1. Parallelization of the k-medoids Algorithm and Problem Decomposition

The k-medoids algorithm was parallelized by dividing the process into two main tasks: assigning points to clusters and updating medoids. These two tasks are executed iteratively until the assignments converge or a maximum number of iterations is reached.

The first task, implemented in the assignClusters kernel, assigns each point to its closest medoid by calculating the squared Euclidean distance between the point and all medoids. Since the distance calculations for different points are independent of each other, the problem was parallelized such that each thread handles one point. Each thread iterates through all medoids, finds the one with the smallest distance, and updates the cluster assignment if it changes. An atomic operation is used to flag whether any point's cluster assignment has changed during the iteration.

The second task, implemented in the updateMedoids kernel, recalculates the medoids for each cluster. For each medoid, the kernel finds the point within the cluster that minimizes the total distance to all other points in the cluster. The update for each medoid is independent, so the kernel processes one medoid per block. Within each block, threads collaborate to compute the total distance from a candidate point to all other points assigned to the same cluster. This ensures that the medoid updates can also be parallelized effectively.

By alternating between these two tasks, the k-medoids algorithm efficiently leverages parallelism to reduce computation time.

2. Elements Processed and Computations per Thread

In the assignClusters kernel, each thread processes a single data point. The thread calculates the squared distance between the point and all K medoids,

where K is the number of clusters and D is the dimensionality of the data. For each medoid, the thread computes the sum of squared differences over D dimensions, resulting in approximately K multiplied by D operations. After identifying the closest medoid, the thread updates the assignment and uses an atomic operation to flag changes if the cluster assignment has been updated.

In the updateMedoids kernel, each block is responsible for processing one cluster (or medoid). Within the block, threads calculate the total distance between a candidate point and all other points in the cluster. Each thread iterates over the points assigned to the cluster, computes the pairwise distance, and accumulates the total cost. For a cluster with C points and dimensionality D, this involves approximately C squared multiplied by D operations per medoid. The threads then use shared memory to track the candidate with the minimum total distance and update the medoid coordinates accordingly.

This approach ensures a balanced distribution of the workload, with threads in the assignClusters kernel processing points independently and threads in the updateMedoids kernel collaborating within blocks to update medoids.

3. Thread Hierarchy

The thread hierarchy for this implementation uses one-dimensional thread blocks and grids. In the assignClusters kernel, each thread processes a single point in the dataset. The number of thread blocks is determined dynamically based on the total number of points N and the number of threads per block. The formula for calculating the number of blocks is as follows:

Grid size = (number of points + threads per block - 1) / threads per block

This ensures that all points are processed concurrently, with each thread

handling the assignment of one point to its closest medoid.

In the updateMedoids kernel, each block processes one cluster or medoid. The number of blocks is set to the total number of clusters K, and threads within each block collaborate to compute the total distance for candidate medoids. The number of threads per block is specified by the user, and the workload for each block involves iterating through all points assigned to the cluster to find the optimal new medoid.

The use of one-dimensional thread blocks and grids simplifies the implementation, as the tasks of distance computation and medoid updates map naturally to linear data structures. This design also ensures efficient memory access patterns and makes full use of the GPU's parallel processing capabilities.

```
joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 256 -1 1
k-medoids clustering time: 34.3623s
```

```
joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 256 -1 2
k-medoids clustering time: 33.0418s
```

```
joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 256 -1 4
k-medoids clustering time: 31.1362s
```

```
joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 256 -1 8
k-medoids clustering time: 27.4115s
```

```
joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 256 -1 16
k-medoids clustering time: 22.2362s
```

```
joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 512 -1 1
k-medoids clustering time: 24.8724s
```

```
joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 512 -1 2
k-medoids clustering time: 23.8173s

joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 512 -1 4
k-medoids clustering time: 22.7804s

joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 512 -1 8
k-medoids clustering time: 20.4676s

joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 512 -1 16
k-medoids clustering time: 16.7571s

joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 1024 -1 1
k-medoids clustering time: 6.9519s

joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 1024 -1 2
k-medoids clustering time: 6.1081s

joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 1024 -1 4
k-medoids clustering time: 5.8221s

joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 1024 -1 8
k-medoids clustering time: 5.4536s

joo00032@csel-cuda-04:/home/joo00032 $ ./km_cuda /export/scratch/CSCI-5451/assignme
nt-3/small_cpd.txt 1024 -1 16
k-medoids clustering time: 4.8429s
```