Accelerating K-Means using GPU and Multi-core

-Topics in Parallel Processing, Monsoon 2011, IIIT-H
K-means Clustering

- **k-means clustering** (also referred as Lloyd’s Algorithm) is a method of cluster analysis which aims to partition \( n \) observations into \( k \) clusters in which each observation belongs to the cluster with the nearest mean.

- It is an unsupervised machine learning Algorithm.

- Given dataset of \( N \) vectors of \( D \) dimensions, and \( K \) initial centers, the algorithm must divide the dataset into \( K \) clusters such that nearest mean of every sample in the \( i^{th} \) cluster is \( \mu_i \).

**Algorithm:**

- Step 1) Compute Initial means by K-means++ Algorithm or randomly
- Step 2) Assign Class label to each sample by finding the nearest mean. (Euclidean distance)
- Step 3) Check the Convergence condition
  - If Convergence achieved, current labels are finalised
  - If Convergence not achieved, Compute new means for each class and repeat Step 1 and 2

Convergence condition :- The algorithm is converged when the labels no longer change between consecutive iterations.
K-means Example
Accelerating K-means

- Clustering large numbers of high-dimensional vectors is very computation intensive.
- In documents clustering and Computer Vision techniques, the vectors are typically 128 dimensional or even more.
- K-means has data parallelism within each data-item (vector) as well as among different data-items.

**Previous Work :-**

- There are approaches for K-means on GPU
- Every thread is associated with a data-object, which sequentially evaluates its label. Evaluation of new means was done entirely on CPU.
- Mean Evaluation was also shifted to GPU.
- Texture Memory of GPU was used to store input vectors, but size of texture memory was limited
- The approaches exploited the parallelism of multiple data-items only. Dimensions of a vector can be treated independently as well.
Implementation

- The implementation is divided into two parts
  - Membership evaluation
  - Checking convergence and Mean evaluation

- We use GPU for membership evaluation. Use 2D block of threads where each thread in a block is associated with a fixed dimension of some vector.

- If there are $N$ vectors of $d$ dimension, we give $q$ vectors to each block. The dimensions of block become $q \times d$

![Diagram](image)

Figure 2. $d$ Dimensional Input data stored in row major format.
Membership Evaluation

- We need to find distance of each vector from the centers.
- Component-wise distances are evaluated and stored in memory. The final distance, which is the sum of component-wise distances is evaluated using a log reduction of these values.
- The label is assigned by finding the minimum of distances from each center.

**Algorithm**

- **Kernel for Thread (tx,ty)**
  - For each center i:
    - Find the distance between ‘ty’ component of center and the vector
    - Synchronize all threads
    - Do Log reduction for finding the final distance
    - If distance < current minimum, assign label ‘i’ to the vector and update current minimum distance

- Transfer control to CPU, which then checks the convergence condition and computes new means
- If no convergence, transfer new means to GPU and execute the kernel
Results

- When 100 thousand vectors of 128 dimension are clustered in 100 sets
  - Time taken to execute 1 iteration of k-means = **around 1 sec**
  - Total time taken = **around 3.5 minutes**
  - % of time taken by GPU = **95.5%**
  - % of time taken by CPU = **4.5%**

- When 100 thousand vectors of 128 dimension are clustered in 1000 sets
  - Time taken to execute 1 iteration of k-means = **around 10.6 sec**
  - Total time taken = **around 10 minutes**
  - % of time taken by GPU = **99.5%**
  - % of time taken by CPU = **0.5%**

- As we can see, most of the computation time is taken by GPU for membership evaluation. The global memory access for vectors, distances, centers and labels is the overhead.

- We use **shared memory** for storing vectors, component wise distances, labels and minimum distances. Centers can also be kept in shared memory, but since they are constant and common to all the threads, we access them through global memory.
Results (using shared memory)

- When 100 thousand vectors of 128 dimension are clustered in 100 sets
  - Time taken to execute 1 iteration of k-means = around 0.34 sec
  - Total time taken = around 1.2 minutes
  - % of time taken by GPU = 87% (decreased)
  - % of time taken by CPU = 13% (increased)

- When 100 thousand vectors of 128 dimension are clustered in 1000 sets
  - Time taken to execute 1 iteration of k-means = around 3 sec
  - Total time taken = around 3 minutes
  - % of time taken by GPU = 98.5%
  - % of time taken by CPU = 1.5%

- On using shared memory, the performance has increased roughly by 3 times

- For 100 thousand vectors of 128 dimension and K=1000, sequential CPU version takes 100 minutes to cluster them!!

- Also, the % of time taken by GPU has decreased, which indicates optimization of functions executed by CPU, i.e., new means computation and convergence checking.
Large Datasets

- For large datasets like 500 thousand vectors of 128 dimensions and $K=4000$
  
  - Time per iteration: around 65 sec
  - % of time taken by CPU: 0.4 %
  - % of time taken by GPU: 99.6 %

All together:

<table>
<thead>
<tr>
<th>$N$</th>
<th>$K$</th>
<th>$D$</th>
<th>Time per iteration</th>
<th>% GPU</th>
<th>% CPU</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>100</td>
<td>128</td>
<td>0.34 sec</td>
<td>87 %</td>
<td>13 %</td>
<td>1.2 min</td>
</tr>
<tr>
<td>100000</td>
<td>1000</td>
<td>128</td>
<td>3 sec</td>
<td>98.5 %</td>
<td>1.5 %</td>
<td>3 min</td>
</tr>
<tr>
<td>500000</td>
<td>4000</td>
<td>128</td>
<td>65 sec</td>
<td>99.6 %</td>
<td>0.4 %</td>
<td>NA</td>
</tr>
</tbody>
</table>
Further Work

- Operations done by CPU can be optimized in order to reduce the time taken by CPU

- There are two possible approaches:
  - CPU can work in parallel with the GPU. When GPU labels some vectors, CPU must add them to the corresponding class's mean. By the time GPU does labeling all the vectors, the means completion for all classes must be computed.
  - At the same time, convergence can be checked. If Vector changed its class, add 1 to number of reclassified vectors. If reclassified > 0, Convergence not reached.

- GPU can do the mean calculation itself. Each thread after labeling a particular vector, can add that vector to the corresponding mean.
  - **Advantage**: Only one extra addition in the current kernel. Since a thread is responsible for a single dimension only, it will add the value in that dimension to the center's dimension.
  - **Disadvantage**: Since all the threads will be accessing the same memory of centers, it must be atomic. Synchronization issues.

- But in both these approaches, the time taken by CPU in current scenario can at max be reduced to zero. May not lead to significant gain in performance.