**a5/cuda_kmeans_shared.cu**

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

#include "kmeans.h"
#include "alloc.h"
#include "error.h"

#ifdef __CUDACC__
inline void checkCuda(cudaError_t e) {
    if (e != cudaSuccess) {
        // cudaGetErrorString() isn't always very helpful. Look up the error
        // number in the cudaError enum in driver_types.h in the CUDA includes
        // directory for a better explanation.
        error("CUDA Error %d: %s\n", e, cudaGetErrorString(e));
    }
}

inline void checkLastCudaError() {
    checkCuda(cudaGetLastError());
}
#endif

__device__ int get_tid() {
  return blockIdx.x * blockDim.x + threadIdx.x;
}

/* square of Euclid distance between two multi-dimensional points using column-base format
*/
__host__ __device__ inline static
double euclid_dist_2_transpose(int numCoords,
                               int numObjs,
                               int numClusters,
                               double *objects,     // [numCoords][numObjs]
                               double *clusters,    // [numCoords][numClusters]
                               int objectId,
                               int clusterId) {
  int i;
  double ans = 0.0;

  /* TODO: Calculate the euclid_dist of elem=objectId of objects from elem=clusterId from
clusters, but for column-base format!!! */
  for (i = 0; i < numCoords; i++) {
    double objectVal = objects[i * numObjs + objectId];
    double clusterVal = clusters[i * numClusters + clusterId];

    double diff = objectVal - clusterVal;
    ans += diff * diff;
  }

  return (ans);
}

```

```
__global__ static
void find_nearest_cluster(int numCoords,
                          int numObjs,
                          int numClusters,
                          double *objects,           //  [numCoords][numObjs]
                          double *deviceClusters,    //  [numCoords][numClusters]
                          int *deviceMembership,          //  [numObjs]
                          double *devdelta) {
  extern __shared__ double shmemClusters[];

  // TODO: Copy deviceClusters to shmemClusters so they can be accessed faster.
  int tid_in_block = threadIdx.x;      // Το ID του νήματος μέσα στο Block
  int block_size = blockDim.x;         // Πόσα νήματα έχει το Block
  int total_cluster_doubles = numClusters * numCoords; // Συνολικά νούμερα προς αντιγραφή

  // Κάθε νήμα αντιγράφει όσα στοιχεία του αναλογούν (με βήμα block_size)
  for (int k = tid_in_block; k < total_cluster_doubles; k += block_size) {
      shmemClusters[k] = deviceClusters[k];
  }

  /* Συγχρονισμός (BARRIER) */

  __syncthreads();

  /* Get the global ID of the thread. */
  int tid = get_tid();

  /* TODO: Maybe something is missing here... should all threads run this? */
  if (tid < numObjs) {
    int index, i;
    double dist, min_dist;

    /* find the cluster id that has min distance to object */
    index = 0;
    /* TODO: call min_dist = euclid_dist_2(...) with correct objectId/clusterId using
clusters in shmem*/


    min_dist = euclid_dist_2_transpose(numCoords, numObjs, numClusters,
                            objects, shmemClusters,
                            tid, index);

    for (i = 1; i < numClusters; i++) {
      dist = euclid_dist_2_transpose(numCoords, numObjs, numClusters,
                           objects, shmemClusters,
                           tid, i);

        /* no need square root */
      if (dist < min_dist) { /* find the min and its array index */
          min_dist = dist;
          index = i;
      }
    }

```

```
104     if (deviceMembership[tid] != index) {
105       /* TODO: Maybe something is missing here... is this write safe? */
106       atomicAdd(devdelta, 1.0);
107     }
108
109     /* assign the deviceMembership to object objectId */
110     deviceMembership[tid] = index;
111   }
112 }
113
114 //
115 //  -----------------------------------------
116 //  DATA LAYOUT
117 //
118 //  objects         [numObjs][numCoords]
119 //  clusters        [numClusters][numCoords]
120 //  dimObjects      [numCoords][numObjs]
121 //  dimClusters     [numCoords][numClusters]
122 //  newClusters     [numCoords][numClusters]
123 //  deviceObjects   [numCoords][numObjs]
124 //  deviceClusters  [numCoords][numClusters]
125 //  -----------------------------------------
126 //
127 /* return an array of cluster centers of size [numClusters][numCoords]       */
128 void kmeans_gpu(double *objects,      /* in: [numObjs][numCoords] */
129                 int numCoords,    /* no. features */
130                 int numObjs,      /* no. objects */
131                 int numClusters,  /* no. clusters */
132                 double threshold,    /* % objects change membership */
133                 long loop_threshold,   /* maximum number of iterations */
134                 int *membership,   /* out: [numObjs] */
135                 double *clusters,   /* out: [numClusters][numCoords] */
136                 int blockSize) {
137   double timing = wtime(), timing_internal, timer_min = 1e42, timer_max = 0;
138   double timing_gpu, timing_cpu, timing_transfers, transfers_time = 0.0, cpu_time = 0.0,
    gpu_time = 0.0;
139   int loop_iterations = 0;
140   int i, j, index, loop = 0;
141   int *newClusterSize; /* [numClusters]: no. objects assigned in each
142                                 new cluster */
143   double delta = 0, *dev_delta_ptr;          /* % of objects change their clusters */
144   /* TODO: Copy me from transpose version*/
145   double **dimObjects = (double **) calloc_2d(numCoords, numObjs, sizeof(double));
    //calloc_2d(...) -> [numCoords][numObjs]
146   double **dimClusters = (double **) calloc_2d(numCoords, numClusters, sizeof(double));
    //calloc_2d(...) -> [numCoords][numClusters]
147   double **newClusters = (double **) calloc_2d(numCoords, numClusters, sizeof(double));
    //calloc_2d(...) -> [numCoords][numClusters]
148
149   double *deviceObjects;
150   double *deviceClusters;
151   int *deviceMembership;
152
153   printf("\n|-----------Shared GPU Kmeans------------|\n\n");
```

```
154
155   /* TODO: Copy me from transpose version*/
156   for (i=0 ; i <numObjs; i++){
157     for (j=0; j<numCoords; j++){
158       dimObjects[j][i]=objects[i*numCoords + j];
159     }
160   }
161
162   /* pick first numClusters elements of objects[] as initial cluster centers*/
163   for (i = 0; i < numCoords; i++) {
164     for (j = 0; j < numClusters; j++) {
165       dimClusters[i][j] = dimObjects[i][j];
166     }
167   }
168
169   /* initialize membership[] */
170   for (i = 0; i < numObjs; i++) membership[i] = -1;
171
172   /* need to initialize newClusterSize and newClusters[0] to all 0 */
173   newClusterSize = (int *) calloc(numClusters, sizeof(int));
174   assert(newClusterSize != NULL);
175
176   timing = wtime() - timing;
177   printf("t_alloc: %lf ms\n\n", 1000 * timing);
178   timing = wtime();
179   const unsigned int numThreadsPerClusterBlock = (numObjs > blockSize) ? blockSize :
numObjs;
180   const unsigned int numClusterBlocks = (numObjs + numThreadsPerClusterBlock - 1) /
numThreadsPerClusterBlock;/* TODO: Calculate Grid size, e.g. number of blocks. */
181
182   /*    Define the shared memory needed per block.
183       - BEWARE: We can overrun our shared memory here if there are too many
184       clusters or too many coordinates!
185       - This can lead to occupancy problems or even inability to run.
186       - Your exercise implementation is not requested to account for that (e.g. always
assume deviceClusters fit in shmemClusters */
187   const unsigned int clusterBlockSharedDataSize = numClusters*numCoords*sizeof(double);
188
189   cudaDeviceProp deviceProp;
190   int deviceNum;
191   cudaGetDevice(&deviceNum);
192   cudaGetDeviceProperties(&deviceProp, deviceNum);
193
194   if (clusterBlockSharedDataSize > deviceProp.sharedMemPerBlock) {
195     error("Your CUDA hardware has insufficient block shared memory to hold all cluster
centroids\n");
196   }
197
198   checkCuda(cudaMalloc(&deviceObjects, numObjs * numCoords * sizeof(double)));
199   checkCuda(cudaMalloc(&deviceClusters, numClusters * numCoords * sizeof(double)));
200   checkCuda(cudaMalloc(&deviceMembership, numObjs * sizeof(int)));
201   checkCuda(cudaMalloc(&dev_delta_ptr, sizeof(double)));
202
203   timing = wtime() - timing;
```

```
204   printf("t_alloc_gpu: %lf ms\n\n", 1000 * timing);
205   timing = wtime();
206
207   checkCuda(cudaMemcpy(deviceObjects, dimObjects[0],
208                        numObjs * numCoords * sizeof(double), cudaMemcpyHostToDevice));
209   checkCuda(cudaMemcpy(deviceMembership, membership,
210                        numObjs * sizeof(int), cudaMemcpyHostToDevice));
211   timing = wtime() - timing;
212   printf("t_get_gpu: %lf ms\n\n", 1000 * timing);
213   timing = wtime();
214
215   do {
216     timing_internal = wtime();
217
218     /* GPU part: calculate new memberships */
219
220     timing_transfers = wtime();
221     // TODO: Copy clusters to deviceClusters
222     checkCuda(cudaMemcpy(deviceClusters, dimClusters[0],
223                          numClusters * numCoords * sizeof(double),
224                          cudaMemcpyHostToDevice));
225
226     transfers_time += wtime() - timing_transfers;
227
228     checkCuda(cudaMemset(dev_delta_ptr, 0, sizeof(double)));
229
230     timing_gpu = wtime();
231     //printf("Launching find_nearest_cluster Kernel with grid_size = %d, block_size = %d,
    shared_mem = %d KB\n", numClusterBlocks, numThreadsPerClusterBlock, clusterBlockSharedDa-
    taSize/1000);
232     find_nearest_cluster
233     <<< numClusterBlocks, numThreadsPerClusterBlock, clusterBlockSharedDataSize >>>
234             (numCoords, numObjs, numClusters,
235              deviceObjects, deviceClusters, deviceMembership, dev_delta_ptr);
236
237     cudaDeviceSynchronize();
238     checkLastCudaError();
239     gpu_time += wtime() - timing_gpu;
240     //printf("Kernels complete for itter %d, updating data in CPU\n", loop);
241
242     timing_transfers = wtime();
243
244     checkCuda(cudaMemcpy(membership, deviceMembership,
245                          numObjs * sizeof(int),
246                          cudaMemcpyDeviceToHost));
247
248
249     checkCuda(cudaMemcpy(&delta, dev_delta_ptr,
250                          sizeof(double),
251                          cudaMemcpyDeviceToHost));
252
253     transfers_time += wtime() - timing_transfers;
254
255     /* CPU part: Update cluster centers*/
```

```
256
257     timing_cpu = wtime();
258     for (i = 0; i < numObjs; i++) {
259       /* find the array index of nestest cluster center */
260       index = membership[i];
261
262       /* update new cluster centers : sum of objects located within */
263       newClusterSize[index]++;
264       for (j = 0; j < numCoords; j++)
265         newClusters[j][index] += objects[i * numCoords + j];
266     }
267
268     /* average the sum and replace old cluster centers with newClusters */
269     for (i = 0; i < numClusters; i++) {
270       for (j = 0; j < numCoords; j++) {
271         if (newClusterSize[i] > 0)
272           dimClusters[j][i] = newClusters[j][i] / newClusterSize[i];
273         newClusters[j][i] = 0.0;   /* set back to 0 */
274       }
275       newClusterSize[i] = 0;   /* set back to 0 */
276     }
277
278     delta /= numObjs;
279     //printf("delta is %f - ", delta);
280     loop++;
281     //printf("completed loop %d\n", loop);
282     cpu_time += wtime() - timing_cpu;
283
284     timing_internal = wtime() - timing_internal;
285     if (timing_internal < timer_min) timer_min = timing_internal;
286     if (timing_internal > timer_max) timer_max = timing_internal;
287   } while (delta > threshold && loop < loop_threshold);
288
289   /*TODO: Update clusters using dimClusters. Be carefull of layout!!!
clusters[numClusters][numCoords] vs dimClusters[numCoords][numClusters] */
290   for (i = 0; i < numClusters; i++) {
291       for (j = 0; j < numCoords; j++) {
292           clusters[i * numCoords + j] = dimClusters[j][i];
293       }
294   }
295
296   timing = wtime() - timing;
297   printf("nloops = %d  : total = %lf ms\n\t-> t_loop_avg = %lf ms\n\t-> t_loop_min = %lf
ms\n\t-> t_loop_max = %lf ms\n\t"
298          "-> t_cpu_avg = %lf ms\n\t-> t_gpu_avg = %lf ms\n\t-> t_transfers_avg = %lf
ms\n\n|-----------------------------------------|\n",
299          loop, 1000 * timing, 1000 * timing / loop, 1000 * timer_min, 1000 * timer_max,
300          1000 * cpu_time / loop, 1000 * gpu_time / loop, 1000 * transfers_time / loop);
301
302   char outfile_name[1024] = {0};
303   sprintf(outfile_name, "Execution_logs/silver1-V100_Sz-%lu_Coo-%d_Cl-%d.csv",
304           numObjs * numCoords * sizeof(double) / (1024 * 1024), numCoords, numClusters);
305   FILE *fp = fopen(outfile_name, "a+");
```

```
306   if (!fp) error("Filename %s did not open succesfully, no logging performed\n",
      outfile_name);
307   fprintf(fp, "%s,%d,%lf,%lf,%lf\n", "Shmem", blockSize, timing / loop, timer_min,
      timer_max);
308   fclose(fp);
309
310   checkCuda(cudaFree(deviceObjects));
311   checkCuda(cudaFree(deviceClusters));
312   checkCuda(cudaFree(deviceMembership));
313
314   free(dimObjects[0]);
315   free(dimObjects);
316   free(dimClusters[0]);
317   free(dimClusters);
318   free(newClusters[0]);
319   free(newClusters);
320   free(newClusterSize);
321
322   return;
323 }
324
325
```