a2/kmeans/omp_naive_kmeans.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "kmeans.h"
/*
 * TODO: include openmp header file
 */
#include <omp.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 */
```

```
52              double threshold,    /* minimum fraction of objects that change membership */
53              long loop_threshold, /* maximum number of iterations */
54              int *membership,     /* out: [numObjs] */
55              double *clusters)    /* out: [numClusters][numCoords] */
56  {
57      int i, j;
58      int index, loop = 0;
59      double timing = 0;
60
61      double delta;        // fraction of objects whose clusters change in each loop
62      int *newClusterSize; // [numClusters]: no. objects assigned in each new cluster
63      double *newClusters; // [numClusters][numCoords]
64      int nthreads;        // no. threads
65
66      nthreads = omp_get_max_threads();
67      printf("OpenMP Kmeans - Naive\t(number of threads: %d)\n", nthreads);
68
69      // initialize membership
70      for (i = 0; i < numObjs; i++)
71          membership[i] = -1;
72
73      // initialize newClusterSize and newClusters to all 0
74      newClusterSize = (typeof(newClusterSize))calloc(numClusters, sizeof(*newClusterSize));
75      newClusters = (typeof(newClusters))calloc(numClusters * numCoords,
    sizeof(*newClusters));
76
77      timing = wtime();
78
79      do
80      {
81          // before each loop, set cluster data to 0
82          for (i = 0; i < numClusters; i++)
83          {
84              for (j = 0; j < numCoords; j++)
85                  newClusters[i * numCoords + j] = 0.0;
86              newClusterSize[i] = 0;
87          }
88
89          delta = 0.0;
90
91  /*
92   * TODO: Detect parallelizable region and use appropriate OpenMP pragmas
93   */
94  #pragma omp parallel for private(index, j)
95          for (i = 0; i < numObjs; i++)
96          {
97              // find the array index of nearest cluster center
98              index = find_nearest_cluster(numClusters, numCoords, &objects[i * numCoords],
    clusters);
99
100             // if membership changes, increase delta by 1
101             if (membership[i] != index)
102             {
103
```

```
104 #pragma omp atomic // protect update on shared "delta" variable
105                 delta += 1.0;
106             }
107 
108             // assign the membership to object i
109             membership[i] = index;
110 
111 // update new cluster centers : sum of objects located within
112 /*
113  * TODO: protect update on shared "newClusterSize" array
114  */
115 #pragma omp atomic
116             newClusterSize[index]++;
117             for (j = 0; j < numCoords; j++)
118 /*
119  * TODO: protect update on shared "newClusters" array
120  */
121 #pragma omp atomic
122                 newClusters[index * numCoords + j] += objects[i * numCoords + j];
123         }
124 
125         // average the sum and replace old cluster centers with newClusters
126         for (i = 0; i < numClusters; i++)
127         {
128             if (newClusterSize[i] > 0)
129             {
130                 for (j = 0; j < numCoords; j++)
131                 {
132                     clusters[i * numCoords + j] = newClusters[i * numCoords + j] /
    newClusterSize[i];
133                 }
134             }
135         }
136 
137         // Get fraction of objects whose membership changed during this loop. This is used
    as a convergence criterion.
138         delta /= numObjs;
139 
140         loop++;
141         printf("\r\tcompleted loop %d", loop);
142         fflush(stdout);
143     } while (delta > threshold && loop < loop_threshold);
144     timing = wtime() - timing;
145     printf("\n        nloops = %3d   (total = %7.4fs)  (per loop = %7.4fs)\n", loop,
    timing, timing / loop);
146 
147     free(newClusters);
148     free(newClusterSize);
149 }
150 
```