1. **Didžiųjų duomenų teorijos ir su ja susietų sistemų pagrindinės sąvokos: pagrindiniai  
   kriterijai, apibūdinantys duomenis kaip didžiuosius, paskirstytieji skaičiavimai ir  
   sistemos bei su jomis susietos sąvokos (mazgas, klasteris ir t.t.).**

***Duomenys*** *– realūs faktai. Duomenys nėra apdoroti žmogaus suvokimui.*

***Informacija*** *laikomi kažkokiu būdu apdoroti, sutvarkyti duomenys. Tai yra faktų matavimų rinkinys, padedantis atsakyti kas, kur, kada, kiek.*

***Didieji duomenys*** *– tai bet kuris duomenų rinkinys, kuris yra pernelyg didelis, kad jį būtų galima apdoroti tradicinėmis priemonėmis (pavyzdžiui, reliacinėmis duomenų bazėmis (RDB)).*

*„Didžiųjų duomenų mokslas yra labiau techninis, nusakantis kaip reikia apdoroti duomenis. Duomenų mokslas daugiau kalba apie duomenų analizę“.*

*Didieji duomenys charakterizuojami šitais parametrais: 1)* ***apimtis*** *– kiek duomenų įeina į rinkinį, 2)* ***įvairovė*** *– kaip stipriai skiriasi skirtingų tipų duomenys rinkinyje, 3)* ***greitis*** *– kaip greitai generuojami nauji duomenys, 4)* ***teisingumas*** *– kiek tikslūs arba teisingi duomenys.*

***Paskirstytųjų išteklių sistema*** *- sistema su tarpusavyje sujungtais mazgais, kurie vykdo bendrą algoritmą su vienu įvesties duomenų rinkiniu. Tokių sistemų skaičiavimai vadinami* ***paskirstytaisiais skaičiavimais*** *(išteklių vienetai yra fiziškai galimai nutolę vienas nuo kito).*

***Mazgai*** *– kompiuteriai arba kitokie potencialiai savarankiški skaičiavimų vienetai.*

***Klasteris*** *– vienoje sistemoje veikiantis mazgų rinkinys.*

1. **Didiesiems duomenims skirti programavimo modeliai ir juos atitinkančios sistemos,**  
   **jų (modelių ir sistemų) veikimo principai, skiriamieji bruožai, klasifikavimo kriterijai.**

***Klasifikavimo kriterijai****: 1) abstrakcijos lygmuo – galimybė paslėpti žemo lygio detales nuo programuotojo, 2) lygiagretumo rūšis aprašo kokiu būdu programavimo modelis arba sistema išreiškia lygiagrečias operacijas ir kokiu būdu vykdomoji aplinka palaiko vienu metu vykdomas operacijas keliuose mazguose, 3) infrastruktūros mastas priklauso nuo programavimo sistemos galimybių efektyviai vykdyti instrukcijas su duotuoju infrastruktūros dydžiu (number of computing nodes), 4) taikymo klasės apibūdina programavimo sistemos pagrindinę paskirtį.*

***Programavimo modeliai (ir sistemos):***

* *MapReduce (Apache Hadoop)*
* *Directed Acyclic Graphs – DAG (Apache Spark, Apache Storm, Apache Flink, Azure Machine Learning)*
* *Message Passing – MPI*
* *Bulk Synchronous Parallel – BSP (Apache Giraph, Apache Hama)*
* *Workflows (Swift, COMPSs, DMCF)*
* *SQL-like (Apache Pig, Apache Hive)*

***MapReduce*** *programavimo modelis, sukurtas didelės apimties skaičiavimams realizuoti. Funkcija map apdoroja duomenų poras (key, value) ir grąžina porų sąrašą. Funkcija reduce apjungia visas tarpines raktų reikšmes ir nebegrąžina didžiųjų duomenų – grąžinami paprasti duomenys, kuriuos galima atvaizduoti ir su kuriais galima dirbti.*

***Apache Hadoop*** *atlaisvina programuotoją nuo klasikinių paskirstytųjų skaičiavimų problemų, tokių kaip krūvio balansavimas, atsparumo gedimams suteikimas, duomenų siuntimas.*

***Directed Acyclic Graph*** *– efektyvi paradigma (taip pat – programavimo modelis) sudėtingiems duomenų analizės procesams aprašyti, kur grafo viršūnės atitinka vykdomas užduotis, o lankai – priklausomybes tarp užduočių;*

***Apache Spark*** *apibrėžiama kaip aibė etapų su tarpusavyje nepriklausomomis užduotimis, leidžianti pasiekti geresnį duomenų apdorojimo greitį kai kurioms taikymo klasėms.*

***Message Passing*** *yra paradigma, kuria remiantis programuotojui suteikiami baziniai duomenų mechanizmai, leidžiantys mazgams komunikuoti paskirstytųjų išteklių sistemoje. Paprastai realizuojama dviem bazinėmis duomenų siuntimo operacijomis: 1) send(gavėjas,pranešimas), 2) receive(siuntėjas,pranešimas).*

***Bulk Synchronous Parallel*** *– masinis sinchroninis lygiagretusis modelis, numatantis visų skaičiavimų padalijimą į globalius žingsnius, kurių kiekviename atliekami: 1) nepriklausomi tarpusavyje lygiagretieji skaičiavimai (kiekvienas skaičiavimo vienetas atlieka skaičiavimus su savo lokaliais duomenimis), 2) globali komunikacija (procesai keičiasi tarpusavyje duomenimis), 3) slenkstinė sinchronizacija (kiekvienas procesas laukia visų likusių procesų).*

***Apache Giraph*** *– iteratyvi grafų apdorojimo sistema, skirta aukšto išplečiamumo aplikacijoms kurti. Jos globalieji žingsniai realizuoti kaip aibė skaičiavimų, atliekamų tik per map darbus.*

***Workflow*** *– vienareikšmiškai apibrėžtas, su galimais užduočių pasikartojimais, šablonas arba sistematizuota veiklų organizacija, skirta tam tikrai duomenų transformacijai atlikti. Šis modelis suteikia galimybę deklaruoti aukšto lygmens logiką skirtingoms taikymo rūšims, paslepiant žemo lygio detales.*

***Swift*** *– lygiagrečioji skriptinė kalba, gebanti paleisti workflow tipo skaičiavimus vienu metu keliose paskirstytose sistemose, tokiose, kaip klasteriai, debesys, superkompiuteriai.*

***NoSQL*** *duomenų bazės dažnai sprendžia tokias problemas, kaip didžiųjų duomenų saugojimas ir valdymas, tačiau dažnai yra netinkamos duomenų analizei atlikti.*

***Apache Pig*** *– atvirojo kodo sistema, skirta duomenų srautų aplikacijų paleidimui naudojant Hadoop. Čia kiekviena operacija nurodo kaip duomenys turi būti apdorojami ir aprašoma į SQL panašia sintakse.*

1. **MapReduce savybės, nulemiančios šio modelio efektyvumą esant tam tikroms**  
   **sąlygoms.**

*MapReduce savybės:*

* ***Horizontalus plėtimasis***
* *Daroma prielaida, kad* ***sutrikimai yra įprastas reiškinys****: jei įvyksta vienas gedimas per 10 darbo dienų, 20000 mazgų sistemoje per dieną įvyks 20 gedimų.*
* ***Nuoseklusis duomenų nuskaitymas*** *– nuskaitymas iš kietųjų diskų yra greitesnis už atsitiktinę duomenų prieigą.*
* ***Programų judėjimas prie duomenų*** *– sistema stebi duomenų lokalumą.*
* ***Sistemos lygmens detalės paslėptos nuo programuotojo***
* ***Sklandus išplėčiamumas*** *(duomenų išplėčiamumas ir išteklių išplėčiamumas –* ***tiesinis*** *išplėčiamumas)*
* *Duomenų apimtys yra* ***petabaitai***
* *Vyksta* ***vienkartinis įrašymas*** *ir* ***daug nuskaitymų***
* *Duomenys yra interpretuojami* ***apdorojimo metu***
* ***Aukštas*** *duomenų persidengimo lygis (nenormalizuoti duomenys)*

1. **Hadoop MapReduce apdorojimo eiga.**

*Hadoop MapReduce apdorojimo eiga:*

1. *Darbas padalinamas į dviejų rūšių užduotis: map ir reduce.*
2. *Užduotys valdomos išteklių valdymo sistemos YARN, kuri sudaro tvarkaraštį ir paleidžia užduotis skirtinguose mazguose. Nesėkmingai paleistos užduotys automatiškai perskirstomos kituose mazguose.*
3. *Hadoop skaido MapRecuce įvestį į split‘us ir sukuria po map užduotį kiekvienam iš split‘ų.*
4. *Vykdoma duomenų lokalumo optimizacija (užduotys nėra paleidžiamos mazguose, kuriuose nėra duomenų šioms užduotims vykdyti).*
5. *Map užduotys įrašo savo išvestis į lokaliuosius kietuosius diskus.*
6. *Kai reduce užduočių yra kelios, kiekviena map užduotis skaiddo savo išvestį ją rūšiuodama pagal raktų reikšmes.*
7. *Skaidymai atitinka skirtingas reduce užduotis – įvyksta duomenų perdavimas reduce užduotims (shuffle).*
8. *Reduce pagal jai skirtas instrukcijas apdoroja gautą įvestį ir gaunama galutinė išvestis, kuri įrašoma atgal į HDFS sistemą.*
9. **MapReduce Kombinatorius.**

*Combine turi pasižymėti asociatyvumu ir komutatyvumu, jo įvestis yra map išvestis, o išvestis – reduce įvestis. Combine gali būti pritaikyta visiems duomenims arba tik daliai, combiner kodas gali būti nepaleistas iš viso – tai nusprendžiama atsižvelgiant į skaičiavimų vykdymo dinamiką.*

1. **GFS ir HDFS sistemos ir su jomis susietos sąvokos. MapReduce Hadoop taikymas,**  
   **pseudokodas.**

***GFS*** *(google file system) sukurta atsižvelgiant į: 1) žemo patikimumo „geležies“ naudojimo palaikymas (dažni gedimai), 2) failai yra dideli, 3) yra dvi duomenų nuskaitymo rūšys: dideli nuoseklieji nuskaitymai (reikia nuskaityti visą failą) ir smulkūs atsitiktiniai nuskaitymai (iš failo reikia nedidelės dalies), todėl daroma prielaida, kad vyrauja didelės apimties nuoseklieji įrašymai, kurie pildo failus jų neperrašant iš naujo, 4) failų operacijų laikas yra ribojamas duomenų pralaidumu. GFS netinka, jei: a) yra mažo uždelsimo duomenų nuskaitymo poreikis, b) daug smulkių failų, c) yra nuolatinis duomenų kitimas.*

***Chunk*** *serveriai saugo duomenis, o* ***master*** *– valdo chunk serverių darbą. Tarp šių serverių siunčiami heartbeat pranešimai, nustatantys ar chunk serveris veikia korektiškai bei chunk serverių failų informaciją.* ***Master*** *serveris vykdo komunikaciją su klientu siekiant duomenis nuskaityti/įrašyti.*

***HDFS*** *failai yra suskaidomi į bloko dydžio dalis, kurios saugomos kaip savarankiški vienetai. Čia vienas failas yra mažesnio dydžio nei vienas blokas HDFS sistemoje ir neužima viso bloko atminties. HDFS privalumai: 1) galimybė turėti failus didesnius už atminties kiekį bet kuriame viename kietajame diske tinkle, 2) blokas kaip abstrakcijos vienetas yra palankesnis už failo abstrakcijos vienetą, nes tai supaprastina paskirstytosios sistemos duomenų saugojimą (nereikia saugoti jokių metaduomenų), 3) blokai gerai tinka gedimams valdyti.*

*HDFS turi dviejų tipų mazgus, kurie operuoja šeimininko-darbininkų principu: vardų mazgas (NameNode) ir keletas duomenų mazgų (DataNodes). Paprastai duomenų mazgas nuskaito duomenis iš disko, o vardų mazgas laiko nuorodas į visus failus ir blokus atmintyje.*

1. **Spark palyginimas su kitomis didžiųjų duomenų apdorojimo sistemomis.**

*Apache Spark yra skaičiavimo sistema klasteriuose, skirta didelio mąsto duomenų apdorojimui. Spark nenaudoja MapReduce - jis naudoja savo paskirstytą vykdymo variklį vykdant klasterio darbus. Tačiau Spark turi daug panašumų lyginant su MapReduce. Spark yra glaudžiai integruota į Hadoop: jis gali veikti YARN sistemoje ir dirba su Hadoop failų formatais ir saugykla.*

*Spark yra geriausiai žinomas dėl savo gebėjimo operatyvioje atmintyje laikyti didelių apimčių duomenų rinkinius, tame tarpe juos saugant tarp vykdomų užduočių. Tai leidžia Spark aplenkti MapReduce darbą greičio atžvilgiu (kai kuriais atvejais iki 100 kartų), nes skirtingai nuo Spark, MapReduce duomenis beveik visada įkelia iš disko. Tokį privalumą Spark įgauna dviejų duomenų apdorojimo modelių atvejais: iteraciniai algoritmai (kai funkcija taikoma duomenų rinkiniui daug kartų iki tam tikros sąlygos išpildymo) ir interaktyvi analizė (kai vartotojas realiuoju laiku vykdo iš anksto nežinomų užklausų seriją, norėdamas ištirti duomenis ir išgauti informaciją).*

*Be talpyklos atmintyje, Spark yra patraukli dėl kitų priežasčių: jos DAG variklis ir vartotojo sąsaja. Skirtingai nuo MapReduce, Spark DAG variklis gali apdoroti darbų grandinę ir paversti juos vienu darbu vartotojo požiūriu. Spark aplikacijų programavimo sąsaja (API) yra dar vienas privalumas - palaikoma daugybė funkcijų, skirtų tipinėms duomenų tvarkymo užduotims atlikti.*

1. **Spark ir DAG realizacija Spark terpėje.**

*DAG realizacija Spark:*

* *Pirmajame Spark realizacijos sluoksnyje yra interpretatorius, realizuotas Scala kalba.*
* *Spark sukuria operatorių grafą tuomet, kai yra įvedamas kodas į Spark konsolę.*
* *Kai kviečiamas veiksmas pritaikant jį tam tikrai RDD aukštame lygmenyje, Spark pateikia operatorių grafą DAG tvarkaraščių sudarytojui.*
* *Tvarkaraščio sudarytojas skaido operatorius į užduoties etapus.*
* *Kiekvienas etapas yra sudarytas iš užduoties, kuri priklauso nuo įvesties duomenų padalijimo. Tvarkaraščių sudarytojas grupuoja operatorius, pavyzdžiui, map operatorius sugrupuoja į vieną etapą.*
* *Etapai perduodami užduočių tvarkaraščių sudarytojui, jis paleidžia užduotis per klasterio tvarkytoją.*
* *Užduočių tvarkaraščių sudarytojas nieko nežino apie etapų tarpusavio priklausomybę.*
* *Darbininkai vykdo užduotis.*

1. **Spark sudedamosios dalys.**

*DAG sistemoje Spark sudaromas iš viršūnių ir briaunų, kur viršūnės atitinka specialius duomenų vienetus - tvarias paskirstytąsias duomenų aibes (resilient distributed dataset, RDD), o briaunos reprezentuoja taikomas šioms aibėms operacijas. Į DAG galima žiūrėti kaip į MapReduce modelio apibendrinimą.*

1. **Spark transformacijos ir veiksmai. Siauros ir plačios transformacijos.**

*Aukštesniajame lygmenyje mes galime taikyti dviejų rūšių transformacijas: siaurą transformaciją (map, filter) ir plačią transformaciją (pvz., reduceByKey()). Siaura transformacija, skirtingai nuo plačios, nereikalauja duomenų rūšiavimo bei grupavimo. Todėl plačios transformacijos apibrėžia etapų ribas. Siaurose nevykdomas duomenų rūšiavimas, o plačiose – taikomas. Siauros transformacijos: map, mapValues, flatMap, filter, mapPartitions, mapPartitionsWithIndex. Plačios transformacijos: cogroup, groupWith, join, leftOuterJoin, rightOuterJoin, groupByKey, reduceByKey, combineByKey, distinct, intersection, repartition, coalesce.*

1. **Spark transformacijų taikymas: Map, FlatMap, Reduce, ReduceByKey, Filter ir kt.**

*(funkcijų sąrašas:* [*https://spark.apache.org/docs/2.4.4/rdd-programming-guide.html#transformations*](https://spark.apache.org/docs/2.4.4/rdd-programming-guide.html#transformations) *)*

1. **Spark SQL ypatumai, sintaksė. DataFrame sąvoka.**

*Spark SQL naudojamas norint pasiekti struktūrizuotą ir pusiau struktūrizuotą informaciją. Tai taip pat suteikia galimybę taikyti Spark srautinius/istorinius duomenis interaktyviai analizei. Spark SQL yra Spark modulis struktūrizuotam duomenų apdorojimui su panašiu į SQL funkcionalumu. Taigi, jis veikia kaip paskirstytas SQL užklausų variklis.*

*Spark SQL savybės:*

* *Kaštais pagrįstas adaptyvusis skaičiavimų optimizavimas*
* *Atsparumas klaidoms/gedimams užklausų metu: tai pasiekiama naudojant daugelį mazgų keliavalandinėms užklausoms vykdyti naudojant Spark variklį.*
* *Pilnas suderinamumas su esamais Hive duomenimis.*
* *Duomenų struktūra DataFrames ir SQL suteikia priemones įvairiems duomenų šaltiniams pasiekti. Tokiems, kaip Hive, Avro, Parquet, ORC, JSON ir JDBC.*
* *Struktūrizuotų duomenų palaikymas Spark programose, naudojant SQL arba DataFrame API panašų į analogiškas struktūras kitose bibliotekose, tokiose kaip Pandas.*

1. **Spark Mlib: paskirtis, stipriosios pusės, skiriamieji bruožai.**

*Mlib* ***paskirtis*** *- suteikti patogią galimybę populiariausius mašininio mokymosi algoritmus naudoti lengvai plečiamoje paskirstytoje sistemoje. Kaip jau buvo minėta, Spark yra suderinta su paskirstytųjų failų sistemomis, su tokiomis kaip HDFS.*

*Mlib* ***stipriosios pusės / skiriamieji bruožai****: 1) geros išplečiamumo savybės ir greitaveika, 2) turi mašininio mokymosi bibliotekas, 3) turi tiesinės algebros paketą Breeze (skaitmeninių skaičiavimo ir mašininio mokymosi bibliotekų rinkinį), 4) DataFrame pagrįstas API.*

1. **Lygiagrečiųjų kompiuterių klasifikavimas pagal atminties prieinamumą.**

*Kompiuteriai su bendrąja atmintimi (visi procesoriai gali tiesiogiai pasiekti visas atminties vietas, kuri turi bendrą visiems adresaciją; jie skirstomi į UMA ir NUMA) ir kompiuteriai su paskirstytąja atmintimi (kiekvienas procesorius turi tik savo nuosavą atmintį ir sudaro atskirą mazgą, kurie yra sujungti tarpusavyje tinklu – čia procesorius tik adresuoja savo atmintį).*

1. **NVIDIA GPU įrenginio architektūra**
   1. **Skaičiavimų vienetai**

*GPU susidaro iš NSM multiprocesorių (toliau SM, angl. streaming multiprocessor). Vienas multiprocesorius turi Nm branduolių, sudarytų iš skaičiavimų modulių. Moduliai gali būti skirtingi - tai priklauso nuo konkrečios plokštės GPU architektūros.*

*Visi skaičiavimai ir duomenų siuntimai yra atliekami porcijomis, specialiais gijų apdorojimo vienetais, kurie operuoja gijų porcijomis (angl. warp).*

*Programuotojo užduotis - parašyti kodą taip, kad gijų porcijos veiktų efektyviai, remiantis jų veikimo principais ir NVIDIA rekomendacijomis.*

* 1. **Atminties struktūra**

*1. GPU turi pagrindinę atmintį duomenims talpinti, paprastai tai yra vieno gigabaito eilės dydžio atmintis. Iš jos skaityti ir į ją rašyti gali visi CUDA branduoliai lygiagrečiai. GPU pagrindinė atmintis išskaidyta į segmentus, jų dydis priklauso nuo GPU atminties magistralės dydžio.*

*2. Pagrindinė atmintis naudojama globaliajai ir lokaliajai atmintims saugoti. Jos skiriasi tik tuo, kad lokalioji yra rezervuota tam tikroms gijoms, o iš globaliosios gali skaityti ir GPU, ir CPU (kopijuojant).*

*3. Kiekvienas GPU multiprocesorius turi greitąją atmintį (padalintą į kelias rūšis), paprastai tai yra 10-100 kilobaitų eilės dydžio atmintis. Duomenų nuskaitymas ir įrašymas šioje atmintyje yra žymiai greitesnis, nei pagrindinės atminties.*

*4. Greitąją atmintį galima suskaidyti į šiuos tipus:*

*a. Registrai. Šiuo metu skaičiavimuose dalyvaujantiems duomenims talpinti skirta atmintis*

*b. Bendroji atmintis. Yra prieinama skirtingų vieno multiprocesoriaus gijų, tačiau vieno multiprocesoriaus nėra prieinama kito multiprocesoriaus skaičiavimų moduliams.*

*c. L1 cache atmintis - automatizuota bendrosios atminties versija, kurios panaudojimu rūpinasi kompiliatorius.*

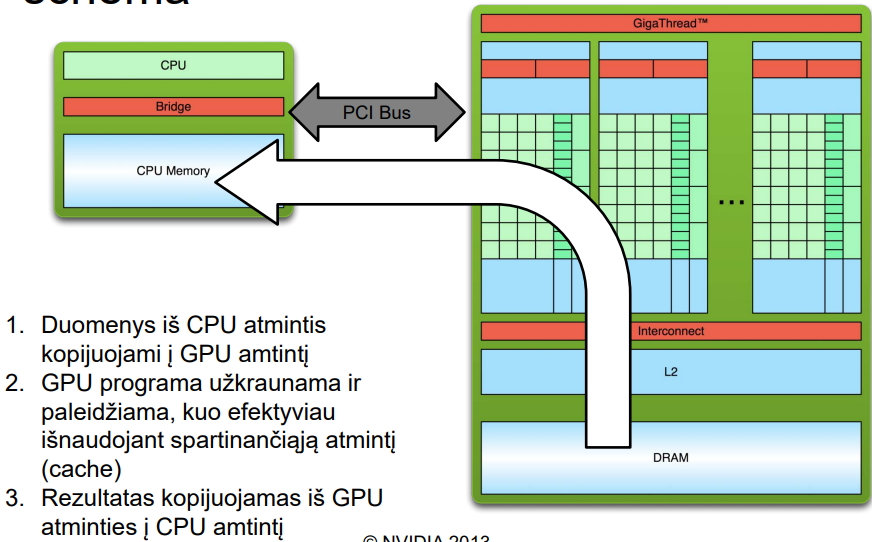
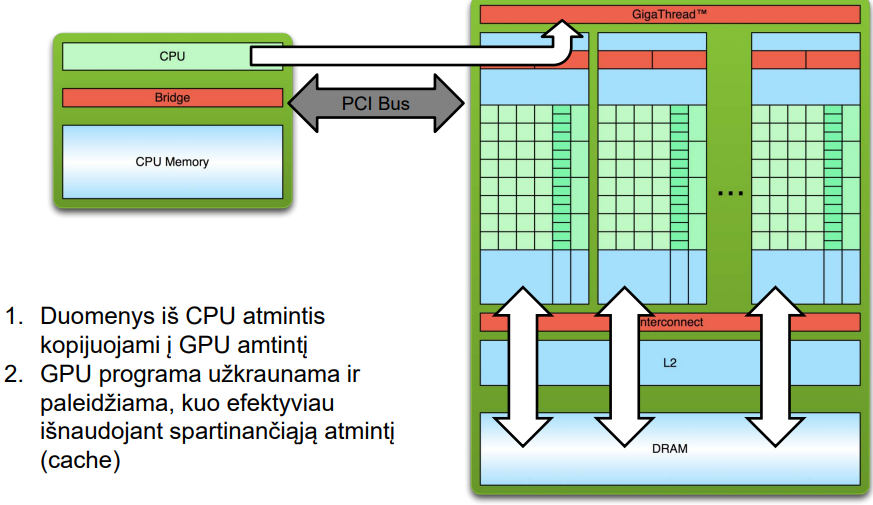
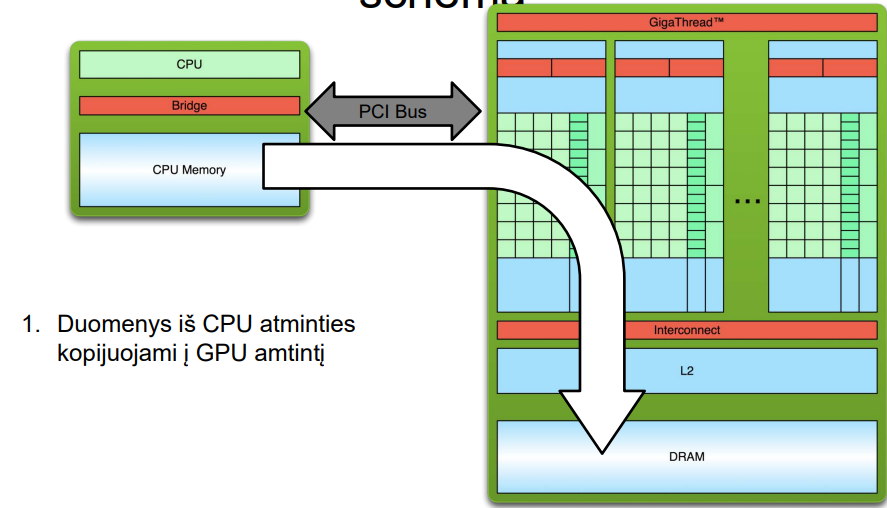
*5. L2 cache atmintis - spartinančioji atmintis, kuri yra bendra visiems multiprocesoriams, greitesnė už pagrindinę, bet lėtesnė už greitąją.*

*6. Bendros atminties siuntimai yra prieinami per taip vadinamus bankus (bank). Skirtinguose bankuose siuntimai vykdomi lygiagrečiai, tačiau kiekvienas multiprocesorius turi ribotą bankų skaičių.*

*7. Kiekvienas GPU multiprocesorius turi ribotą registrų skaičių.*

*8. Dar yra constant ir texture tipų atmintis, tačiau jų mes nenagrinėsime.*

* 1. **Duomenų srauto schema kai skaičiavimuose naudojamas GPU**



* 1. **SIMT principas, gijų porcijų (angl. warp) darbo vykdymo ypatumai**

*SIMT - single instruction multiple threads. Single instruction programuotojo akimis yra GPU funkcijoje esančios instrukcijos.*

*Kiekvienas multiprocesorius sukuria, apdoroja, sudaro tvarkaraštį, paleidžia gijas grupėmis po 32 gijas, vadinsime jas gijų porcijomis (warps).*

*Vienoje gijų porcijoje visos gijos pradeda darbą vienu metu. Programuotojas turi siekti išlaikyti jų sinchronišką veikimą (vengti išsišakojimo su if ir t.t.). Nes kitu atveju kiekviena iš šakų yra vykdoma nuosekliai viena po kitos, o gijų dalis laukia.*

*Kai multiprocesorius gauna vykdyti vieną ar daugiau užduočių blokų - kiekvienas blokas išskaidomas į gijų porcijas ir kiekviena iš porcijų yra valdoma tvarkaraščio sudarymo ir vykdymo mechanizmais.*

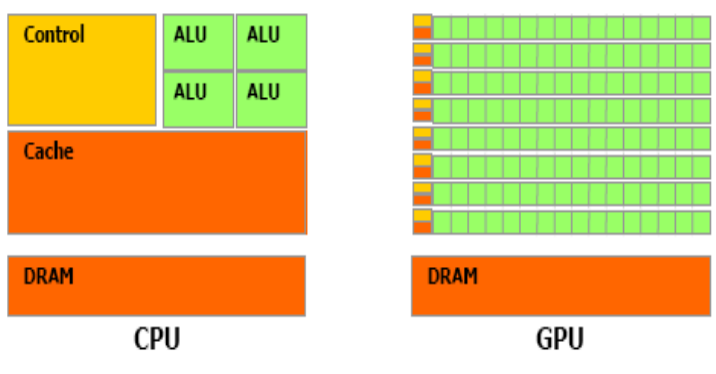
*Viena gijų porcija gali vykdyti tik vieną instrukciją, todėl reikia rūpintis, kad visos gijos vienoje porcijoje tuo pačiu metu atlikinėtų tą pačią instrukciją.*

*Tokio inžinerinio sprendimo priežastimi galima laikyti skaičiavimų valdymo supaprastinimą, kadangi 32-jų gijų valdymui reikalingas tik vienas gijų valdymo mechanizmas (control unit).*

* 1. **GPU skaičiavimų pajėgumas (angl. compute capability)**

*Visi NVIDIA GPU įrenginiai charakterizuojami GPU kartos rodikliu - skaičiavimų pajėgumu, angl. compute capability. GPU su didesniu skaičiavimų pajėgumu paprastai sumažina reikalavimus efektyvios programos kodo rašymui.*

* 1. **CPU ir GPU palyginimas**

 *Mažiau skaičiavimų reikalaujančios ir blogiau išlygiagretinamos algoritmo dalys skaičiuojamos su CPU. Daugiausia skaičiavimų reikalaujančios algoritmų dalys (jei jos gerai išlygiagretinamos) skaičiuojamos su GPU.*

# PRIEDAI

## 1 PRIEDAS. PIRMOJO UŽDAVINIO SPRENDIMO KODAS

### main.py

1 #!/usr/bin/python

2

3 **import** sys

4

5 **import** map

6 **import** sort

7 **import** red

### map.py

1 #!/usr/bin/env python

2 **import** sys

3

4 sys**.**stdin **=** **open(**"duom\_full.txt"**,"**r"**)**

5 sys**.**stdout **=** **open(**"mapout.txt"**,**"w"**)**

6

7 **for** line **in** sys**.**stdin**:**

8 line **=** line**.**strip**()[**2**:-**2**]**

9 stops **=** line**.**split**(**'}}{{'**)**

10 **for** stop **in** stops**:**

11 #print('%s\t%s' % (1,stop))

12 atts **=** stop.split**(**'}{'**)**

13 siuntu\_sk **=** **None**

14 klientu\_sk **=** **None**

15 geo\_zona = **None**

16 savaites\_d **=** **None**

17 **for** at **in** atts:

18 name**,** value **=** at**.**split**(**'='**)**

19 **if** **(**name **==** "siuntu skaicius"**):**

20 siuntu\_sk **=** **int(**value**)**

21 **if** **(**name **==** "Sustojimo klientu skaicius"**):**

22 klientu\_sk **=** value

23 **if** **(**name **==** "geografine zona"**):**

24 geo\_zona **=** value

25 **if** **(**name **==** "sustojimo savaites diena"**):**

26 savaites\_d **=** value

27 **if** siuntu\_sk **!=** **None** **and** klientu\_sk **!=** **None** **and** geo\_zona **!=** **None** **and** **len(**geo\_zona**)>**1 and savaites\_d **!= None**:

29 **print(**'%s\t%s' **%** **(str(str(**geo\_zona**)+**" "**+str(**savaites\_d**))**, **str(str(**siuntu\_sk**)+**"\_"**+str(**klientu\_sk**))))**

### sort.py

1 #!/usr/bin/env python

2 **import** sys

3

4 sys**.**stdin **=** **open(**"mapout.txt"**,**"r"**)**

5 sys.stdout **= open(**"smapout.txt"**,**"w"**)**

6 A **=** **[]**

7 **for** line **in** sys**.**stdin**:**

8 key**,** val **=** line.strip**().**split**(**'\t'**,** 1**)**

9 A**.**append**([**key**,**val**])**

10

11 A**.**sort**(**key**=lambda** tup**:** tup**[**0**])**

12 **for** el **in** A**:**

13 **print(**"%s\t%s" **%** **(**el**[**0**],** el**[**1**]))**

### red.py

1 #!/usr/bin/env python

2 **import** sys

3

4 sys**.**stdin **=** **open(**"smapout.txt"**,**"r"**)**

5 sys**.**stdout **=** **open(**"redout.txt"**,**"w"**)**

6

7 current\_k **=** **None**

8

9 k **= None**

10 siuntu\_skaicius **=** 0

11 klientu\_skaicius **=** 0

12

13 **for** line **in** sys**.**stdin**:**

14 line **=** line**.**strip**()**

15 k**,** v **=** line**.**split**(**'\t'**,** 1**)**

16 vv1**,** vv2 **=** v**.**split**(**'\_'**,** 1**)**

17 **try:**

18 vv1 **=** **int(**vv1**)**

19 vv2 **=** **int(**vv2**)**

20 **except ValueError:**

21 **continue**

22 **if** current\_k **==** k**:**

23 siuntu\_skaicius **+=** vv1

24 klientu\_skaicius **+=** vv2

25 **else:**

26 **if** current\_k **!=** **None:**

27 **print** **(**'%s\t%s' **%** **(**current\_k**,** **str(**siuntu\_skaicius**)** **+" "+** **str(**klientu\_skaicius**)))**

28 siuntu\_skaicius **=** vv1

29 klientu\_skaicius **=** vv2

30 current\_k **=** k

31

32 **if** current\_k **!=** **None:**

33 print **(**'%s\t%s' **%** **(**current\_k**,** **str(**siuntu\_skaicius**) +" "+** **str(**klientu\_skaicius**)))**

## 2 PRIEDAS. ANTROJO UŽDAVINIO SPRENDIMO KODAS

1 **from** pyspark **import** SparkContext**,** SparkConf

2

3 conf **=** SparkConf**().**setAppName**(**'MyFirstStandaloneApp'**)**

4 sc **=** SparkContext**(**conf**=**conf**)**

5

6 lines **=** sc**.**textFile**(**"duom\_full.txt"**)**

7 **def** FM**(**line**):**

8 line **=** line**.**strip**()**

9 line **=** line**[**2**:-**2**]**

10 susstring **=** line**.**split**(**'}}{{'**)**

11 **return** susstring

12

13 A **=** lines**.**flatMap**(**FM**)**

14

15 **def** MapF**(**stopas**):**

16 parstrings **=** stopas**.**split**(**'}{'**)**

17 marsrutas **= None**

18 svoris **=** **None**

19 siuntos **=** **None**

20 geo\_zona **=** **None**

21

22 z1 **=** 0

23 z2 **=** 0

24 z3 **=** 0

25

26 **for** parstring **in** parstrings**:**

27 **(**vardas**,** reiksme**)** **=** parstring**.**split**(**'='**)**

28 **if (**reiksme **!=** '' **and** vardas **==** 'marsrutas'**):**

29 marsrutas **=** reiksme

30 **if (**reiksme **!=** '' **and** vardas **==** 'svoris'**):**

31 svoris **=** reiksme

32 **if (**reiksme **!=** '' **and** vardas **==** 'siuntu skaicius'**):**

33 siuntos **=** reiksme

34 **if (**reiksme **!=** '' **and** vardas **==** 'geografine zona'**):**

35 geo\_zona **=** reiksme

36 **if** **(** geo\_zona **==** "Z1" **):**

37 z1 **+=** 1

38 **elif (** geo\_zona **==** "Z2" **):**

39 z2 **+=** 1

40 **else:**

41 z3 **+=** 1

42

43 **try:**

44 marsrutas **=** **int(**marsrutas**)**

45 svoris **=** **float(**svoris**)**

46 siuntos **=** **int(**siuntos**)**

47  **except:**

48 svoris **=** 0

49 siuntos **=** 0

50 **return (** marsrutas**, (**svoris**,** siuntos**,** z1**,** z2**,** z3**) )**

51

52 B **=** A**.map(**MapF**)**

53

54 **def** red**(**a**,** b**):**

55 **return (** a**[**0**]+**b**[**0**],** a**[**1**]+**b**[**1**]**, a**[**2**]+**b**[**2**],** a**[**3**]+**b**[**3**] ,** a**[**4**]+**b**[**4**] )**

56

57 C **=** B.**reduceByKey(**red**)**

58 ats **=** C**.**collect**()**

59 **print(**ats**)**

60

61 C2 **=** C**.filter(lambda** a **:** a**[**0**] != None)**

62 C2**.saveAsTextFile(**"ats"**)**

## 3 PRIEDAS. TREČIOJO UŽDAVINIO SPRENDIMO KODAS

1 **from** pyspark **import** SparkContext, SparkConf

2 conf **=** SparkConf**().**setAppName**(**'MyFirstStandaloneApp'**)**

3 sc **=** SparkContext**(**conf**=**conf**)**

4

5 **from** pyspark.sql **import** SparkSession

6 spark **=** SparkSession.builder.appName**(**'Laboras3').getOrCreate**()**

7

8 **from** pyspark.sql.functions **import** udf, log

9 **from** pyspark.sql.types **import** **\***

10

11 text\_file **=** spark.sparkContext.textFile**(**"duom\_full.txt"**)**

12

13 **def** parsinam**(**line**):**

14 #line=line[2:len(line)-2]

15 **return** line**[**2**:**len**(**line**)**-2**].**split**(**'}}{{'**)**

16

17 **def** parsinam2**(**line**):**

18 objs **=** line.split**(**'}{'**)**

19 k1 **=** **None**

20 k3 **=** **None**

21 tipas **=** **None**

22 siuntu\_sk **=** 0

23 **for** at **in** objs**:**

24 temp **=** at.split**(**'='**)**

25 **if(**len**(**temp**)** **<** 2**):**

26 **break**

27 key,val **=** at.split**(**'='**)**

28 **if** (key **==** 'Masinos tipas'):

29 tipas **=** val

30 **if**(key **==** 'marsrutas'):

31 k1 **=** val

32 **if**(key **==** 'sustojimo data'):

33 k3 **=** val

34 **if** (key **==** "siuntu skaicius"):

35 siuntu\_sk **=** val

36 **if(**k1 **!=** **None** **and** k3**!=None** **and** tipas**!=None** **and** siuntu\_sk**!=None):**

37 **return** **(str(**k1**)+**"\_"**+str(**k3**),(int(**siuntu\_sk**),**tipas**))**

38 **else**:

39 **return** **(**"0"**,(**0**,**tipas**))**

40

41 fmap **=** text\_file.**flatMap(**parsinam**)**

42 tipai **=** fmap.**map(**parsinam2**).map(lambda** t**:(**t**[**1**][**1**],**0**))\**

43 .**reduceByKey(lambda** a,b**:**a**).map(lambda** t**:**t**[**0**])**.collect**()**

44 **print(**tipai**)**

45 #for tip in tipai:

46 aggregated\_rdd **=** fmap.**map(**parsinam2**)\**

47 .**reduceByKey(lambda** a,b**:** **(**a**[**0**]+**b**[**0**],**a**[**1**]))\**

48 .**filter(lambda** t**:** t**[**1**][**1**]==**"depot"**)**.**map(lambda** t**:** **(**t**[**0**],int(**t**[**1**][**0**])))**

49 aggregated\_df **=** aggregated\_rdd.**toDF([**"ID"**,** "Siuntu skaicius"**])**

50 aggregated\_df**.show()**

51

52 routes **=** spark.**read**.**option(** "header"**,** **True)**.**csv(** "RouteSummary.txt",

inferSchema **=** **True)**

53 routes.printSchema()

54 routes **=** routes.**drop(**"M", "BendrasAtstumas","BendrasSvoris","BendraKaina"**)**

55

56 **def** makeID**(**str1, str2**):**

57 **return** **str(**str1**)+**"\_"**+**str2

58 makeID\_UDF **=** **udf(lambda** z1**,**z2**:** makeID**(**z1,z2**),**StringType**())**

59

60 **def** convertTime**(**l**):**

61 m,s **=** l.split**(**":"**)**

62 m **= float(**m**)**

63 **return** **float(**s**) +** m**\***60

64

65 convertTime\_UDF **=** **udf(lambda** z1**:** convertTime**(**z1**)**,DoubleType**())**

66 routes2 **=** routes.**withColumn(**'sekundes', convertTime\_UDF**(**"BendrasLaikas"**)).**

**drop(**"BendrasLaikas"**)**

67 routes2**.**show**()**

68

69 routes3 **=** routes2**.withColumn(**'ID', makeID\_UDF**(**"marsrutas", "sustojimo

data"**)).drop(**"marsrutas", "sustojimo data"**)**

70 routes3**.**printSchema**()**

71

72 #dviejų lentelių apjungimas

73 routes\_full **=** aggregated\_df**.join(**routes3**,**'ID','inner'**)**

74

75 **from** pyspark.ml.feature **import** VectorAssembler

76 **from** pyspark.ml.regression **import** LinearRegression

77

78 vectorAssembler **=** VectorAssembler**(**inputCols **=** **[**'Siuntu skaicius'**],**

outputCol **=** 'features'**)**

79 vhouse\_df **=** vectorAssembler.**transform(**routes\_full**)**

80 training **=** vhouse\_df.**select([**'features'**,** 'sekundes'**])**

81 training.**show(**3**)**

82

83 lr **=** LinearRegression**(**featuresCol **=** 'features', labelCol**=**'sekundes'**)**

84

85 lrModel **=** lr.**fit(**training**)**

86

87 **print(**"Coefficients: " **+** **str(**lrModel.coefficients**))**

88 **print(**"Intercept: " **+** **str(**lrModel.intercept**))**

89

90 a **=** lrModel.intercept

91 b **=** lrModel.coefficients**[**0**]**

92

93 trainingSummary **=** lrModel.summary

94 **print(**"numIterations: %d" **%** trainingSummary.totalIterations**)**

95 **print(**"objectiveHistory: %s" **%** **str(**trainingSummary.objectiveHistory**))**

96 trainingSummary.residuals.**show()**

97 **print(**"RMSE: %f" **%** trainingSummary.rootMeanSquaredError**)**

98 **print(**"r2: %f" **%** trainingSummary.r2**)**

99

100 pandasDF **=** routes\_full.toPandas**()**

101 pandasDF.**head()**

102

103 X **=** pandasDF**[**'Siuntu skaicius'**].to\_list()**

104 Y **=** pandasDF**[**'sekundes'**].to\_list()**

105

106 **import** matplotlib.pyplot **as** plt

107 **from** matplotlib **import** cm

108 **from** matplotlib.ticker **import** LinearLocator

109 **from** mpl\_toolkits.mplot3d **import** Axes3D

110

111 fig, axes **=** plt.subplots**(**nrows**=**1, ncols**=**3, figsize**=(**20,5**))**

112 axes**[**2**]**.scatter**(**X,Y, s**=**5**)**

113 lineX **=** **[min(**X**),** **max(**X**)]**

114 lineY **=** **[**a **+** b **\*** lineX**[**0**],** a **+** b **\*** lineX**[**1**]]**

115 axes**[**2**].**plot**(**lineX, lineY, color**=**'red'**)**