a6/kmeans/kmeans.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

#include "kmeans.h"

// square of Euclid distance between two multi-dimensional points
inline static double euclid_dist_2(int    numdims,  /* no. dimensions */
                                   double * coord1,   /* [numdims] */
                                   double * coord2)   /* [numdims] */
{
    int i;
    double ans = 0.0;

    for(i=0; i<numdims; i++)
        ans += (coord1[i]-coord2[i]) * (coord1[i]-coord2[i]);

    return ans;
}

inline static int find_nearest_cluster(int     numClusters, /* no. clusters */
                                       int     numCoords,   /* no. coordinates */
                                       double * object,      /* [numCoords] */
                                       double * clusters)    /* [numClusters][numCoords]
*/
{
    int index, i;
    double dist, min_dist;

    // find the cluster id that has min distance to object
    index = 0;
    min_dist = euclid_dist_2(numCoords, object, clusters);

    for(i=1; i<numClusters; i++) {
        dist = euclid_dist_2(numCoords, object, &clusters[i*numCoords]);
        // no need square root
        if (dist < min_dist) { // find the min and its array index
            min_dist = dist;
            index    = i;
        }
    }
    return index;
}

void kmeans(double * objects,         /* in: [numObjs][numCoords] */
            int     numCoords,        /* no. coordinates */
            int     numObjs,          /* no. objects */
            int     numClusters,      /* no. clusters */
            double   threshold,       /* minimum fraction of objects that change
membership */
            long    loop_threshold,   /* maximum number of iterations */
            int   * membership,       /* out: [numObjs] */
```

```
51              double * clusters)        /* out: [numClusters][numCoords] */
52  {
53      int i, j;
54      int index, loop=0;
55      double timing = 0;
56
57      /* Every variable has its "rank_" version, which is used to store local data,
58       * and its "new" version, which is used to store global data.
59       */
60      double rank_delta, delta = 0;               // fraction of objects whose clusters
    change in each loop
61      int * rank_newClusterSize, * newClusterSize; // [numClusters]: no. objects assigned in
    each new cluster
62      double * rank_newClusters, *newClusters;     // [numClusters][numCoords]
63
64      // Get rank of this process
65      int rank;
66      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
67
68      // initialize membership
69      for (i=0; i<numObjs; i++)
70          membership[i] = -1;
71
72      // initialize rank_newClusterSize and rank_newClusters to all 0
73      rank_newClusterSize = (typeof(rank_newClusterSize)) calloc(numClusters,
    sizeof(*rank_newClusterSize));
74      rank_newClusters    = (typeof(rank_newClusters))  calloc(numClusters * numCoords,
    sizeof(*rank_newClusters));
75      newClusterSize      = (typeof(newClusterSize)) calloc(numClusters,
    sizeof(*newClusterSize));
76      newClusters         = (typeof(newClusters))  calloc(numClusters * numCoords,
    sizeof(*newClusters));
77
78      timing = wtime();
79      do {
80          // before each loop, set cluster data to 0
81          for (i=0; i<numClusters; i++) {
82              for (j=0; j<numCoords; j++)
83                  rank_newClusters[i*numCoords + j] = 0.0;
84              rank_newClusterSize[i] = 0;
85          }
86
87          rank_delta = 0.0;
88
89          for (i=0; i<numObjs; i++) {
90              // find the array index of nearest cluster center
91              index = find_nearest_cluster(numClusters, numCoords, &objects[i*numCoords],
    clusters);
92
93              // if membership changes, increase rank_delta by 1
94              if (membership[i] != index)
95                  rank_delta += 1.0;
96
97              // assign the membership to object i
```

```
98              membership[i] = index;
99
100             // update new cluster centers : sum of objects located within
101             rank_newClusterSize[index]++;
102             for (j=0; j<numCoords; j++)
103                 rank_newClusters[index*numCoords + j] += objects[i*numCoords + j];
104         }
105
106         /*
107          * TODO: Perform reduction of cluster data (rank_newClusters, rank_newClusterSize)
    from local arrays to shared.
108          */
109         MPI_Allreduce(rank_newClusters, newClusters, numClusters * numCoords, MPI_DOUBLE,
    MPI_SUM, MPI_COMM_WORLD);
110         MPI_Allreduce(rank_newClusterSize, newClusterSize, numClusters, MPI_INT, MPI_SUM,
    MPI_COMM_WORLD);
111
112
113         // average the sum and replace old cluster centers with newClusters
114         for (i=0; i<numClusters; i++) {
115             if (newClusterSize[i] > 0) {
116                 for (j=0; j<numCoords; j++) {
117                     clusters[i*numCoords + j] = newClusters[i*numCoords + j] /
    newClusterSize[i];
118                 }
119             }
120         }
121
122         /*
123          * TODO: Perform reduction from rank_delta variable to delta variable, that will
    be used for convergence check.
124          */
125         MPI_Allreduce(&rank_delta, &delta, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
126
127         // Get fraction of objects whose membership changed during this loop. This is used
    as a convergence criterion.
128         delta /= numObjs;
129
130         loop++;
131         //printf("\r\tcompleted loop %d", loop);
132         //fflush(stdout);
133     } while (delta > threshold && loop < loop_threshold);
134
135     timing = wtime() - timing;
136     if (rank == 0) fprintf(stdout, "        nloops = %3d   (total = %7.4fs)  (per loop =
    %7.4fs)\n", loop, timing, timing/loop);
137
138     free(rank_newClusters);
139     free(rank_newClusterSize);
140     free(newClusters);
141     free(newClusterSize);
142 }
143
```