CSE598-Spring 2020

Assignment8.

Dense Matrix Multiplication Comparison: Hand-coded, Breeze, Spark

Project Report

Professor: Zou. Jia

Student Name: Weichi Zhao

Student ID: 1209692845

Email: wzhao42@asu.edu

- **Hand-coded implementation can work: 3pt**

```
235.72977312008226 252.88690305799346 257.7261071487286 256.654293488066 246.86907071834776 251.17513357449062 254.49413546911236 248.0
230.4289002129897 242.29005686522243 247.25587773282257 249.86769986153343 240.70191945895988 250.40399524715798 250.32381536288884 239
240.95844398980145 261.47284709334525 257.5733504044482 261.781634011792 247.19643842936205 257.1685109615922 258.00132479758514 248.78
243.62963995654408 252.58329846606318 256.68133073538394 261.74509140086184 249.41536232788698 254.19255374959178 250.69116160355716 23
242.76553448515463 248.75725109786782 253.27383707628775 251.64391934269221 249.10055646901674 253.44304956933368 255.47922933952424 24
245.07044026009856 259.1065521048037 264.69461295042015 272.4110184137721 256.4658837103749 270.0318456841251 260.24712359797894 257.06
239.8452382785641 258.0787828197392 254.21967419798935 260.30885857377166 247.87529167676158 251.67840879603443 253.47893846293607 245.
236.86452118511484 249.67978385540474 256.02131427127466 265.05490732318935 246.12903115406178 260.7819197078856 253.43159003827864 247
234.12371999704308 250.43064303316072 254.95244638594613 255.27314615227772 242.55734634252187 259.5859565284178 253.81302374809198 239
234.7777120729522 248.70030338675656 251.47811854192298 252.9843266344241 240.63775570088936 251.73461316409683 247.98216393615758 242.
232.26158951715087 239.14567620690883 248.41475537205557 258.29492461594305 241.7583591237724 252.16388374285438 249.66491117610448 234
241.55873178040326 259.66270267982793 264.58282812273615 263.1052790376752 252.5054750219284 263.8752025608239 258.10355490766375 248.2
239.33107369865417 250.14283216760518 254.96532808140574 255.51195467222541 246.8923654790311 260.78415847682066 253.0431803605605 243.
235.311338371044 254.53294955905838 255.85978531485745 261.5414091749018 251.47922766738043 261.6971834279119 255.96938345177654 242.99
247.44912944239914 260.35833953980085 256.7568826517732 262.8101538631769 254.1892721934339 261.9955091824311 260.0600541074917 244.434
245.01925140077324 251.84565844730813 259.33667485973456 262.16880258071177 250.71186319602978 260.47232928431305 252.3957798117815 243
233.19896892146595 247.1462339595994 247.778496047435 250.99765031892753 237.44290593197871 246.88260573755454 252.53004134264344 237.4
240.22877826042446 250.43502734775464 255.6260475218437 261.786105155756 241.4276478607329 252.70572459191314 252.48512122235206 242.94
239.9376293962724 247.32998503638285 248.10709471484859 253.49808449799892 243.04930055742403 253.61156287596512 252.78273261740603 244
228.650268391992 241.67617383333487 242.86297587217382 245.12787011678023 232.3857151401967 244.26912714016692 241.14458992440655 236.5
243.15442135511458 254.20445442560424 253.56597307611696 259.23390868311096 244.8749298408432 258.7292613599439 253.24974785798 238.885
243.09740968188237 256.53905296741726 258.55598910333197 264.1553361467152 251.54990487039515 260.17334121455355 254.2256324476929 247.
```

```
Process time of Dense Matrix Multiplication using Hand-coded scala: 13.606358689 second(s)
Average process time of Dense Matrix Multiplication using Hand-coded scala: 0.13606358689 second(s)

Process finished with exit code 0
```

- **Spark implementation can work: 3pt**

```
242.9867376894244   248.5825547266446   251.5181927924703   ...
244.59990744725627  248.676496028416    252.42677050094335  ...
249.791275652174    252.19696340090385  252.6966172564088   ...
251.37674250239166  250.5075413546023   253.8527149868731   ...
240.72794601938102  238.75058788896465  245.22262038233654  ...
244.80146781625157  247.97261319631917  245.25700490123774  ...
253.66813054003654  254.76470681891894  250.64514409864813  ...
254.53072287553647  253.97937413250312  252.21323857561384  ...
246.22275085598687  248.51237083348016  247.33250223462954  ...
249.21138910798072  255.16426518557256  253.57085803977256  ...
254.77628455634294  252.50901513958195  255.2510632554872   ...
248.51599246655755  250.70379556191213  255.3546740202075   ...
256.51240708257995  257.94929839665343  260.48475733108137  ...
249.9147221042649   254.32735329094365  252.16173821138986  ...
246.6484393111493   251.42728582510864  250.60222783424467  ...
253.7471871547881   254.6162582264906   254.8861678984912   ...
244.34703051496527  250.08128592983925  247.62279709881284  ...
255.244789779947    256.98281357992556  245.9889723298474   ...
255.20857054615018  258.78903137323107  256.0026070271346   ...
249.59119300484292  250.57297289360707  255.84107533436747  ...
248.51382372999524  257.699775789721    251.26472531070414  ...
248.59191925097082  251.00159488200993  246.40311828530383  ...
... (1000 total))
ProcessTime of Dense Matrix Multiplication using Spark: 1.058135671 second(s)
Average process time of Dense Matrix Multiplication using Spark: 0.01058135671 second(s)
20/04/16 23:50:12 INFO SparkUI: Stopped Spark web UI at http://192.168.223.128:4040
20/04/16 23:50:12 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
```

- **Breeze implementation can work: 3pt**

```
251.47878185462542  248.45668842400116  242.4087610987665   ...
255.95707092243714  251.88585302239235  252.51037737355273  ...
255.27698838151917  247.41248406976504  243.37714312695772  ...
253.43775358887902  245.1227518748211   250.0506128129136   ...
255.1606269140931   253.04855328316802  246.34587335091405  ...
268.52641341175837  262.49299811366234  253.43006237993893  ...
267.8559403924122   259.7478200831416   253.6058107267238   ...
259.9930826310499   256.7047605782091   250.5999059102858   ...
262.8689588144696   258.3745854931106   252.73965008119023  ...
252.15131650405283  250.04963522881226  243.42976581202626  ...
258.1494096246601   254.46997393968593  251.28330647475195  ...
253.94264678198786  249.39255465961588  239.80033725055338  ...
267.4051666083614   258.56252507479746  252.31578850770404  ...
251.59081614303557  248.7537676435388   243.6728822004324   ...
244.7623312260687   239.5275810517491   234.03903616890918  ...
253.1397209742874   250.16210529512367  248.69608282995245  ...
256.4332589023562   251.8272970018962   250.75567328037536  ...
258.0267958689733   254.02456443772815  245.8408532306435   ...
249.58864950118777  243.75878885328086  233.651821320769    ...
251.13163426374885  247.2972173898635   243.80801140425396  ...
253.40796205432406  251.1819784588757   246.34594073197607  ...
263.6845722642794   258.4853600827644   247.67560186146596  ...
... (1000 total)
ProcessTime of Dense Matrix Multiplication using breeze: 1.02007906 second(s)
Average process time of Dense Matrix Multiplication using breeze: 0.0102007906 second(s)

Process finished with exit code 0
```

- **In the report, the machine hardware configuration is listed: 3pt**
  - **Framework: VMware**
  - **Virtualized System:** Ubuntu 18.04 LTS
  - **Virtualized RAM:** 4096 MB
  - **Virtualized CPU cores:** 4
- **In the report, measured time for the three implementations are listed and compared in a way that is easy to understand: 4pt**
  - Hand-coded Scala code for executing 1000x1000 dense matrix multiplication for N times and get the average time for executing one matrix multiplication.
    - **The number of times:**
      - 1000 times
    - **Processing time:**
      - 13.606358689 seconds
    - **Average time for executing one matrix multiplication:**
      - 0.13606358689 seconds
  - Spark code for executing 1000x1000 local dense matrix multiplication for N times and get the average time for executing one matrix multiplication.
    - **The number of times:**
      - 1000 times
    - **Processing Time:**
      - 1.058135671 seconds
    - **Average time for executing one matrix multiplication:**
      - 0.01058135671 seconds
  - Scala code using Breeze for executing 1000x1000 local dense matrix multiplication for N times and get the average time for executing one matrix multiplication.
    - **The number of times:**
      - 1000 times
    - **Processing Time:**
      - 1.02007906 seconds
    - **Average time for executing one matrix multiplication:**
      - 0.0102007906 seconds
  - **Comparison(processing time):**
    - Hand-coded Scala code > Spark code ≈ Scala code using Breeze
    - Hand-coded Scala code took the longest time to finish matrix multiplication, Spark code and Scala code using Breeze have a similar processing time.

- **In the report, observations and things learned from the implementation and performance comparison are summarized: 4pt**
    - **Observation:** After implementing the three different approaches for matrix multiplication, I noticed that different procedures of mathematical calculation could result in significant performance differences. I also notice that Scala is a powerful OO programming language and Spark provides a human-kind interface for submitting the tasks.
    - **Things Learned:** During this assignment, I practiced the setup of the SBT environment, basic Scala programming ability, and Spark hands-on implementation for submitting the task to the local cluster. I learned how to hand-code the simple matrix multiplication by using Scala, and how to utilize the library like breeze and Spark-Mllib to create the matrix and implement matrix multiplication. And I also learned how to measure and evaluate the performance of the program in Scala and Spark.
    - **Performance Comparison:** While the implementation of matrix multiplication in three different approaches above, the results of processing time under the same hardware environment are different. The implementation of hand-coded in Scala took the longest time which is around 14 seconds and it took 0.13606358689 seconds to finish one matrix multiplication. Both performances of Scala code using breeze and Spark code are similar which is around 1 second and it took 0.01 second to do one matrix multiplication, but the breeze implementation is slightly faster than Spark implementation.