D = 2 in N = 2^{21}. Meritve smo izvajali na številu procesorjev P = 1, 2, 4, 8, 16. Iz slike 5 je razvidno, da je skaliranje pri spremenljivkah N in K blizu konstanti 1, pri večanju števila dimenzij D pa se je izvajanje ene iteracije celo pohitrilo. Domnevamo, da je to posledica optimizacije izvajanja zanke v vrsticah 16-21 (Slika 1).

5 Zaključek

V članku smo predstavili vzporedni algoritem EM za modeliranje modelov Gaussovih mešanic. Algoritem smo razvili na podlagi paralelizacije postopka k-tih povprečij, ki je podobno strukturiran kot algoritem EM. Pri načrtovanju algoritma smo upoštevali možnosti, ki jih omogoča izvajanje programa na superračunalniku Arctur-1. Ta je sestavljen kot gruča strežnikov s hitrimi medsebojnimi povezavami. Predpostavili smo, da ves čas izvajanja programa vsi strežniki delujejo in so vsem ostalim dostopni za medsebojno komunikacijo.

Nadaljnje delo bo v prvi vrsti usmerjeno na meritve na večjih količinah realnih podatkov. Pri tem se bomo verjetno srečali z omejitvami računskih zmogljivosti, predvsem s pomanjkanjem delovnega pomnilnika. Tu bo verjetno potrebno povsem spremeniti pristop k obravnavi algoritma. Poleg tega bomo preverili tudi modeliranje s polnimi kovariančnimi matrikami. Osnovna ideja je, da bi lahko s takšnim algoritmom pospešili učenje UBM modelov, kar predstavlja veliko časovno omejitev pri razpoznavanju govorcev.

Literatura

[1] D. A. Reynolds, T. F. Quatieri, R. B. Dunn. Speaker Verification Using Adapted Gaussian Mixture Models. *Digital Signal Processing*, 10, 19-41, 2000

[2] B. Vesnicer, *Postopki normalizacije v sistemih za samodejno razpoznavanje govorcev*, doktorska disertacija, Fakulteta za elektrotehniko, Univerza v Ljubljani, 2010

[3] A. Dempster, N. Laird, D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39, 1-38, 1977

[4] P. E. Lopez-de-Teruel, J. M. Garcia, M. E. Acacio: The Parallel EM Algorithm and its Applications in Computer Vision. *Parallel and Distributed Processing Techniques and Applications*, 571-578, 1999

[5] R. Nowak: Distributed EM algorithms for density estimation and clustering in sensor networks. *Signal Processing, IEEE Transactions on,* 51(8), 2245 – 2253, avg. 2003.

[6] N. Kumar, S. Satoor, I. Buck: Fast Parallel Expectation Maximization for Gaussian Mixture Models on GPUs using CUDA. *High Performance Computing and Communications*, 103-109, 2009

[7] R. Yang, T. Xiong, T. Chen, Z. Huang, S. Feng: DISTRIM: Parallel GMM Learning on Multicore Cluster. *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, 2, 630-635, 2012

[8] I. Dhillon and D. Modha: A Data-Clustering Algorithm On Distributed Memory Multiprocessors. *Proceedings of Large-scale Parallel KDD Systems Workshop, ACM SIGKDD,* Aug. 15-18, 1999

[9] Arctur-1, https://www.arctur.si/reference/superracunalnik/