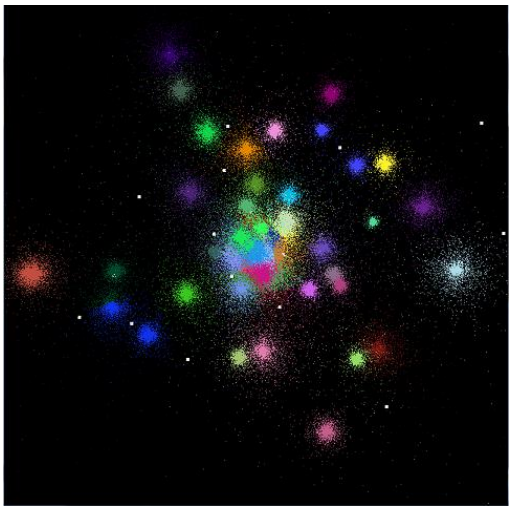
GPU Programming

TP5 – Kmeans

In the field of data science, the Kmeans algorithm is a simple approach that solves the well-known clustering problem: partition the points of a point cloud into k distinct clusters (k predefined) such that each data point belongs to the cluster with the nearest mean value.



This is the Kmeans pseudo-code for N data points and K clusters:

```
points[N] = array of data points (fixed)
centroids[K] = array of centroids
pointlabel[N] = array of point memberships
// pointlabel[i] = k means that point i belongs to cluster k

Procedure Kmeans
      initialize the K centroids
      while (centroid convergence not satisfying)

            // phase 1 (“assignment”):
            // assign each data point to the closest centroid
            for i = 0 to N-1
                  for j = 0 to K-1
                        distance = |points[i]- centroids[j]|
                        if (distance < dmin)
                              dmin = distance
                              n = j
                  pointlabel[i] = n // point i assigned to cluster n

            // phase 2 (“reduction”): recompute centroids
            for j=0 to K-1
                  newcentroids[j] = 0;
                  newcentroidSize[j] = 0;
            for i=0 to N-1
                  newcentroids[pointlabel[i]] =
                    newcentroids[pointlabel[i]] + points[i]
                  newcentroidSize[pointlabel[i]]++
            for j=0 to K-1
                  centroids[j] = newcentroids[j] / newcentroidSize[j]
      end while
```

Execute the provided base code.

- A sample data point cloud has been generated.
- The cluster centroids are already initialized but not used.
- Each cluster has its own random color.
- The current classification algorithm stupidly assigns the data point i to the cluster i%K.
- Current GPU mode = CPU mode

Exercices (1 point per question)

1) Write a CPU version of the Kmeans algorithm.
2) GPU version 1: write a kernel kernelAssign where phase 1 (assignment) is executed on the GPU. Copy the resulting pointlabel array back to the host and continue phase 2 (reduction) on the CPU.
3) GPU version 2: Improve kernelAssign so that it computes the array of point colors at the same time, and copy it back to the host
4) GPU version 3: Copy the array of centroids and the array of centroid colors to the constant memory instead of the global memory.
5) GPU version 4: write a second kernel, kernelReduce where phase 2 (reduction) is executed on the GPU and execute kernelAssign followed by kernelReduce