Speeding up Generalized Fuzzy k-Means Clustering Algorithm by GPUs

Dinh Trung Vu

Department of Computer Science and Engineering

National Taiwan Ocean University

Keelung, Taiwan 202

R. O. C.

ABSTRACT

The graphics hardware is becoming increasingly more powerful and programmable with the introduction of Graphics Processing Units (GPU) like the NVidia GeForce series. The GPU’s exceed the ordinary general purpose CPU’s ability to do ﬂoating point operations due to the massively parallel architecture in the GPU’s.

With the newest GPU’s one actually have enough programmable freedom to do other computations than computer graphics processing. This project will take advantage of this in order to get high performance implementations of image analysis algorithms.

In this project we will implement an image analysis algorithm, which is Generalized Fuzzy k-Means Clustering Using m nearest Cluster Centers (GFKM) [1], on a GPU. We will also make comparisons with CPU based implementations and analysis the pros and cons of using GPU’s in image analysis.

I. INTRODUCTION

II. GFKM Clustering Algorithm

1. Input an initial set of cluster centers *SC*0 = {**C***j*(0)} and the values of ε and *M*. Set *p* = 0. Let , *NNTi*, and *DNNTi* responding to the squared Euclidean distance between **X***i* and **C***j*, the set of *M* nearest cluster centers for the data point**,** and the set of *M* corresponding shortest distances for the data point****, calculateand initialize *NNTi* and *DNNTi*.
2. Given the set of cluster centers *SCp*, update membership **** using equation (1). If **C***j*∈*NNTi* is the *l*th nearest neighbor of **X***i*, set  = ****; otherwise let  = 0.

**** = , for *r* = 1 to *N* and *s* = 1 to *M* (1)

1. Compute the center for each cluster using equation (2) to obtain a new set of cluster representatives *SCp+*1 = {**C***j*(*p*+1)}.

**C***j* =  , for S *j* = {**X** *i*: **X** *i*∈ *NNTj*, *i* = 1 to *N*} (2)

1. Calculate, update *NNTi* and *DNNTi* for *i* = 1 to *N*, and calculate distortion value *J* using equation (3).

*J* =  (3)

1. If < ε, then stop, where ε > 0 is a very small positive number. Otherwise set *p = p + 1* and go to step (2).

The computational complexity of GFKM is also O(*Nkt*), where *t* is the number of iterations.

Algorithm 1: CPU-based GFKM

//

//

//

//



III. Design a GPU-based parallel GFKM algorithm

Analysis steps of the algorithm can be parallel implemented on GPUs: (1) calculate  andinitializing *NNTi* and *DNNTi*, (2) updating memberships **** and , (3) computing the new center for each clusters, (4) calculate  andupdating *NNTi* and *DNNTi*, (5) calculating new distortion value *newJ*.

Design GPU-based parallel algorithm for each section:

* Step (1), (2), and (4): We utilize the GPU on-chip registers to minimize the latency of data access [2].
* Step (3): It is difficult to be fully parallelized due to write conflict, so this task is executed on CPU.
* Step (5): We use the parallel reduction algorithm for this step.

A. Calculating  andinitializing *NNTi* and *DNNTi*

Algorithm 2: Calculating  andinitializing *NNTi* and *DNNTi* based on CPU

//

//

//

//



Algorithm 3: Calculating  andinitializing *NNTi* and *DNNTi* based on GPU

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//



The CPU-based algorithm of calculating  andinitializing *NNTi* and *DNNTi* is shown in Algorithm 2. My first method parallelizes computing the distance between each data point and each centroid in Algorithm 2. One data point is dispatched to one thread, and then each thread calculates the distance from a corresponding data point to k centroids, and then initializes *NNTi* and *DNNTi*, as shown in Algorithm 3. Line 1 and 2 show how the algorithm designs the thread block and gird. Line 3 to 6 calculate the position of the corresponding data point, NNT, and DNNT for each thread in global memory. Line 7 loads the data point into the register. Lines 8-13 calculate the distance and initialize *NNTi* and *DNNTi*.

Algorithm 3 only has one level of loop instead of two levels in Algorithm 2, because the loop for N data points has been dispatched to N threads, which decreases the time consumption significantly because many threads are working in parallel. It is worth pointing out that the key step of achieving high efficiency is loading the data points into the on-chip registers, which ensures that reading the data point from global memory happens only once when calculating the distances between the data point and k centroids. Obviously, reading from register is much faster than reading from global memory. Besides, coalesced access to the global memory also decreases the reading latency.

B. Updating memberships **** and 

Apply the design as described in the section A.

Algorithm 4: Updating memberships **** and  based on CPU

//

//

//

//

//

//



Algorithm 5: Updating memberships **** and  based on GPU

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//



C. Computing the new center for each clusters

Algorithm 6: Computing the new center for each clusters based on CPU

//

//

//

//

//

//



D. Calculating, updating *NNTi* and *DNNTi* for *i* = 1 to *N*, and calculating distortion value *J*

Algorithm 7: Calculating, updating *NNTi* and *DNNTi*, and calculating distortion value *J* based on CPU

//

//

//

//

//



Algorithm 8: Calculating, updating *NNTi* and *DNNTi*, and precomputing for reduction distortion value *J* based on GPU

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//



E. Calculating distortion value *J* using the parallel reduction algorithm on GPU

//

//

//

//

//

//

//

//

1. ;
2. \*M




8. s
10. s
12. s
13. s
14. s
15. s
16. s
17. s

IV. EXPERIMENTAL RESULTS

Implement GPU-based parallel GFKM algorithm and make comparisons with CPU based implementations.

REFERENCES

[1] Franklin J. C. Lai, Eric Y. T. Juan, and Jim Z. C. Lai, Generalized Fuzzy k-Means Clustering Using m nearest Cluster Centers, 2013.

[2] You Li, Kaiyong Zhao, Xiaowen Chu, and Jiming Liu, Speeding up K-Means Algorithm by GPUs, 2010.

[3] http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86\_website/projects/reduction/doc/reduction.pdf