

Uvod v ogrodje NVIDIA RAPIDS

Krajše izobraževanje (NOO)

Potek izobraževanja

■ Ponedeljek, 16. 06. 2025, 10:00-14:00

- Uvod in predstavitev predavatelja, dostop do učnih materialov
- Ogrodje NVIDIA RAPIDS in njegove funkcionalnosti
- Pospeševanje algoritmov na GPE (knjižnice cuDF, cuML in cuGraph)
- **10:00-14:00 – teoretični in praktični del v F-103**

■ Torek, 17. 06. 2025, 10:00-14:00




- E2E cevovodi z NVIDIA RAPIDS
- Praktični primer 1: Priporočanje filmov
- Praktični primer 2: Vektorsko iskanje
- **10:00-14:00 – teoretični in praktični del v F-103**

■ Sreda, 18. 06. 2025, 10:00-14:00

- **10:00-14:00 – preverjanje znanja in anketa v F-103**

Predstavitev predavatelja

- **Kontakt:**

-  mladen.borovic@um.si
-  +386 2 220 7460
-  [LinkedIn](#)

- **Raziskovalna področja:**

- Aplikacije umetne inteligence
- Priporočilni sistemi in iskalniki
- Obdelava naravnega jezika
- Detekcija podobnih vsebin
- Visokozmogljivo računalništvo (HPC)



dr. Mladen Borovič

Univerza v Mariboru

Fakulteta za elektrotehniko, računalništvo in informatiko

Inštitut za računalništvo

Laboratorij za heterogene računalniške sisteme

Motivacija in cilji izobraževanja

- Predmetniki študijskih programov **ne zajamejo** vseh aktualnih tematik
- Dodatna izobraževanja kot **alternativna oblika podajanja znanja**
- Krajši format, hiter in učinkovit pristop k pridobivanju novih znanj
- Cilji tega izobraževanja 🙌
 - Približati udeležencem tematike na področju **visokozmogljivega računalništva**
 - Naučiti udeležence **dobrih praks** obdelave velepodatkov na GPE
 - **Uporaba ogrodja NVIDIA RAPIDS z namenom pospeševanja algoritmov**
 - Demonstrirati pridobljena znanja **na praktičnih primerih**
- Na koncu izobraževanja se izvaja **preverjanje znanja**
 - Pogoji za pridobitev mikrodokazila (1 ECTS)
- **Anketa**
 - Povratna informacija o vaši izkušnji udeležbe na krajšem izobraževanju



Učni materiali in gradivo


■ **GitHub** - <https://github.com/lhrs-workshops/noo-uonr>

■ **Prosojnice in zvezki Jupyter**

- Prosojnice vsebujejo teoretični del in so barvno označene v zgornjem desnem kotu

 **Prvi dan** (16. 06. 2025)

 **Drugi dan** (17. 06. 2025)

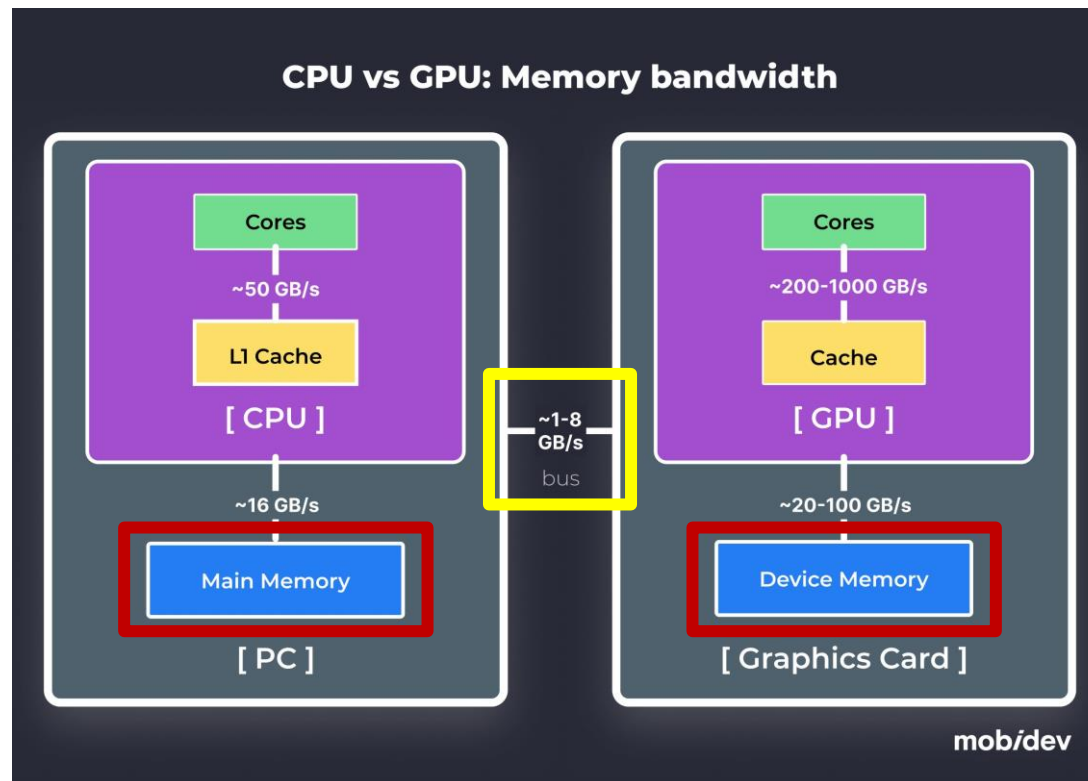
- Zvezki Jupyter vsebujejo praktične primere z razlago
- Za izvedbo zvezkov Jupyter lahko uporabite platformo Google Colab  [Open in Colab](#)

Procesne enote

- Pri delu z računalniškimi sistemi uporabljamo **centralne procesne enote (CPE)**
 - Splošno-namenska procesna enota
 - CPE z več jedri in več nitmi omogoča sočasno izvajanje operacij, kar vodi v pohitritve
- Specifične procesne enote kot **koprocesorji**
 - Te procesne enote so **namenske**
 - Primer: grafična procesna enota (GPE) je namenjena vizualizaciji
 - Različne arhitekture, ki omogočajo namensko delovanje
- Danes pogosto uporabljamo **CPE in GPE**
 - Računalniške igre
 - CPE izvaja logiko igre
 - GPE izvaja vizualizacijo igre
 - Umetna inteligenca
 - CPE izvaja koordinacijo učenja modela,
 - GPE izvaja zahtevne računske operacije med učenjem modela

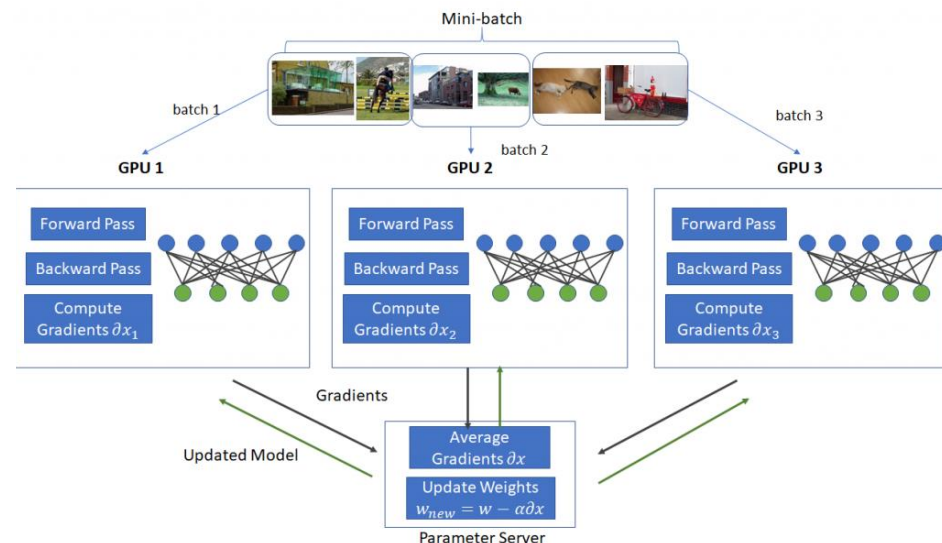
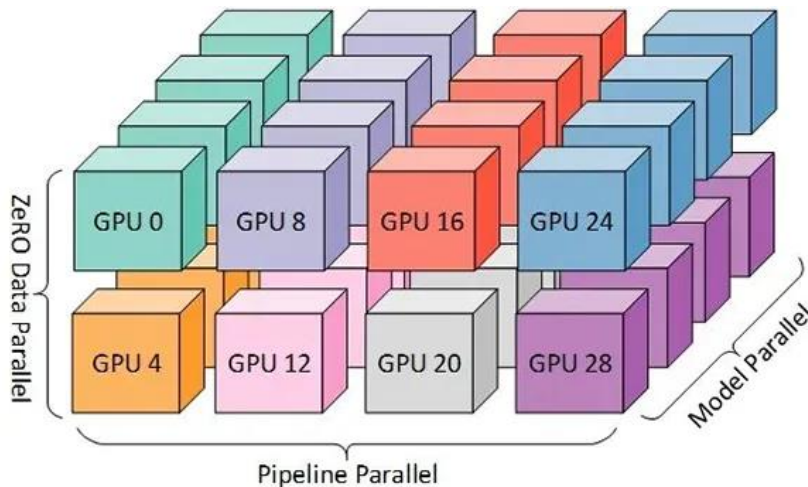
Računanje na CPE in GPE

- Izvedba računskih operacij na CPE in GPE zahteva posebno pozornost
 - Ko se računska operacija izvaja na CPE je rezultat v **RAM-u**
 - Ko se računska operacija izvaja na GPE je rezultat v **VRAM-u**
 - Neoptimalni delovni tok računskih operacij na različnih procesnih enotah vodi v **zakasnitve**



Strojno pospeševanje

- Včasih lahko celotno breme obdelave prenesemo na specifično procesno enoto
 - CPE je v vlogi **upravljalca**, specifična procesna enota je v vlogi **izvajalca**
- Izkaže se, da so GPE zelo uporabne pri izvedbi **vektorskih** in **matričnih** operacij
 - To je uporabno za hitrejšo izvedbo različnih algoritmov
 - Algoritmi strojnega učenja, algoritmi nad grafi, vizualizacijski algoritmi
 - Paralelna obdelava podatkov (ang. data parallelism)
 - Paralelno računanje (ang. task parallelism)



Strojno pospeševanje

- Za strojno pospeševanje obstaja veliko orodij
- Sočasna uporaba **več CPE** na različnih vozliščih
 - MPI (ang. Message Passing Interface)
 - OpenMP (ang. Open Multi-Processing)
- Sočasna uporaba **CPE in GPE** ali **več GPE**
 - CUDA (ang. Compute Unified Device Architecture) – izključno NVIDIA GPE
 - ROCm - primarno za AMD GPE, deluje tudi na NVIDIA GPE
 - OpenACC (ang. Open Accelerators)
 - OpenCL (ang. Open Computing language)
- V programih lahko tudi kombiniramo zgornje rešitve
 - Ponavadi uporabljamo knjižnice ali ogrodja, ki jih uporabljajo
 - NVIDIA RAPIDS, PyTorch Distributed Data Parallel, Apache Spark



NVIDIA.
CUDA®



OpenACC
More Science, Less Programming



Strojno pospeševanje

- Strojno pospeševanje lahko **skaliramo**
 - Sistem z več CPE in GPE (računsko vozlišče, ang. compute node)
 - Sistem z več računskimi vozlišči (računska gruča, ang. compute cluster)
 - Sistem z več računskimi gručami (razpršeno računalništvo, ang. grid computing)
 - Visokozmogljivo računalništvo in superračunalništvo

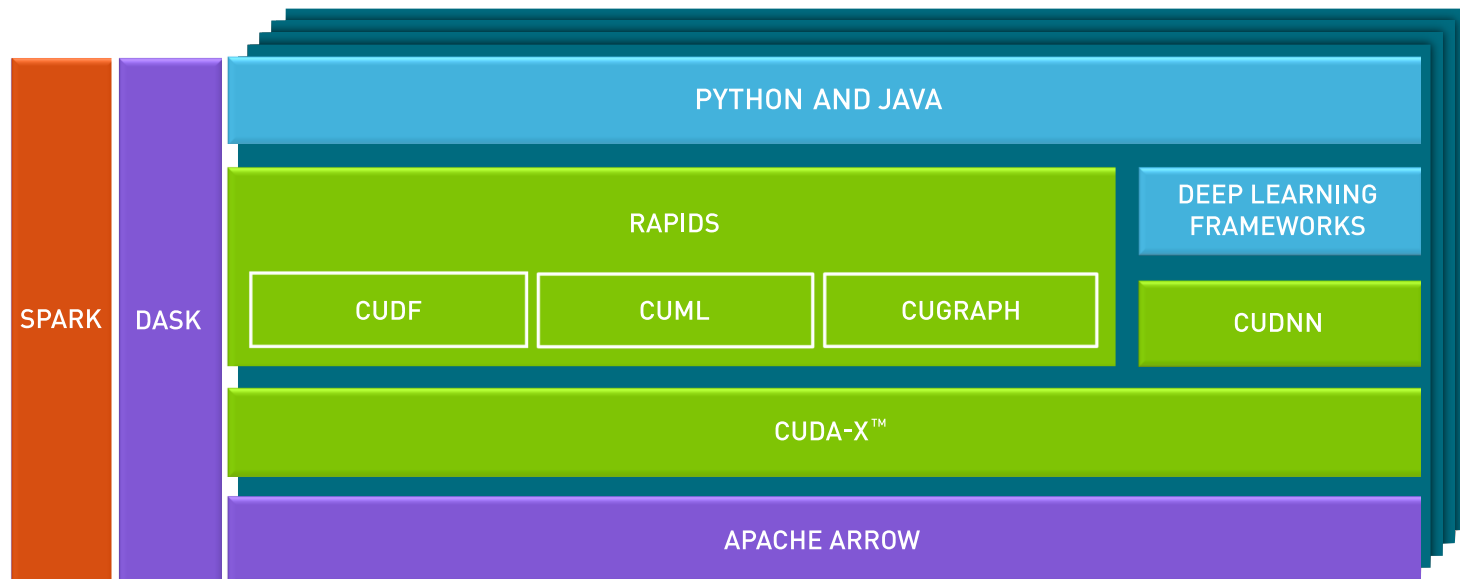
- Skaliranje zahteva ustrezno **komunikacijo** med strojno opremo
 - Skaliranje na več CPE (OpenMP in MPI – usklajevanje implementiramo sami v programu)
 - Skaliranje na več CPE+GPE (**NVIDIA RAPIDS**, PyTorch Distributed Data Parallel)

- Ogrodja kot je NVIDIA RAPIDS nam bistveno olajšajo strojno pospeševanje
 - Nabor uporabnih knjižnic
 - Enostaven programski vmesnik za implementacijo strojnega pospeševanja

Ogrodje NVIDIA RAPIDS

- **Odprtokodno ogrodje** za strojno pospeševanje na **GPE**
- Nabor uporabnih knjižnic (cuDF, cuML, cuGraph, cuxfilter, ...)
- Kompatibilnost z drugimi rešitvami (CUDA, Apache Spark in Arrow, Dask)
- Programski jezik Python

Machine Learning to Deep Learning: All on GPU





Primerjava delovnih tokov

Hadoop Processing, Reading from Disk



Spark In-Memory Processing



25-100x Improvement
Less Code
Language Flexible
Primarily In-Memory

Traditional GPU Processing



5-10x Improvement
More Code
Language Rigid
Substantially on GPU

RAPIDS



50-100x Improvement
Same Code
Language Flexible
Primarily on GPU

Knjižnice NVIDIA RAPIDS

■ cuDF

- Delo s podatki in velepodatki na GPE, Pandas-like API, pospešeno branje in operacije I/O (cuIO)
- Podpora za datotečni format Parquet
- Integracija z Dask (dask-cudf) za skaliranje

■ cuML

- Izvedba algoritmov strojnega učenja na GPE, scikit-learn-like API
- Širok nabor algoritmov (logistična regresija, Random Forest, K-Means, PCA, UMAP, DBSCAN, ...)
- Integracija z Dask (dask-cuml) za skaliranje

■ cuGraph

- Izvedba algoritmov nad grafi na GPE, NetworkX-like API
- Širok nabor algoritmov (PageRank, MST, BFS, HITS, ...)

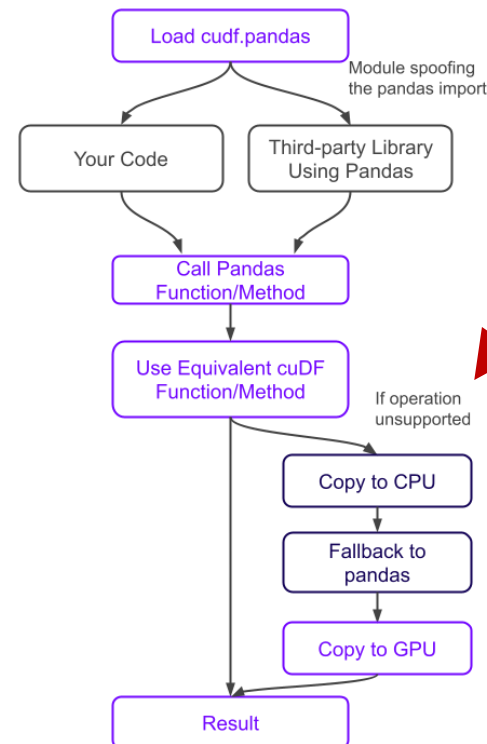
Knjižnice NVIDIA RAPIDS

■ Druge knjižnice NVIDIA RAPIDS

- RMM (Rapids Memory Manager) – delo s pomnilnikom na več GPE
- RAFT (Reusable Accelerated Function Templates) – optimizirane funkcije za računanje
- cuIO – delo z operacijami I/O, branje in nalaganje v VRAM
- cuSpatial – delo z geoprostorskimi podatki, podpora za GIS
- cuSignal – delo s signali na GPE (FFT, filtri, konvolucija), SciPy.signal-like API
- cuStrings – delo z nizi znakov (regularni izrazi, tokenizacija, iskanje podnizov), sočasna obdelava
- cuXfilter – vizualizacija velikih podatkovnih zbirk (dashboard)
- NVTabular – pridobivanje značilk in predobdelava podatkov za priporočilne sisteme

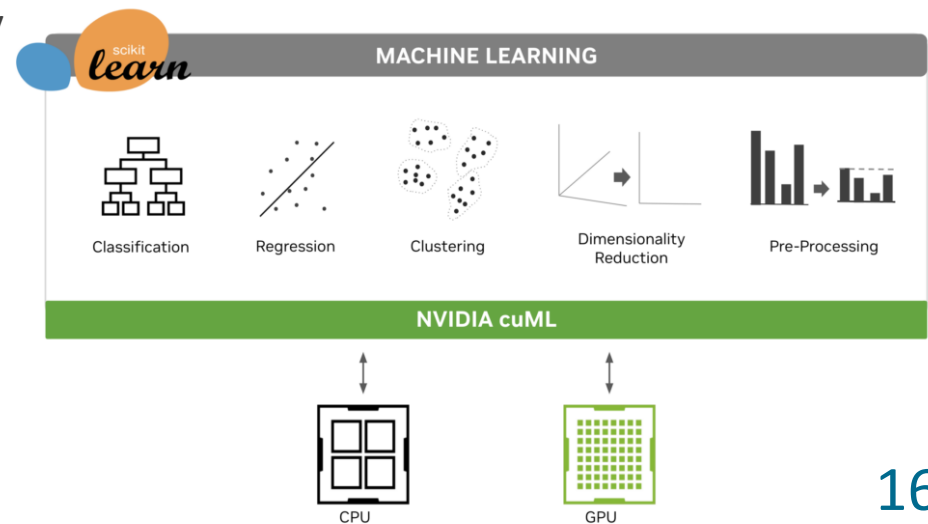
Knjižnica cuDF

- Pospeševanje nalaganja podatkov za nadaljnjo obdelavo na GPE
- Integracija s knjižnico **Pandas** (Python)
 - Tudi s knjižnico **Polars** (Python) **[beta]**
- Pospeševanje Pandas lahko vključimo z **modulom cudf.pandas**
 - Jupyter Notebook: `%load_ext cudf.pandas`
 - Python: `python -m cudf.pandas program.py`
- Pozorni moramo biti na podprte funkcije!
 - **Spomnite se profiliranja na GPE**
(Napredna obdelava velepodatkov v Pythonu)



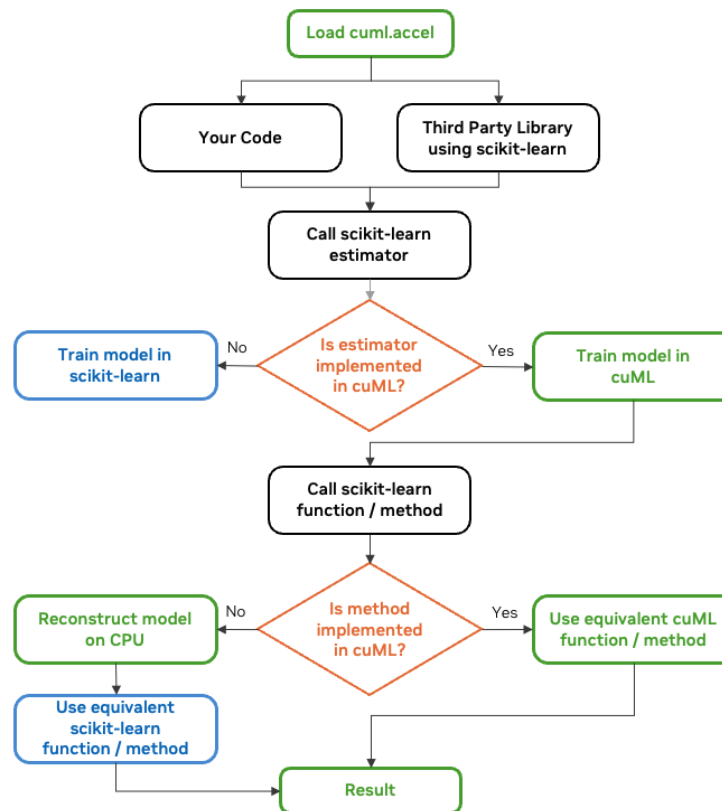
Knjižnica cuML

- Pospeševanje algoritmov strojnega učenja na GPE je smiselno, kadar imamo **velike količine podatkov** nad katerimi je potrebno izvesti **računske operacije**
- cuML je "drop-in replacement" za scikit-learn (Python)
 - **Prototipiranje** na CPE z manjšo količino podatkov in scikit-learn
 - **Produksijska izvedba** na GPE z večjo količino podatkov in cuML
 - Programska koda pri tem ostane praktično identična
- Pospeševanje scikit-learn lahko vključimo tudi z **modulom cuml.accel**
 - Jupyter Notebook: `%load_ext cuml.accel`
 - Python: `python -m cuml.accel program.py`



Knjižnica cuML

- Pospeševanje scikit-learn s `cuml.accel` poteka [avtomatsko](#)
- Samodejno preklapljanje med CPE in GPE
 - Optimiziran prenos podatkov iz RAM v VRAM in nazaj

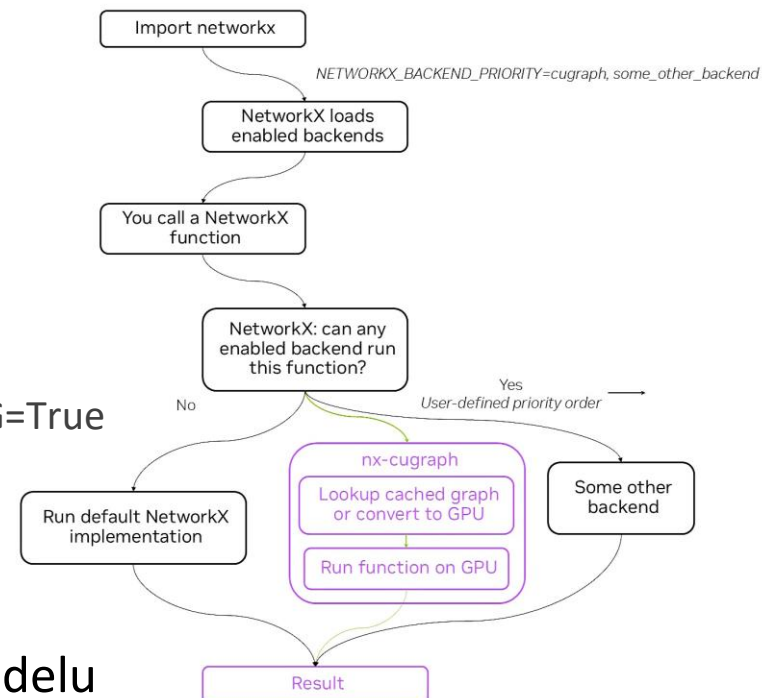


Knjižnica cuML

- Podprti algoritmi strojnega učenja
 - Algoritmi za regresijo in klasifikacijo (linearna regresija, Random Forest, naivni Bayes)
 - Algoritmi za gručenje (K-Means, DBSCAN)
 - Postopki za redukcijo dimenzij (PCA, t-SNE, UMAP)
 - Algoritmi, ki upoštevajo sosedstvo (k-NN)
 - Algoritmi za časovne vrste (Holt-Winters, ARIMA)
 - Algoritmi razložljive umetne inteligence (SHAP)
- [Postopki predobdelave podatkov, izračun metrik in druge uporabne operacije](#)
- Vse podprte algoritme lahko preverite v [dokumentaciji](#)
- Primerjava knjižnice scikit-learn in cuML v praktičnem delu

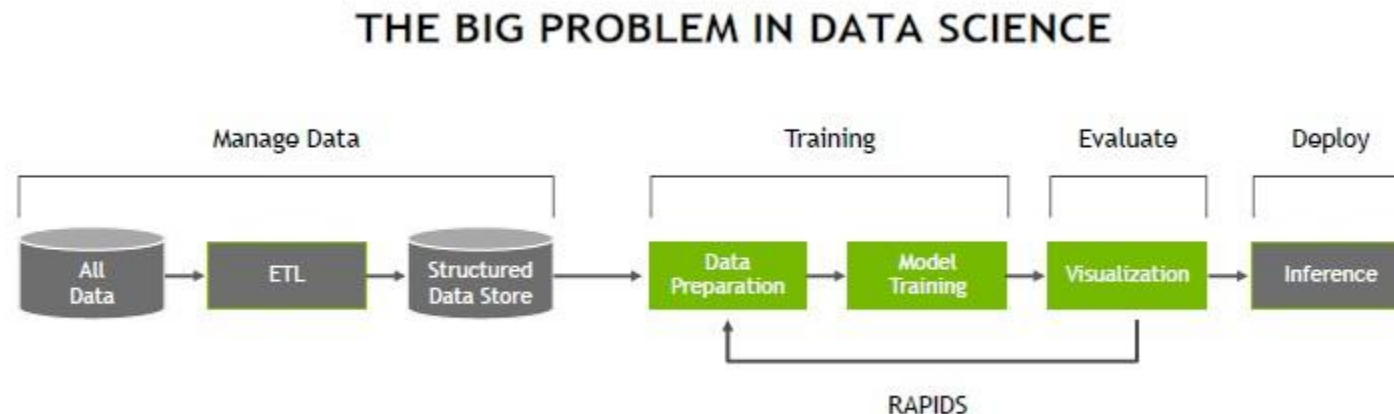
Knjižnica cuGraph

- Pospeševanje algoritmov nad grafi je še posebej zaželeno saj so grafi ponavadi ogromne podatkovne strukture
- cuGraph podpira integracijo z različnimi ogrodji za delo z grafi
 - [NetworkX](#), GSQL, TigerGraph
- V tem izobraževanju bomo uporabljali integracijo z [NetworkX](#)
- Pospeševanje omogočimo s paketom **nx-cugraph**
 - `pip install nx-cugraph-cu-12`
 `–extra-index-url https://pypi.nvidia.com`
 - Jupyter Notebook: `%env NX_CUGRAPH_AUTOCONFIG=True`
 - Python: `export NX_CUGRAPH_AUTOCONFIG=True`
 `&& python program.py`
- [Podprti algoritmi in operacije](#)
- Primerjava NetworkX in cuGraph v praktičnem delu



NVIDIA RAPIDS cevovodi

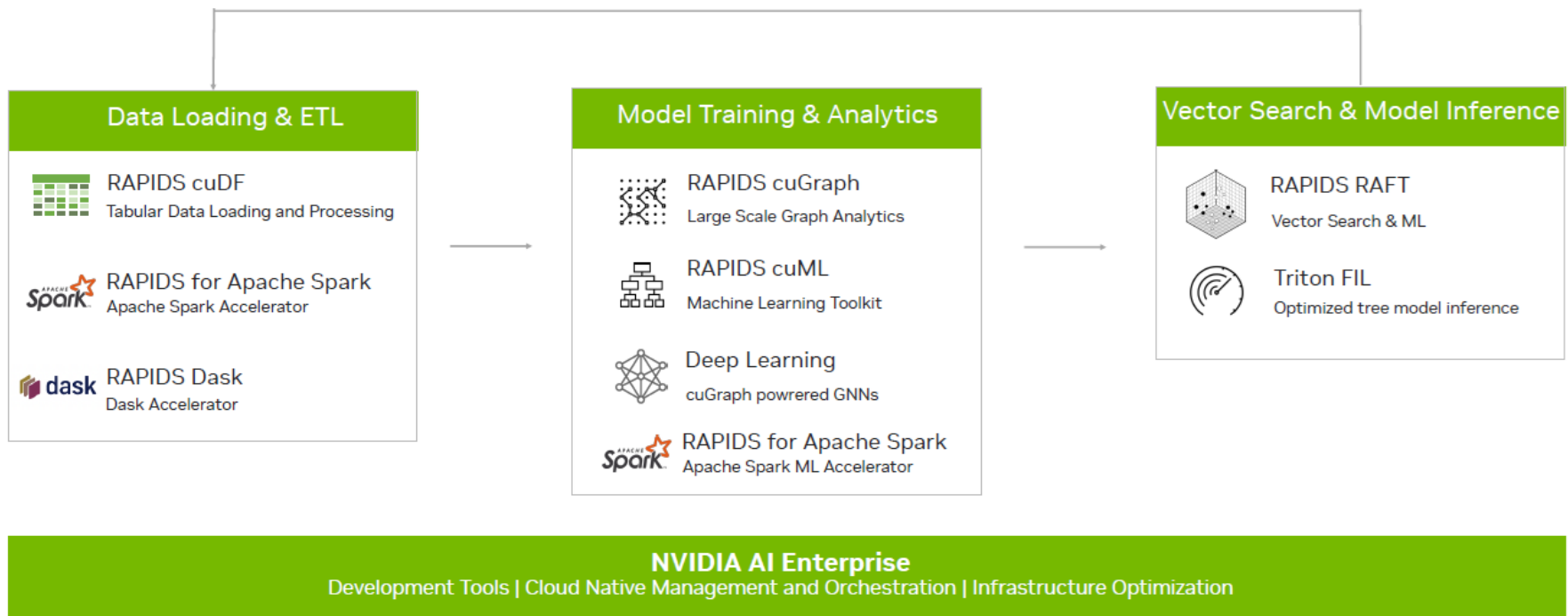
- Z NVIDIA RAPIDS lahko na GPE izvedemo vrsto **zaporednih operacij** z različnimi knjižnicami, kar vodi v **delovni tok znotraj cevovoda**
 - Optimalno naložimo podatke na GPE
 - Podatke na GPE obdelamo z različnimi knjižnicami
 - Prikažemo rezultate
- Velik del cevovoda se lahko izvaja pospešeno na GPE



NVIDIA RAPIDS cevovodi

- Ideja je spraviti celoten cevovod na GPE (**E2E – End-to-end**)

RAPIDS: Accelerating Data Science End-to-End



Praktični primeri

- V nadaljevanju si bomo pogledali izvedbo cevovodov z NVIDIA RAPIDS
- Cilj cevovodov je, da se čimveč zgodi na GPE preden se prikaže rezultat
- Pri tem bomo uporabljali knjižnice cuDF, cuML in cuGraph

- Primer 1: Priporočanje filmov (cuDF + cuGraph)
- Primer 2: Vektorsko iskanje (cuDF + cuML)

Priporočanje filmov

- Podatkovna zbirka MovieLens-25M
 - Sami lahko poskusite z večjo zbirko MovieLens-32M
- **Ideja:** priporočilni sistem, ki bo za posameznega uporabnika vrnil K najbolj smiselnih filmov
- Implementacija zajema:
 1. **Nalaganje** interakcij med uporabniki in filmi (ocene)
 2. Izvedba **algoritma**, ki omogoča priporočanje
 3. **Vračanje** priporočil
- Koraka 1 in 2 se lahko izvajata ločeno od koraka 3
 - Posodobitev se zgodi vsako noč
- Korak 3 mora biti izveden v časovnem obsegu največ do 3 sekunde

Priporočanje filmov

- Imamo ogromno količino podatkov o interakcijah med uporabniki in filmi
 - Gre za **velepodatke**, saj se vsak dan zelo hitro ustvarjajo novi podatki
 - Različni uporabniki dnevno podajajo nove ocene filmov
 - Koraki implementacije **ne smejo trajati predolgo**
 - Korak 1 (nalaganje) in 2 (algoritem) lahko zaženemo čez noč
 - Korak 3 (vračanje) mora biti izredno hiter, da zagotovimo dobro uporabniško izkušnjo
- Pri implementaciji bomo pri koraku 2 (algoritem) uporabljali algoritma **PageRank** in **HITS**
- Izvedba na CPE bo problematična => poskusimo z GPE
- **Ideja**: naložimo podatke na GPE (cuDF), nato jih tam obdelajmo (cuGraph)
 - Rezultate algoritma lahko pustimo v GPE in nato izvajamo korak 3 na GPE in CPE

Priporočanje filmov

- Izvedba algoritmov PageRank in HITS na GPE
 - cuDF + cuGraph
 - `cg.pagerank(G, personalization=[posebitveni vektor uporabnika])`
 - `cg.hits(G, max_iter=[št. iteracij])`
- Izvedba priporočanja z algoritmom PageRank na CPE
 - rezultat algoritmov PageRank in HITS prenesemo iz GPE na CPE in tvorimo seznam K filmov
- Kaj smo naredili?
 - [GPE] Prebrali smo velepodatke (VRAM)
 - [GPE] Izvedli smo algoritem PageRank/HITS
 - [GPE+CPE] Prenesli smo rezultate algoritma PageRank/HITS v RAM
 - [CPE] Pripravili smo seznam priporočil s K filmi
 - [GPE] Seznam priporočil smo shranili v VRAM
- Zakaj smo šli iz GPE na CPE? Zakaj smo shranili seznam priporočil na GPE?

Priporočanje filmov

■ Zakaj smo šli iz GPE na CPE?

- .itertuples() v cuDF še **ni podprta funkcija** in jo moramo izvesti na CPE
- Alternativa je **drugačen pristop**, ki ne uporablja .itertuples()

■ Zakaj smo shranili seznam K priporočil na GPE?

- Demonstriranje NVIDIA End-to-end cevovoda (vse oz. čim več se zgodi na GPE)
- V praksi bi lahko izvajali **predpomnenje** rezultatov na GPE
- V praksi bi lahko **nadaljevali delovanje cevovoda** z dodatnimi funkcionalnostm
- Naknadna obdelava priporočil, **hibridni priporočilni sistemi**

Vektorsko iskanje

- Podatkovna zbirka Coronavirus (COVID-19) Tweets
 - 8 mio tvitov na temo COVID-19 (april 2020)
 - Več datotek CSV (po dnevih)
- **Ideja:** narediti iskalnik po tvitih, ki uporablja vektorski prostor za ugotavljanje podobnih besedil (semantično iskanje)
- Implementacija zajema:
 1. **Nalaganje** podatkov (tvitov)
 2. Izvedba **ustvarjanja iskalnega indeksa** (vektorskega prostora)
 3. Implementacija iskanja s **funkcijo podobnosti** (kosinusna razdalja)
- Koraka 1 in 2 se lahko izvajata ločeno od koraka 3
 - Posodobitev vsako noč (če bi imeli vsak dan nove tvite)
- Korak 3 mora biti izveden v časovnem obsegu največ do 3 sekunde

Vektorsko iskanje

- Imamo ogromno količino tвитov, ki jih je potrebno obdelati
 - Gre za velepodatke, saj lahko tвiti nastajajo dnevno
 - Koraki implementacije **ne smejo trajati predolgo**
 - Korak 1 (nalaganje) in 2 (iskalni indeks) lahko zaženemo čez noč
 - Korak 3 (iskanje) mora biti izredno hiter, da zagotovimo dobro uporabniško izkušnjo
- Pri implementaciji bomo pri koraku 2 (iskalni indeks) uporabljali utežno shemo TF-IDF (ang. term frequency, inverse document frequency)
- Izvedba na CPE bo problematična => poskusimo z GPE
- **Ideja:** naložimo podatke na GPE (cuDF), nato jih tam obdelajmo (cuML)
 - Rezultate algoritma lahko pustimo v GPE in nato izvajamo korak 3 na GPE in CPE

Vektorsko iskanje

- Izvedba ustvarjanja vektorskega indeksa na GPE
 - [TF-IDF](#) pretvori besede v besedilu v večdimenzionalne vektorje z utežmi
 - cuML omogoča izvedbo TfidfVectorizer, ki ga lahko nastavljamo s parametri
 - min_df (minimum document frequency) – kolikokrat se mora beseda pojaviti v vseh tvitih
 - max_features (maksimalno število značilk) – določa dimenzionalnost vektorja
- Ustvarjanje vektorskega indeksa je operacija, ki jo lahko bistveno pohitrimo z izvedbo na GPE
- **Problem:** imamo visokodimenzionalni vektorski prostor (več tisoč dimezij), kjer se lahko zgodi, da bo večina komponent vektorjev enaka 0
 - beseda se ne pojavi v besedilu
- S čim imamo opravka? Na kakšen način lahko rešimo ta problem?

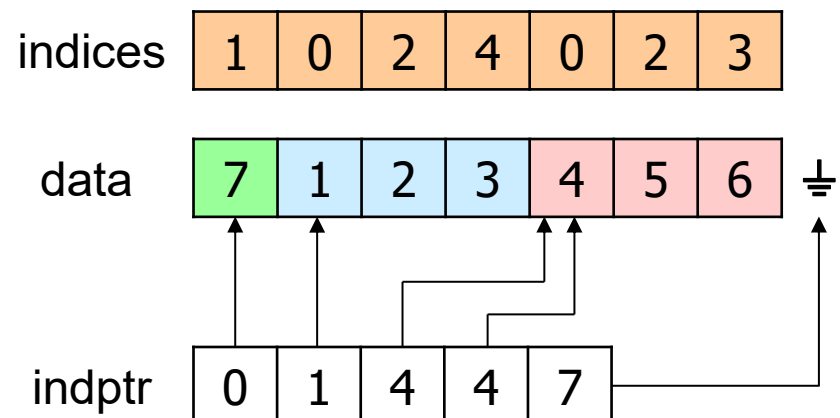
Vektorsko iskanje

■ S čim imamo opravka?

- Gre za **redko matriko**! (ang. sparse matrix)
- Redke matrike lahko hranimo na bolj **učinkovit** in **stisnjen** način
- **CSR** (compressed sparse row) ali **CSC** (compressed sparse column)
- `cuml.common.sparsefuncs`

■ Primer CSR:

	0	1	2	3	4
0	0	7	0	0	0
1	1	0	2	0	3
2	0	0	0	0	0
3	4	0	5	6	0

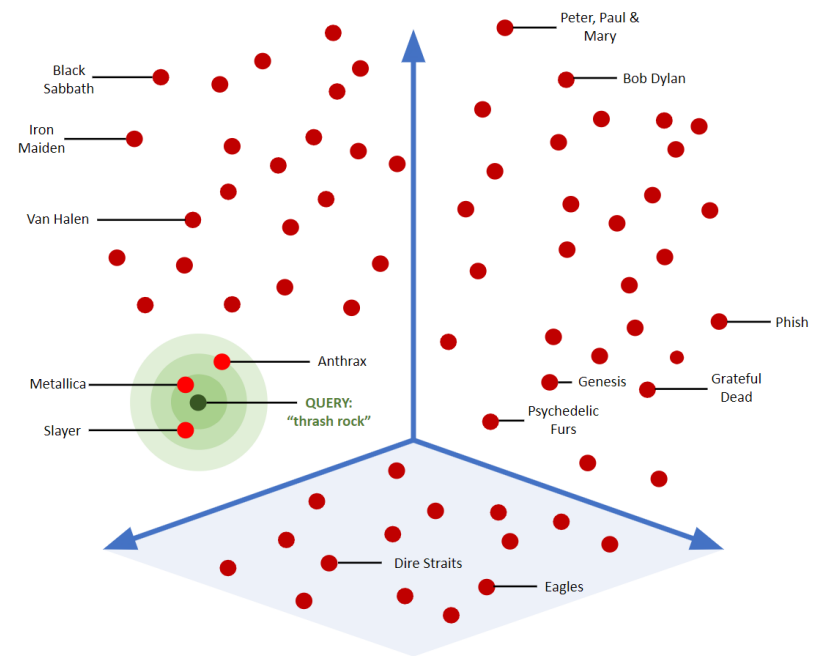
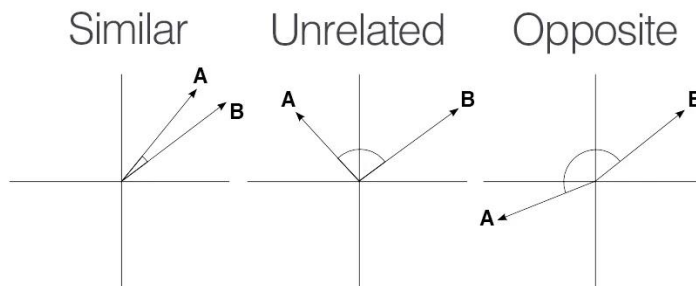


shape = (4, 5)

S CSR shranimo samo 35% matrike

Vektorsko iskanje

- Naš iskalni indeks lahko predstavimo kot redko matriko v obliki CSR
 - Zasedamo manj pomnilnika
 - Omogočimo hitre operacije nad vrsticami
- Imeli bomo visokodimenzionalni vektorski prostor (~30.000 dimenzij)
- Funkcija podobnosti bo kosinusna razdalja

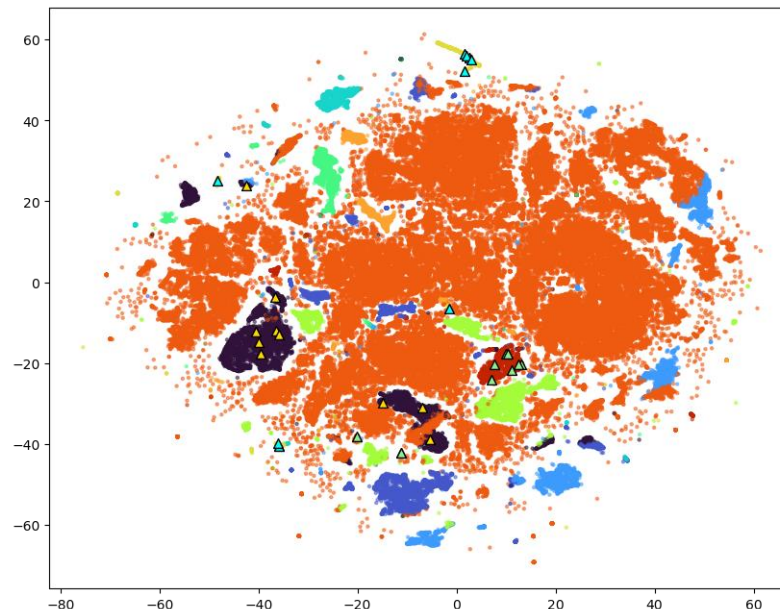


Vektorsko iskanje

- Ker je iskalni indeks v formatu CSR bomo morali ob izračunu podobnosti s kosinusno razdaljo izvesti pretvorbo vhodnega niza v format CSR
 - funkcija `csr_row_normalize_l2` => izvaja se namenski ščepec CUDA
 - Normalizacija L2 (Evklidska norma) => vrednosti matrike CSR bodo največ 1
 - Na ta način ohranimo izračun kosinusnih razdalj v formatu CSR ob poljubnem vhodnem nizu!
- Izračun podobnosti je skalarni produkt med indeksom in vhodnim nizom
- Vektorska operacija => dobimo podobnosti vhoda z vsemi elementi matrike CSR
 - Po domače: vhod smo primerjali z vsemi indeksiranimi tviti
- Na koncu izberemo K zadetkov in jih prikažemo uporabniku

Vektorsko iskanje

- Dodatno lahko izvedemo analizo tvitov z gručenjem (KMeans)
 - Zaradi strojnih omejitev vzemimo vzorec 100k tvitov (od 8 mio)
 - Podatki so še vedno na GPE!
- Z gručenjem lahko ugotovimo ali so tviti vezani na **specifične tematike**
 - Praktični primer prikazuje gručenje na 10 gruč
 - KMeans zahteva število gruč kot parameter; sami poskusite s kakšnim drugim algoritmom, kjer to ni zahteva
- Vizualizacija gručenja s t-SNE
 - Preslikava iz n-D v 2D
 - Podatki so še vedno na GPE!



Vektorsko iskanje

■ Kaj smo naredili?

- [GPE] Prebrali smo velepodatke (VRAM)
- [GPE] Ustvarili smo iskalni indeks (TfidfVectorizer)
- [GPE] Izvedli smo iskanje (izračun kosinusnih razdalj)
- [GPE] Pripravili smo seznam K zadetkov
- [GPE] Seznam priporočil smo shranili v VRAM

dodatno

-
- [GPE] Vzorec iskalnega indeksa smo obdelali z gručenjem (KMeans)
 - [GPE] Gruče smo vizualizirali (t-SNE)

Zaključek

- Spoznali ste **ogrodje NVIDIA RAPIDS**
 - Pospeševanje algoritmov na GPE
 - Primerjava izvajanja algoritmov na CPE in GPE
 - E2E cevovodi kot celovite pospešene aplikacije

- Uporaba različne strojne opreme
 - Poznavanje prednosti in omejitev
 - Inženirska iznajdljivost
 - Različni tipi koprocesorjev s poudarkom na GPE

- Samostojno predelajte še ostale praktične primere v zvezkih Jupyter

Vabljeni še na druga izobraževanja

- Uvod v obdelavo velepodatkov v Pythonu (12.-15. 05. 2025)



-
- Napredna obdelava velepodatkov v Pythonu (02.-04. 06. 2025)



-
- Uvod v ogrodje NVIDIA RAPIDS (16.-18. 06. 2025)



-
- V delu so nova izobraževanja!

- Programiranje s CUDA v Pythonu
- Porazdeljeno učenje modelov umetne inteligence
- Napredna uporaba ogrodja NVIDIA RAPIDS

Hvala za udeležbo!