a6/kmeans/main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>     /* strtok() */
#include <sys/types.h>  /* open() */
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>     /* getopt() */
#include <mpi.h>

int _debug;
#include "kmeans.h"

static void usage(char *argv0) {
    char *help =
        "Usage: %s [switches]\n"
        "       -c num_clusters    : number of clusters (must be > 1)\n"
        "       -s size            : size of examined dataset\n"
        "       -n num_coords      : number of coordinates\n"
        "       -t threshold       : threshold value (default : 0.001)\n"
        "       -l loop_threshold  : iterations threshold (default : 10)\n"
        "       -d                 : enable debug mode\n"
        "       -h                 : print this help information\n";
    fprintf(stderr, help, argv0);
    exit(-1);
}

int main(int argc, char **argv)
{
    long i, j, opt;
    extern char* optarg;
    extern int optind;

    long    numClusters=0, numCoords=0, numObjs=0;
    long    rank_numObjs=0;
    int   * membership;     // [rank_numObjs] this array will contain membership information for this rank's objects
    int   * tot_membership; // [numObjs]      this array will contain membership information for all objects
    double * objects;       // [numObjs * numCoords] data  objects
    double * clusters;      // [numClusters * numCoords] cluster center
    double   dataset_size = 0, threshold;
    long    loop_threshold;
    double  io_timing_read;

    /* some default values */
    _debug         = 0;
    threshold      = 0.001;
    loop_threshold = 10;
    numClusters    = 0;

    while ( (opt = getopt(argc,argv,"n:t:l:c:s:dh")) != EOF) {
        switch (opt) {
```

```
51              case 'c': numClusters = atol(optarg);
52                        break;
53              case 't': threshold=atof(optarg);
54                        break;
55              case 'l': loop_threshold=atol(optarg);
56                        break;
57              case 's': dataset_size=atof(optarg);
58                        break;
59              case 'n': numCoords=atol(optarg);
60                        break;
61              case 'd': _debug = 1;
62                        break;
63              case 'h':
64              default: usage(argv[0]);
65                        break;
66          }
67      }
68      if (numClusters <= 1) {
69          usage(argv[0]);
70      }
71
72      int rank, size;
73      MPI_Init(&argc,&argv);
74      MPI_Comm_rank(MPI_COMM_WORLD,&rank);
75      MPI_Comm_size(MPI_COMM_WORLD,&size);
76
77      numObjs = (dataset_size*1024*1024) / (numCoords*sizeof(double));
78
79      if (numObjs < numClusters) {
80          if (rank == 0) printf("Error: number of clusters must be larger than the number of
    data points to be clustered.\n");
81          MPI_Finalize();
82          return 1;
83      }
84      if (rank == 0) printf("dataset_size = %.2f MB    numObjs = %ld    numCoords = %ld
    numClusters = %ld\n", dataset_size, numObjs, numCoords, numClusters);
85
86      objects = dataset_generation(numObjs, numCoords, &rank_numObjs);
87
88      // Allocate space for clusters (coordinates of cluster centers)
89      clusters = (double*)  malloc(numClusters * numCoords * sizeof(double));
90
91      // The first numClusters elements are selected as initial centers. Only rank 0 needs
    to calculate this, and later broadcast it to all ranks.
92      if (rank == 0) {
93          for (i=0; i<numClusters; i++)
94              for (j=0; j<numCoords; j++)
95                  clusters[i*numCoords + j] = objects[i*numCoords + j];
96
97          // check initial cluster centers for repetition
98          if (check_repeated_clusters(numClusters, numCoords, clusters) == 0) {
99              printf("Error: some initial clusters are repeated. Please select distinct
    initial centers\n");
100             MPI_Finalize();
```

```
101             return 1;
102         }
103         /*
104         printf("Initial cluster centers:\n");
105         for (i=0; i<numClusters; i++) {
106             printf("(0) clusters[%ld] =",i);
107             for (j=0; j<numCoords; j++)
108                 printf(" %6.6f", clusters[i*numCoords + j]);
109             printf("\n");
110         }
111         */
112     }
113 
114     /*
115      * TODO: Broadcast initial cluster positions to all ranks
116      */
117     MPI_Bcast(clusters, numClusters * numCoords, MPI_DOUBLE, 0, MPI_COMM_WORLD);
118 
119 
120     // membership: the cluster id for each data object
121     membership = (int*) malloc(rank_numObjs * sizeof(int));
122     tot_membership = (int*) malloc(numObjs * sizeof(int));
123 
124     // start the core computation
125     /*
126      * TODO: Fix number of objects that this kmeans function call will process
127      */
128     kmeans(objects, numCoords, rank_numObjs, numClusters, threshold, loop_threshold,
    membership, clusters);
129 
130     /*
131     if (rank == 0) {
132         printf("Final cluster centers:\n");
133         for (i=0; i<numClusters; i++) {
134             printf("clusters[%ld] = ",i);
135             for (j=0; j<numCoords; j++)
136                 printf("%6.6f ", clusters[i*numCoords + j]);
137             printf("\n");
138         }
139     }
140     */
141 
142     // Gather membership information from all ranks to tot_membership
143     int recvcounts[size], displs[size];
144     if (rank == 0) {
145         /* TODO: Calculate recvcounts and displs, which will be used to gather data from
    each rank.
146          * Hint: recvcounts: number of elements received from each rank
147          *       displs: displacement of each rank's data
148          */
149         int sum_disp = 0;
150         int remainder = numObjs % size;
151 
152         for (j = 0; j < size; j++) {
```

```
153             // Υπολογισμός πόσα αντικείμενα περιμένουμε από το rank 'j'
154             int j_numObjs = numObjs / size;
155             if (j < remainder) {
156                 j_numObjs++;
157             }
158 
159             recvcounts[j] = j_numObjs; // Εδώ είναι σκέτα αντικείμενα (int)
160             displs[j] = sum_disp;
161             sum_disp += recvcounts[j];
162         }
163     }
164 
165     /*
166      * TODO: Broadcast the recvcounts and displs arrays to other ranks.
167      */
168     MPI_Bcast(recvcounts, size, MPI_INT, 0, MPI_COMM_WORLD);
169     MPI_Bcast(displs, size, MPI_INT, 0, MPI_COMM_WORLD);
170 
171 
172     /*
173      * TODO: Gather membership information from every rank. (hint: each rank may send different number of objects)
174      */
175     MPI_Gatherv(membership, rank_numObjs, MPI_INT, tot_membership, recvcounts, displs, MPI_INT, 0, MPI_COMM_WORLD);
176 
177 
178     if (_debug && rank == 0)
179         for (i = 0; i < numObjs; ++i)
180             fprintf(stderr, "%d\n", tot_membership[i]);
181 
182     free(objects);
183     free(membership);
184     free(tot_membership);
185     free(clusters);
186 
187     MPI_Finalize();
188     return 0;
189 }
190 
```