Exercise 2

Parallel & Distributed Computer Systems

Dec 10, 2022

Implement in `MPI` a distributed brute-force all-KNN search algorithm for the k nearest neighbors (k-NN) of each point $x \in X$.

The set of X points will be passed to you as an input array along with the number of points n, the number of dimensions d and the number of neighbors k.

Each `MPI` process P_i will calculate the distance of its own points from all (other) points and record the distances and global indices of the k nearest for each of its own points.

V0. Sequential

Write the sequential version, which finds for each point in a query set X the k nearest neighbors in the corpus set Y, according to this spec

```
// Definition of the kNN result struct
typedef struct knnresult{
  int    * nidx;    //!< Indices (0-based) of nearest neighbors [m-by-k]
  double * ndist;   //!< Distance of nearest neighbors          [m-by-k]
  int      m;       //!< Number of query points                 [scalar]
  int      k;       //!< Number of nearest neighbors            [scalar]
} knnresult;

//! Compute k nearest neighbors of each point in X [m-by-d]
/*!

  \param  X      Query data points              [m-by-d]
  \param  Y      Corpus data points               [n-by-d]
  \param  m      Number of query points          [scalar]
  \param  n      Number of corpus points         [scalar]
  \param  d      Number of dimensions            [scalar]
  \param  k      Number of neighbors             [scalar]

  \return  The kNN result
*/
```

```
knnresult kNN(double * X, double * Y, int n, int m, int d, int k);
```

To calculate an $m \times n$ Euclidean distance matrix D between two sets points X and Y of m and n points respectively, use the following operations

$$
D = (X \odot X)\, e\, e^{T} - 2\, X\, Y^{T} + e\, e^{T}(Y \odot Y)^{T}
$$

like in this `MATLAB` line:

```
D = sqrt(sum(X.^2,2) - 2 * X*Y.' + sum(Y.^2,2).');
```

Note: In older `MATLAB` versions, the above command raises an error of dimension mismatch since singleton expansion was not supported. Use `bsxfun` instead.

Hint: Use high-performance `BLAS` routines for matrix multiplication. For large values of m, n, block the corpus Y and compute the distance matrix one block at a time, to avoid storing large intermediate distance matrices.

After that, only keep the k shortest distances and the (global!) indices of the corresponding points (using k-select).

V1. Asynchronous parallel

Use your V0 code to run as p `MPI` processes and find all k-NN of the points that are distributed in disjoint blocks to all processes, according to the following spec

```
//! Compute distributed all-kNN of points in X
/*!

  \param  X      Data points                     [n-by-d]
  \param  n      Number of data points           [scalar]
  \param  d      Number of dimensions            [scalar]
  \param  k      Number of neighbors             [scalar]

  \return  The kNN result
*/
knnresult distrAllkNN(double * X int n, int d, int k);
```

We move the data along a ring, (receive from previous and send to the next process) and update the k nearest every time.

- How many times do we need to iterate to have all points checked against all other points?

- Hide all communication costs by transferring data using asynchronous communications, while we compute.
- How the total run time of `V1` compares with the total time of `V0`?
- Assume that you can fit in memory the local points `X` (query), the points `Y` (corpus) you are working with, and space for the incoming (corpus) points `Z`.
- Use pointers to exchange the locations `Y` and `Z`, do not copy!
- In the first iteration, the query points are also the corpus points.

Compare run times and make sure you agree with the `V0` results

What to submit

- A 4-page report in PDF format . Report execution times of your implementations with respect to the number of data points nn$n$, the number of dimensions dd$d$, and the number of processes pp$p$.
- Test with regular grids (3d and 27 neighbors, 4d and 81 neighbors), and real-data sets like MNIST hand-written digits.
- Upload the source code on GitHub, BitBucket, Dropbox, Google Drive, etc. and add a link in your report.
- List any references you use, including ChatGPT!