# Exercise 2: KNN Search

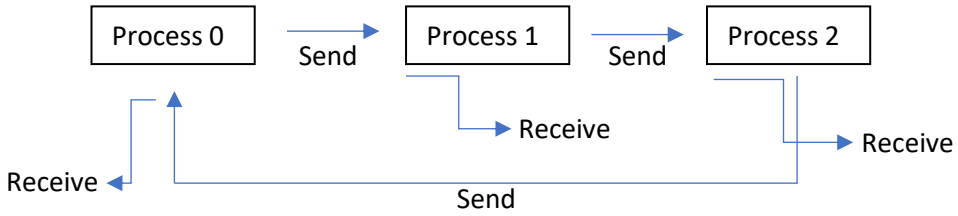
CODES:

## Version 0: Sequential

In this version the query set is Y[m by d] and the corpus set is X[n by d].
First of all we have to calculate the distances between the points of each set and after that select the k minimum distances for every point in query set Y.
To calculate the distances, we use the functions *cblas_ddot* and *cblas_dgemm* from the *cblas library*, to create the function of Euclidean distance, *"D = sqrt(sum(X.^2,2) - 2 * X*Y.' + sum(Y.^2,2).')"*.
Because of the way that the array of distances (D) is being calculated, every column corresponds to the query points. Then to find the k-nearest neighbours of every query point, we use the function "*kSelect*". In this function, at first we separate every column of array (D). Then we use *selection sort* to sort each column and get the k-nearest neighbours for every column (=query point).
It is more preferable to use this *sort* because we can use it to sort only the first k-elements of an array instead of the whole array.

## Version 1: Asynchronous

In this version the query set and the corpus set is the same array, X[n by d].
We are using MPI, were we can set "n" processes and set parts of the query and corpus set, on each one. With that we can achieve faster execution times for large corpus and query sets.
At first, we use *"MPI_Scatterv"* to separate the query set to "n" parts and assign each part to each process.
Then we use the functions *MPI_send* and *MPI_recv,* to send every part of corpus set, from the running process to the next one.




At this phase, because our sets are type of *Double*, *the MPI_send* and *MPI_recv* were able to work only with a specific buffer size, so we have to split those parts of corpus set into more parts. To do that we use a *const divide number (div)* instead of the number of processes.
This const helps with the decreasing the run time too. *(It is mentioned below on results)*
The distances are being calculated as before, but this time, we calculate multiple arrays of distances (D) which are temporary. Because of that , we are using a function called *"Convert"*, where each time we get the new calculated array (D) and add it next to the previous one.

For example, we have array A = $\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$ and B = $\begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix}$ the result will be $\begin{bmatrix}1 & 2 & 5 & 6\\ 3 & 4 & 7 & 8\end{bmatrix}$.

Finally, we call "*kSelect*" to find the k-nearest neighbors which works as before, the only difference is that it will be called as "n" times. *(n = number of processes)*

**More comments in codes.**

**NOTES: All "runs" for each version, were executed locally because I could not run them on hpc. This may affect the results of run times. I used only random generated arrays. The arrays of struct "knnresult" are saved in text files.*

Runs and Results:

```
csal@07743a15e0c1: ~/pds/programs/mpi
csal@07743a15e0c1:~/pds/programs/mpi$ gcc v0.c -lopenblas -o run1
csal@07743a15e0c1:~/pds/programs/mpi$ ./run1

Time measured: 5.064889 seconds.

csal@07743a15e0c1:~/pds/programs/mpi$ gcc v0.c -lopenblas -o run2
csal@07743a15e0c1:~/pds/programs/mpi$ ./run2

Time measured: 24.225018 seconds.

csal@07743a15e0c1:~/pds/programs/mpi$ gcc v0.c -lopenblas -o run3
csal@07743a15e0c1:~/pds/programs/mpi$ ./run3

Time measured: 66.099528 seconds.

csal@07743a15e0c1:~/pds/programs/mpi$ gcc v0.c -lopenblas -o run4
csal@07743a15e0c1:~/pds/programs/mpi$ ./run4

Time measured: 28.652319 seconds.
```

Version 0:

Run 1 → n=10000, m=5000, d=20, k=25 Time: 5.065 s
Run 2 → n=15000, m=10000, d=15, k=50 Time: 24.225 s
Run 3 → n=20000, m=15000, d=10, k=75 Time: 66.099 s
Run 4 → n=20000, m=20000, d=5, k=10 Time: 28.65 s

Comments – Version 0:

If we compare the times of Run 2 and Run 4, with the Run 3, it is clear that not only the size of corpus and query set can affect the run time, but also the number of k-neighbors affects it significantly.

Version 1:

Run 1a → *n=15000, d=15, k=10, div=1500, numP=2* Time=7.60 s
Run 1b → *n=15000, d=15, k=20, div=1500, numP=2* Time=12.69 s
Run 1c → *n=15000, d=15, k=10, div=750, numP=2* Time=18.38 s

```
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 2 ./run1

Time measured: 7.601617 seconds.

csal@07743a15e0c1:~/pds/programs/mpi$ mpicc v1.c -lopenblas -o run1b
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 2 ./run1b

Time measured: 12.697748 seconds.

csal@07743a15e0c1:~/pds/programs/mpi$ mpicc v1.c -lopenblas -o run1c
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 2 ./run1c

Time measured: 18.338738 seconds.
```

Run 3a → *n=25000, d=10, k=10, div=1000, numP=4* Time=41.94 s
Run 3b → *n=25000, d=10, k=10, div=1000, numP=5* Time=27.87 s

```
csal@07743a15e0c1:~/pds/programs/mpi$ mpicc v1.c -lopenblas -o run3a
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 4 ./run3a

Time measured: 41.943409 seconds.

csal@07743a15e0c1:~/pds/programs/mpi$ mpicc v1.c -lopenblas -o run3b
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 5 ./run3b

Time measured: 27.872929 seconds.
```

Run 2a → *n=20000, d=10, k=10, div=1000, numP=2* Time=32.65 s
Run 2b → *n=20000, d=10, k=10, div=2000, numP=2* Time=16.51 s
Run 2c → *n=20000, d=10, k=10, div=1000, numP=4* Time=21.72 s
Run 2d → *n=20000, d=10, k=10, div=2000, numP=4* Time=11.84 s
Run 2e → *n=20000, d=10, k=20, div=2000, numP=4* Time=14.27 s

```
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 2 ./run2a

Time measured: 32.656126 seconds.

csal@07743a15e0c1:~/pds/programs/mpi$ mpicc v1.c -lopenblas -o run2b
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 2 ./run2b

Time measured: 16.513171 seconds.

csal@07743a15e0c1:~/pds/programs/mpi$ mpicc v1.c -lopenblas -o run2c
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 4 ./run2c

Time measured: 21.724998 seconds.

csal@07743a15e0c1:~/pds/programs/mpi$ mpicc v1.c -lopenblas -o run2d
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 4 ./run2d

Time measured: 11.849320 seconds.

csal@07743a15e0c1:~/pds/programs/mpi$ mpicc v1.c -lopenblas -o run2e
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 4 ./run2e

Time measured: 14.277608 seconds.
```

Run 4a → *n=30000, d=10, k=10, div=1000, numP=6* Time=76.43 s
Run 4b → *n=30000, d=10, k=10, div=1000, numP=8* Time=38.90 s
Run 4c → *n=30000, d=10, k=10, div=2000, numP=8* Time=23.93 s

```
csal@07743a15e0c1:~/pds/programs/mpi$ mpicc v1.c -lopenblas -o run4a
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 6 ./run4a

Time measured: 76.431174 seconds.

csal@07743a15e0c1:~/pds/programs/mpi$ mpicc v1.c -lopenblas -o run4b
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 8 ./run4b

Time measured: 38.905138 seconds.

csal@07743a15e0c1:~/pds/programs/mpi$ mpicc v1.c -lopenblas -o run4c
csal@07743a15e0c1:~/pds/programs/mpi$ mpirun -np 8 ./run4c

Time measured: 23.935266 seconds.
```

Comments – Version 1:

First of all, according to the results above, we can see that by increasing the *number of processes (numP),* or the *number (div)* that splits our corpus set into "div" parts, we get a better run time.
For example, *"Run 2a"* and *"Run 2b"* use the same number of processes, but if we split the corpus set into more parts, the execution time is x2 faster. The same happens with *"Run 1a"*, *"Run 1c"*, *"Run 4b"* and *"Run 4c"* .
Also if we compare *"Run 2a"* with *"Run 2c"*, were the only difference is the number of processes, the run time execution is faster. Furthermore, we can compare *version 0* with *version 1*. If we compare the highlighted "runs", with 2 processes we get almost x2 faster run time and with 4 processes we get almost x3 faster run time, even if the corpus and query set have more dimensions (d). If the sets were the same size, we could get even faster run times.

To make sure that the const "*div*" does not affect our results, there are some more runs below, that use the same random array. *(Comments in code.)*

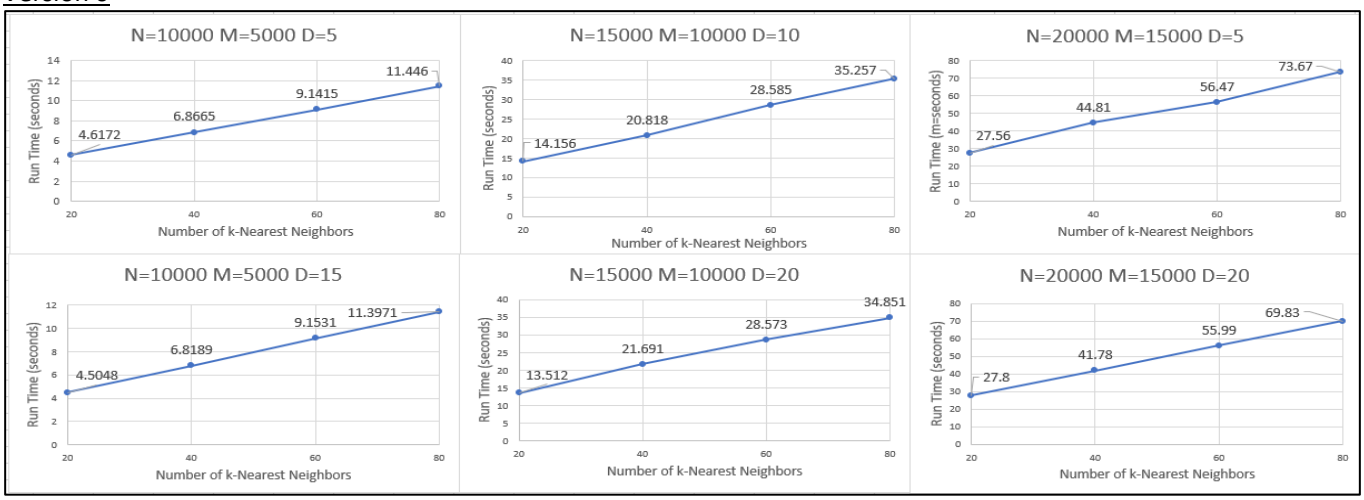
| Run 5a → *n=20000, d=10, k=10, div=1000, numP=4*: Time=22.21 s | Run 5b → *n=20000, d=10, k=10, div=2000, numP=4*: Time=9.05 |
|---|---|

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 16623049 | 17622200 | 21983862 | 24482293 | 27227681 | 30704991 | 33168598 | 34128867 | 34422928 | |
| 0 | 13807747 | 15442313 | 15999328 | 17543141 | 20678292 | 21008168 | 21763244 | 22224718 | 24638077 | |
| 0 | 11473248 | 13979653 | 14273374 | 18276613 | 18419105 | 19578612 | 19940624 | 20809338 | 21614074 | |
| 0 | 8122179 | 12182103 | 13567636 | 17467441 | 17762352 | 17956414 | 18320150 | 20075407 | 21031419 | |
| 0 | 19017614 | 21382222 | 21566374 | 26192709 | 26569293 | 27539208 | 27655326 | 28015289 | 28471828 | |
| 0 | 17742438 | 19598301 | 19678508 | 21624576 | 22861413 | 23050522 | 26566127 | 26666551 | 26955461 | |
| 0 | 12617738 | 14203515 | 18845018 | 19488740 | 22862232 | 24216441 | 24997015 | 25130793 | 26774824 | |
| 0 | 14811181 | 15567981 | 16824147 | 16968441 | 17138004 | 17393310 | 19119236 | 19955157 | 21097054 | |
| 0 | 8161933 | 11264070 | 11547654 | 12472275 | 15240774 | 15699175 | 17685422 | 18511989 | 19496768 | |
| 0 | 11069506 | 11951567 | 19312798 | 20655604 | 21296190 | 22613505 | 23010630 | 23582512 | 24282305 | |
| 0 | 12061392 | 14245066 | 15211980 | 15863198 | 18517053 | 18716065 | 19824068 | 20205911 | 20660442 | |
| 0 | 13229626 | 13428049 | 15547093 | 16024239 | 16408197 | 16828548 | 17937735 | 18082851 | 18164575 | |
| 0 | 10050462 | 14069466 | 17043145 | 17587837 | 21196536 | 22316708 | 23086425 | 23281173 | 24309300 | |
| 0 | 15925050 | 17088219 | 17548318 | 19458232 | 19914531 | 21223001 | 21823245 | 22327360 | 22747844 | |
| 0 | 15787718 | 19665640 | 20262783 | 20472305 | 20586516 | 21512309 | 22133672 | 23013209 | 23364921 | |
| 0 | 15269113 | 17552535 | 18609336 | 18674382 | 18838154 | 19741307 | 20114474 | 21250206 | 21269282 | |
| 0 | 7343665 | 10687345 | 15160887 | 16368270 | 17197284 | 17231020 | 17960772 | 19575980 | 23047531 | |

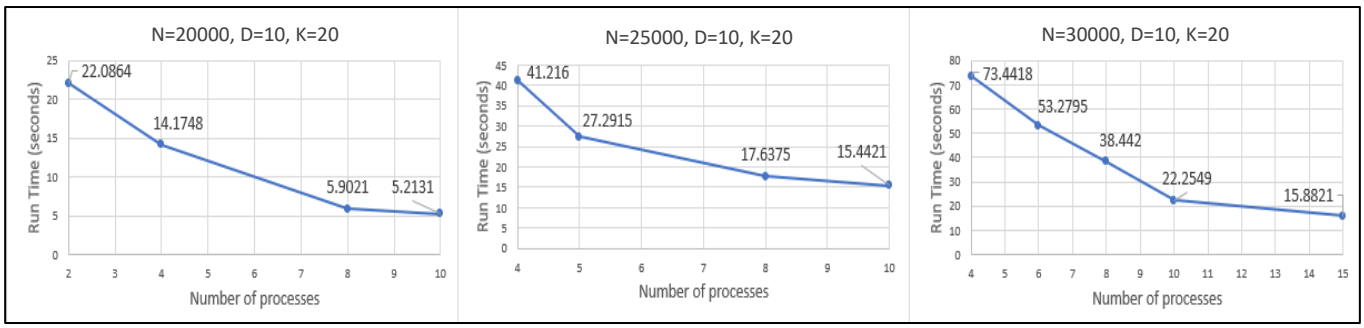
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 13807747 | 15442313 | 15999328 | 17543141 | 20678292 | 21008168 | 21763244 | 22224718 | 24638077 | |
| 0 | 17742438 | 19598301 | 19678508 | 21624576 | 22861413 | 23050522 | 26566127 | 26666551 | 26955461 | |
| 0 | 11069506 | 11951567 | 19312798 | 20655604 | 21296190 | 22613505 | 23010630 | 23582512 | 24282305 | |
| 0 | 15925050 | 17088219 | 17548318 | 19458232 | 19914531 | 21223001 | 21823245 | 22327360 | 22747844 | |
| 0 | 11473248 | 13979653 | 14273374 | 18276613 | 18419105 | 19578612 | 19940624 | 20809338 | 21614074 | |
| 0 | 8248645 | 18468373 | 19756710 | 20874625 | 21097612 | 21455877 | 21614521 | 21672730 | 21760012 | |
| 0 | 12617738 | 14203515 | 18845018 | 19488740 | 22862232 | 24216441 | 24997015 | 25130793 | 26774824 | |
| 0 | 12061392 | 14245066 | 15211980 | 15863198 | 18517053 | 18716065 | 19824068 | 20205911 | 20660442 | |
| 0 | 14860923 | 14905088 | 17067040 | 17643870 | 19645369 | 19791222 | 20352045 | 21389773 | 21503356 | |
| 0 | 15787718 | 19665640 | 20262783 | 20472305 | 20586516 | 21512309 | 22133672 | 23013209 | 23364921 | |
| 0 | 12607567 | 16852926 | 17995701 | 20399733 | 20987127 | 21893699 | 23412296 | 23918283 | 24854778 | |
| 0 | 9277214 | 14065982 | 17387739 | 18773105 | 18810934 | 19295234 | 21389452 | 22073441 | 22676981 | |
| 0 | 7583647 | 7949446 | 11128175 | 12085351 | 14196504 | 14509532 | 15214206 | 15544613 | 15900029 | |
| 0 | 16437231 | 16826640 | 16921699 | 16978922 | 19116345 | 19303540 | 19824693 | 21134234 | 22812100 | |
| 0 | 11688703 | 16885928 | 17567376 | 18425774 | 18857771 | 18876693 | 18944940 | 19574077 | 19723708 | |
| 0 | 15233529 | 17196421 | 19576945 | 21261065 | 21540083 | 21573356 | 22310225 | 22735729 | 23345685 | |
| 0 | 9396687 | 14394179 | 15881265 | 17182659 | 18190810 | 18771018 | 18772957 | 18935214 | 19547681 | |

Furthermore, we executed some more runs to represent in graph, how the run time could be affected.

Version 0




Version 1




GitHub Link: https://github.com/mariostavr/Exercise2_kNN