CPU-GPU Dynamic Approximation

Xi Chen
Mingze Gao
Yanzhou Liu

Outline

- Motivation for Approximation Computation
- CUDA Architecture and GPU Programming
- Approximation for Clustering Algorithm
- K-means Algorithm
- Expectation and Maximization (EM) Algorithm
- Performance Analysis
- Problems and Future Work

Motivation

- Modern General-Purpose GPUs (GPGPUs) can accelerate “appropriate” algorithms
- Approximation: trade off between computation quality and computational effort
- Error resilience exists in applications, no “golden” output value for these applications
- Approximation on iterative methods (IMs)
  - Possible to give a lightweight quality estimator

Approximation for Clustering Algorithm

- Clustering:
  - Divide up data into groups/clusters so that points in the same group are more “similar” to each other in some way than points outside.
- Applications:
  - Marketing, Classifications in Biology and Geology, Insurance Field, online documentation classification
- Quality Evaluation Metric: Hamming distance
- Static design:
  - ∑(distance all the cluster centers move) < 0.02% * the largest distance in the dataset

CUDA Architecture and GPU Programming in OpenCL

- OpenCL platform model on CUDA Architecture

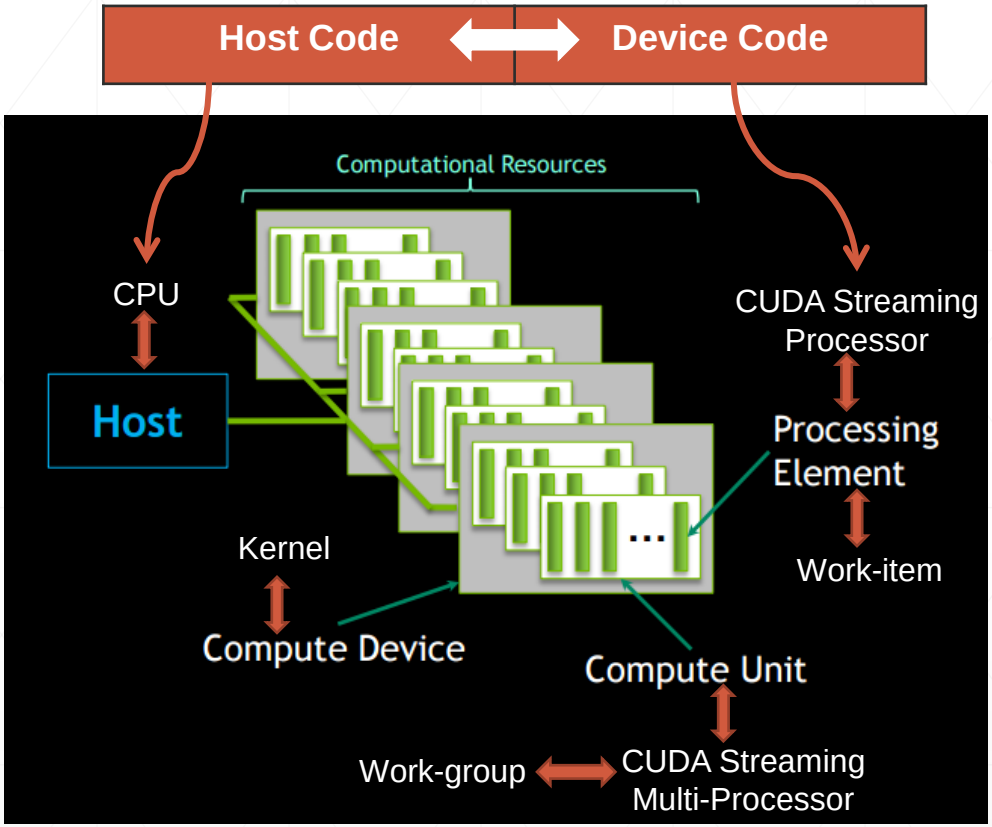


Fig1: OpenCL Platform model on CUDA Architecture

- Host code sends commands to the Devices:
  - Transfer data between host memory and device memory
  - Execute kernel code
- Commands are pushed to command queue, executing in order or out of order

CUDA Architecture and GPU Programming in OpenCL

- Memory Space
  - Private memory: Specific to a work-item; not visible to other work-items.
  - Local memory: Local to a work-group.
  - Constant memory: A memory region used for objects allocated and initialized by the host.
  - Global memory: Accessible to all work-items executing in a context.

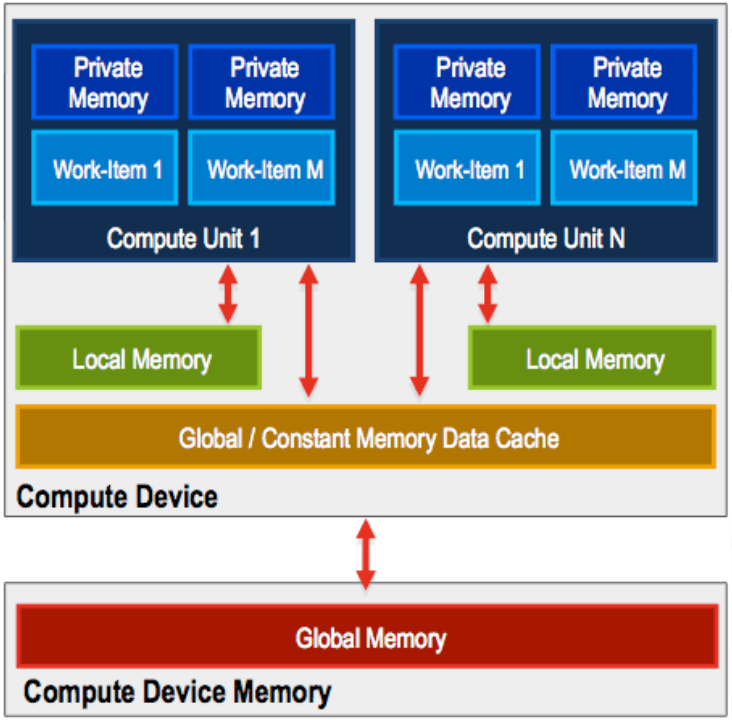
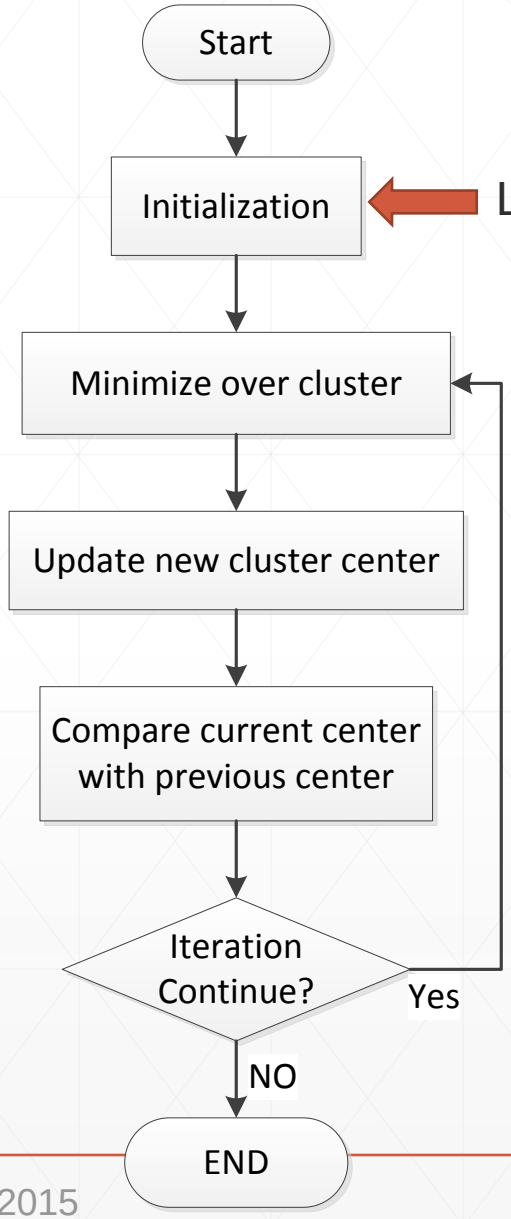


Fig2: OpenCL Memory Model

K-means Algorithm—Block Diagram



Load data and randomly select the cluster centers

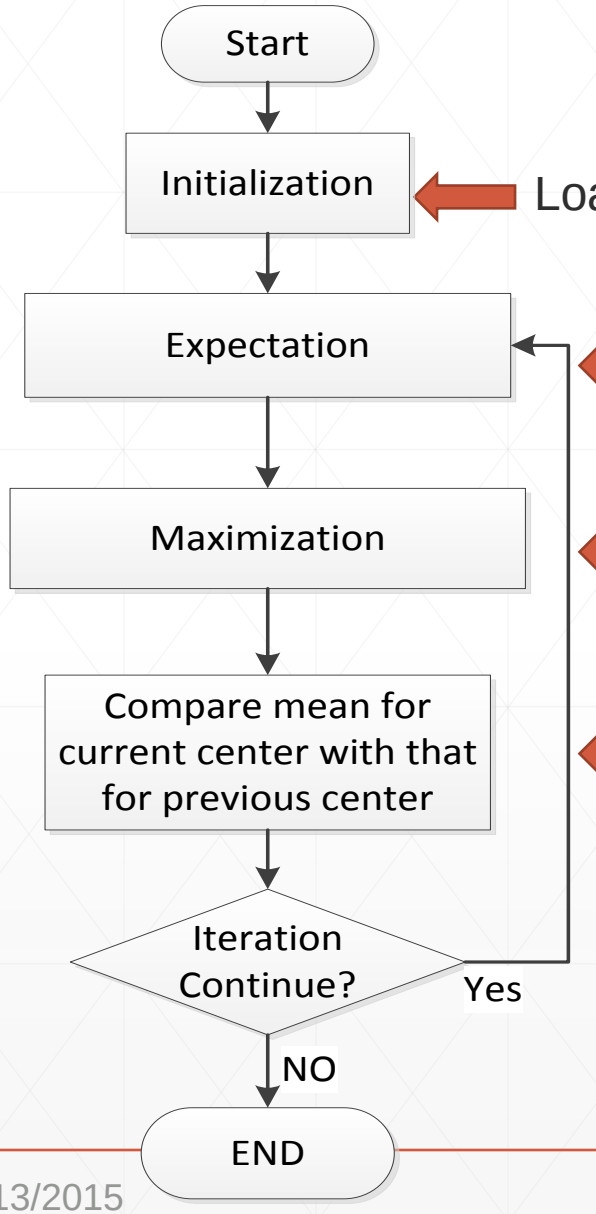
For every point, find the cluster center closest to itself

Update the new cluster centers based on classification in the previous step

Compare the previous center with the current center

If matching the criterion, stop iteration

EM Algorithm—Block Diagram



Load data and set covariance matrix for clusters

Calculate the probability that each data point belongs to each cluster

Re-calculate the cluster means and covariance based on the probabilities calculated in the expectation step

Compare means of the previous center with the current center

If matching the criterion, stop iteration

K-means Algorithm—Result and Performance

- Use different memory regions

Table1: GPU performance with different memory allocation

CPU execution time	GPU execution time (local memory)	GPU execution time (global memory)
9ms	3.186ms	3.486ms

  - Data set: 100 points, 9 dimensions, 4 clusters

- CPU implementation vs GPU implementation vs GPU approximation

# of points	# of clusters	# of dimensions	CPU time (ms)	GPU time (ms)	GPU time (ms) (Approximation)	Cut-off iterations
100	4	9	9	3.486	1.743	3/6
4000	4	2	389	174.276	87.138	6/12
40000	4	2	4562	2594.892	1297.446	6/12
40000	4	16	40081	20008.490	2589.334	74/85

- Device:
  - CPU: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
  - GPU: NVIDIA GeForce GTX 680

K-means Algorithm—Result and Performance

- Result comparison：

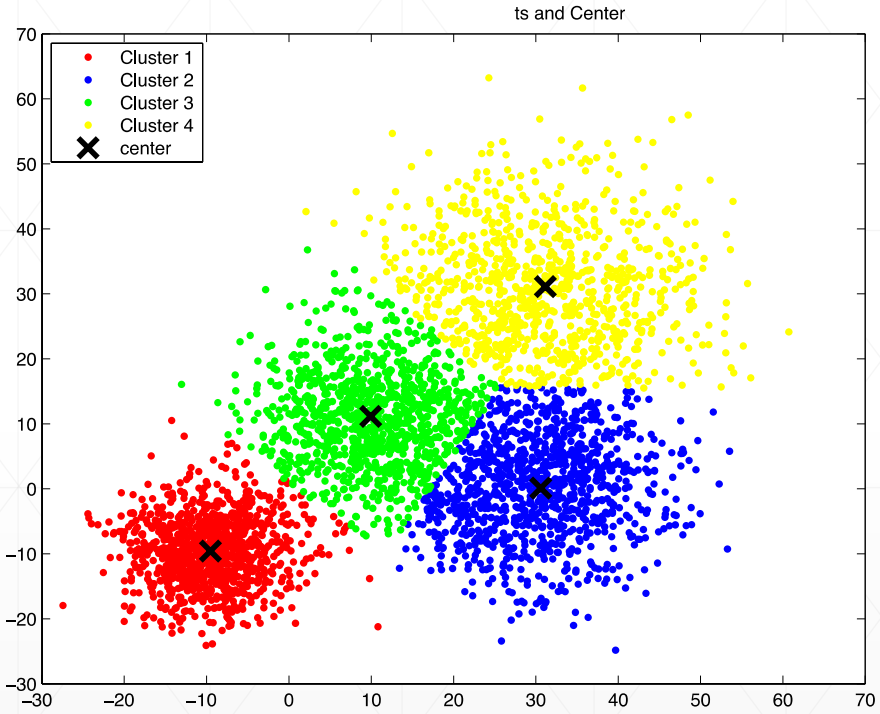


Fig3: Accurate clustering result

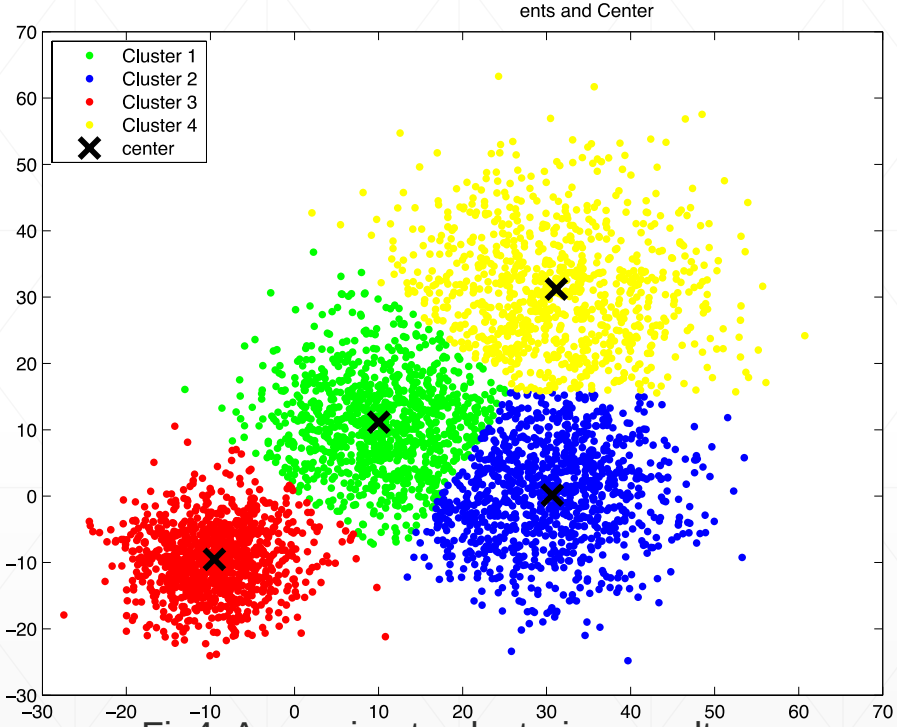


Fig4: Approximate clustering result

Data set: 4000 points; 4 clusters; 2 dimensions

EM Algorithm—Result and Performance

- GPU implementation vs GPU approximation

# of points	# of clusters	# of dimensions	GPU time (ms)	GPU time (ms) (Approximation)	Cut-off iterations
1000	3	2	3.486	1.743	96/169
4000	4	2	174.276	87.138	125/144
40000	2	2	2594.892	1297.446	157/265

- Device:
  - CPU: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
  - GPU: NVIDIA GeForce GTX 680

EM Algorithm—Result and Performance

- Result comparison：

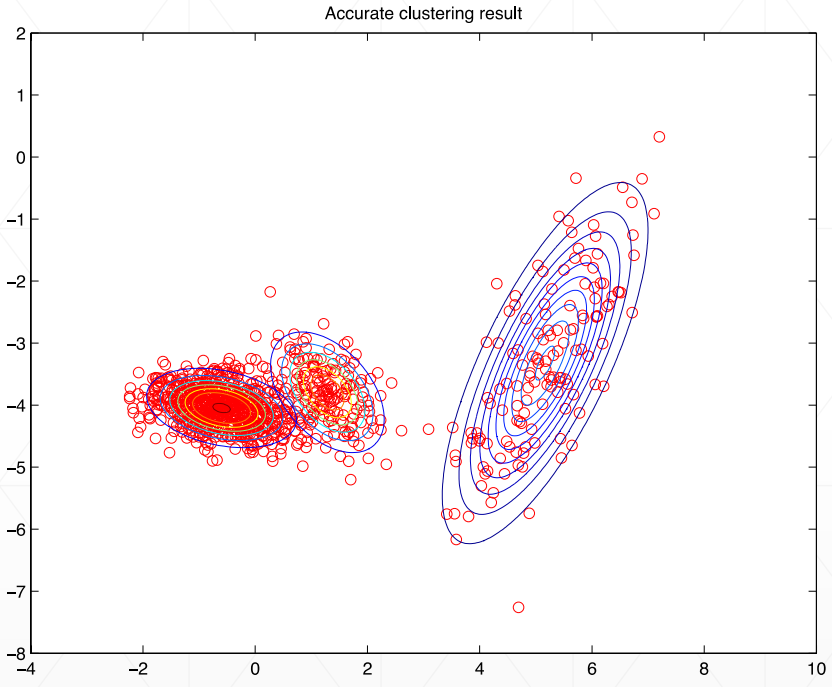


Fig5: Accurate clustering result

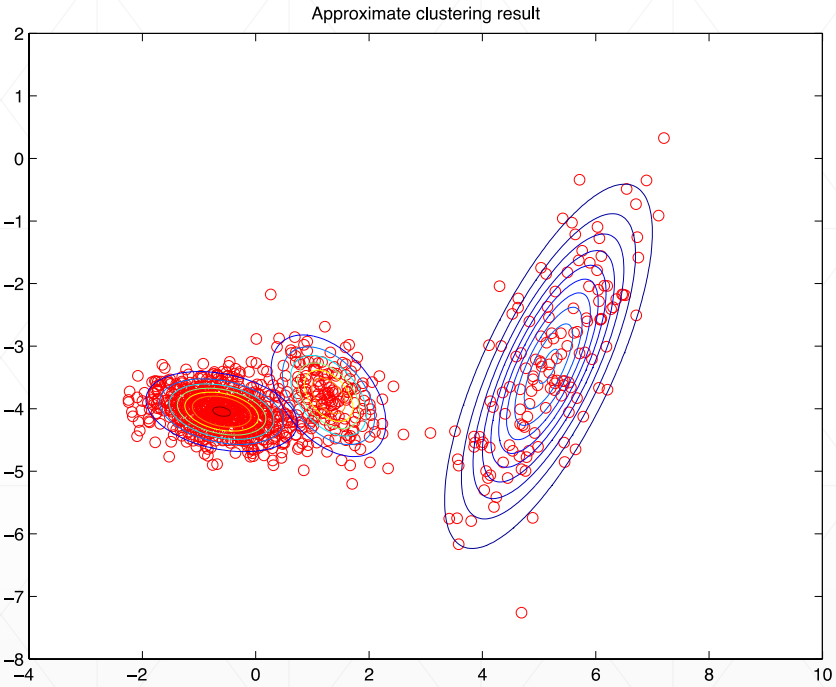


Fig6: Approximate clustering result

Data set: 1000 points; 3 clusters; 2 dimensions

Performance Analysis

- Local memory vs global memory
  - Less latency and better performance
  - Local memory resource is limited
- Speedup issues
  - K-means
    - Parallelized reduction
    - Extra matrices were added to parallelize center updating
    - Limited by matrix multiplication
      - Work group size is M*K if given M*N matrix and N*K matrix
        - In the kernel code, summation of N elements is sequentially processed (N is the number of points)
  - EM
    - Some part of the code can not be parallelized
      - E.g. matrix inversion, matrix determinant
    - Barriers for synchronization to prevent data race
    - Use double type
    - Similar issues (e.g. matrix multiplication) to K-means

Problems

- Problem: Parallelism for Matrix Multiplication

$$\begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{MN} & \cdots & a_{MN} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & a_{1K} \\ \vdots & \ddots & \vdots \\ b_{N1} & \cdots & b_{NK} \end{bmatrix}$$

  - Flatten the multiplication process: (1). Compute all a_{ij}*b_{jk} and get M*N*K results (2). Summation for every N results in (1) in parallel
  - When doing summation, running the same kernel and same data multiple times gives different results even adding barrier function for synchronization

- Data race conditions during kernel execution
  - `for` loop in kernel code easily causes data race problems
    - To avoid such condition, either do it serially or call kernel multiple times
    - Barriers needed

Problems

- **`clblas`** library usage
  - Basic Linear Algebra Subprograms supported by AMD
  - Bad portability, difficult to use on NVIDIA GPU
  - Poor performance (even slower than our implementation)
- Matrix operations
  - Matrix Determinant
    - As size of matrix increases, recursive methods needed for computation which is not easy for OpenCL implementation
  - Matrix Inversion
    - As size of matrix increases, it is hard to get solution of the inversion
- Use GPU in Deepthought2
  - Compile OpenCL Host device code successfully
  - When sbatch jobs with specifying GPU request, it returns segmentation fault.

Future Work

- Optimize the criteria for approximation to get more reliable results, e.g. use the ratio of nodes changing their clusters in each iteration or even dynamic approximation

- Further optimize the matrix multiplication
  - Data race problems need to be solved first

Reference

- Khronos OpenCL Working Group. "The OpenCL Specification, version 1.0. 29, 8 December 2008." *U RL http://khronos. org/registry/cl/specs/opencl-1.0* 29.
- CUDATM, N. "OpenCL Programming Guide for the CUDA Architecture."*NVIDIA Corporation* (2009).
- Zhang, Qian, et al. "ApproxIt: An approximate computing framework for iterative methods." *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*. IEEE, 2014.
- http://www.cc.gatech.edu/~vetter/keeneland/tutorial-2011-04-14/06-intro_to_opencl.pdf
- http://sa09.idav.ucdavis.edu/docs/SA09_NVIDIA_IHV_talk.pdf
- https://chrisjmccormick.wordpress.com/2014/08/04/gaussian-mixture-models-tutorial-and-matlab-code/

Q&A

Thanks for Your Attention!