Exercise 2

Parallel & Distributed Computer Systems

Nov 6, 2019

Implement in `MPI` a distributed search and find 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 an index array that enumerates the points, 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 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 Y the k nearest neighbors in the corpus set X, 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 [n-by-d]
/*!

  \param  X      Corpus data points              [n-by-d]
  \param  Y      Query data points               [m-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 kNN(double * X, double * Y, int n, 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, like in this `MATLAB` line:

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

Hint: Use high-performance `BLAS` routines, for matrix multiplication.

After that, only keep the k shortest distances and the indices of the corresponding points (using k-select like in the previous assignment?).

V1. Synchronous

Use your V0 code to run as p `MPI` processes and find the all kNN 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?
- How much time is spent computing and how much communicating?

V2. Asynchronous

In this version we will hide all communication costs by transferring data using asynchronous communications, while we compute.

- How the total run time of `V2` compares with the total time of `V1`?
- Assume that you can fit in memory the local points `X`, the points `Y` you are working with, and space for the incoming points `Z`.

Global reductions and all-to-all

Extend V2 to also compute the global minimum and maximum distances found on the final kNN distances.

What to submit

- A 3-page report in PDF format (any pages after the 3rd one will not be taken into account). Report execution times of your implementations with respect to the number of data points n and the number of dimensions d.
- Upload the source code on GitHub, BitBucket, Dropbox, Google Drive, etc. and add a link in your report.
- Check the validation of your code using the automated tester on e-learning.

Afternotes

You could have 2 levels of blocking. One level to partition the data in m_1 blocks to the MPI processes and a second level of blocking with blocks size m_2 in each MPI process, to reduce the required amount of total memory per node.