Efficient k-Means on GPUs

Clemens Lutz
DFKI GmbH
clemens.lutz@dfki.de

Sebastian Breß
DFKI GmbH
sebastian.bress@dfki.de

Tilmann Rabl
TU Berlin
tilmann.rabl@tu-berlin.de

Steffen Zeuch
DFKI GmbH
steffen.zeuch@dfki.de

Volker Markl
TU Berlin
volker.markl@tu-berlin.de

ABSTRACT

k-Means is a versatile clustering algorithm widely-used in practice. To cluster large data sets, state-of-the-art implementations use GPUs to shorten the data to knowledge time. These implementations commonly assign points on a GPU and update centroids on a CPU.

We show that this approach has two main drawbacks. First, it separates the two algorithm phases over different processors, which requires an expensive data exchange between devices. Second, even when both phases are computed on the GPU, the same data are read twice per iteration, leading to inefficient use of memory bandwidth.

In this paper, we describe a new approach that executes *k*-means in a single data pass per iteration. We propose a new algorithm to updates centroids that allows us to perform both phases efficiently on GPUs. Thereby, we remove data transfers within each iteration. We fuse both phases to eliminate artificial synchronization barriers, and thus compute *k*-means in a single data pass. Overall, we achieve up to 20× higher throughput compared to the state-of-the-art approach.

ACM Reference Format:
Clemens Lutz, Sebastian Breß, Tilmann Rabl, Steffen Zeuch, and Volker Markl. 2018. Efficient k-Means on GPUs. In *DaMoN'18: 14th International Workshop on Data Management on New Hardware, June 11, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3211922.3211925

1 INTRODUCTION

To find patterns in large data sets, *k*-means [21, 22] is an essential tool in the data scientist's toolkit. In particular, practitioners of sciences involving "big data", such as genome analysis [16, 31, 33] and climatology [8, 10, 19] require fast *k*-means implementations for a short data to knowledge time. Furthermore, many algorithms build on top of *k*-means to cover new use-cases, e.g., BIRCH [34] and streaming *k*-means [30]. Thus, speeding up *k*-means enables data scientists to create new insights by exploiting larger data sets in higher quality. Although relational databases support *k*-means via SQL [17, 24], high-performance *k*-means requires specialized database features [25]. The ubiquitous availability of GPUs provided by cloud computing platforms [1, 2] promises inexpensive and fast execution of machine learning algorithms. However, to exploit the full performance of GPUs, algorithms require careful design and tuning, as shown by previous research on accelerating relational data management with GPUs [5, 6, 12, 14, 15, 18, 26, 27].

ACM ISBN 978-1-4503-5853-8/18/06...$15.00
https://doi.org/10.1145/3211922.3211925

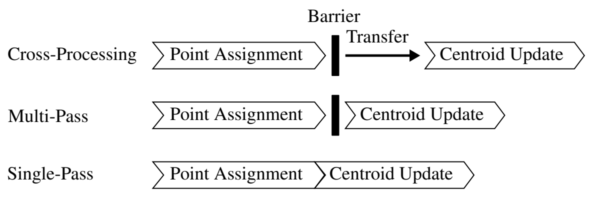

Figure 1: *k*-Means Execution Strategies.

Harnessing the computing power of GPUs requires an efficient execution strategy. In Figure 1, we show individual steps to achieve this goal. *k*-Means refines results by iterating over two phases: *Point Assignment* and *Centroid Update*. Research over the last decade focused mostly on accelerating both *k*-means phases on CPU or GPU separately [7, 11, 13, 29, 32]. In particular, they assign points on GPU and update centroids on CPU, thus *cross-processing* over multiple processors. This *Cross-Processing* strategy has the inherent overhead of exchanging data between GPU and CPU over the PCI-e bus in every iteration. Some approaches perform Point Assignment and Centroid Update on GPUs [4, 20]. However, both types of approaches introduce artificial synchronization barriers between the two phases. The barriers require the algorithm to make two passes over points data per iteration, and cause the *multi-pass problem*.

Our contributions are as follows:

(1) We outline an algorithm for a highly-efficient GPU-optimized Centroid Update to solve the cross-processing problem.
(2) We introduce the *Single-Pass* execution strategy on GPUs to solve the multi-pass problem.
(3) We show a preliminary comparison of our Single-Pass execution strategy to previous approaches.

2 APPROACH

The main challenges to compute *k*-means in a single pass are two-fold: 1. Due to the fundamentally different hardware architectures of CPUs and GPUs, the Centroid Update designed for CPUs is cache-inefficient on GPUs. 2. The two phases of *k*-means, Point Assignment and Centroid Update, access data in opposite directions (row- vs. column-wise access). Thus, fusing these phases to a single data pass is non-trivial [12]. We describe the key insights of our approach that address these challenges.

Efficient Centroid Update on GPU. The large, per-core L2 cache of CPUs allows cache-efficient algorithms to store working sets of up to several megabytes in size. In contrast, on GPUs, tens to hundreds of threads store their private working sets in several tens of kilobytes *shared memory* caches. The aggregate size of the working sets can easily exceed the available cache space, which leads to cache thrashing. Due to these architectural differences, Centroid Update strategies optimized for CPUs underperform on GPUs. Nevertheless, in principle, such a CPU-optimized Centroid Update is capable of running on GPUs [20]

Our first key insight is that, in Centroid Update, we are able to process each data feature independently of the others. We use this insight to reduce the cache space required by each thread to store its working set. In a CPU-optimized design, we partition data among all threads (i.e., cores) and process them in parallel. Finally, we aggregate the results of all partitions into a new set of centroids. In contrast, in our GPU-optimized Centroid Update, we partition data *and decompose each data point into its discrete features*. Thus, we partition data in two dimensions — points and features — instead of one dimension. Whereas partitioning data points has no influence on the working set, partitioning data by its features also partitions the working set. In effect, the working set becomes smaller, which prevents cache thrashing when running hundreds of GPU threads.

Single-Pass GPU *k*-Means. Fusing our Centroid Update algorithm for GPUs with Point Assignment involves synchronizing multiple threads. In contrast, fusing their CPU counterparts involves a single thread [23], and thus is straightforward. In Point Assignment, we assign each point to a cluster. In this step, we compute point-to-centroid distances, for which we access all the point's and centroid's features. Thus, we partition in only one dimension (i.e., points), because we are unable to partition in two dimensions. The intermediate results are passed on to Centroid Update, in which we access columns of data and partition data two-way. This mismatch in how we access and partition data on GPUs incurs a global synchronization. In contrast, in the CPU-optimized strategy, both algorithm phases partition data in the same way, thus directly fuse together.

Our second key insight is that thread warps make parallel, on-the-fly data transpose fast. A warp consists of multiple threads that execute processor instructions in lock-step, and is a fundamental trait of GPU architecture. Due to executing in lock-step, synchronizing GPU threads with a thread barrier is much cheaper than on a CPU. Thus, each thread writes its partition of data and intermediate results into shared memory. After all threads have executed the thread barrier, each thread reads its now-transposed partition. Using this technique, we are able to fuse Point Assignment and our GPU-optimized Centroid Update into the *Single-Pass k*-means strategy.

3 RESULTS & DISCUSSION

In Figure 2, we compare our Single-Pass execution strategy to the Multi-Pass and Cross-Processing execution strategies. As points of reference, we also include two open-source baselines, Armadillo [28] on CPU and Rodinia [9] on GPU. We execute all strategies on an Intel Core i7-6700K ("Skylake") CPU with 32 GB memory and an Nvidia GeForce GTX 1080 ("Pascal") GPU with 8 GB memory on a 2 GB generated data set [3] with 4 features. We set $k = 4$ to show a data-intensive scenario unbounded by computation.

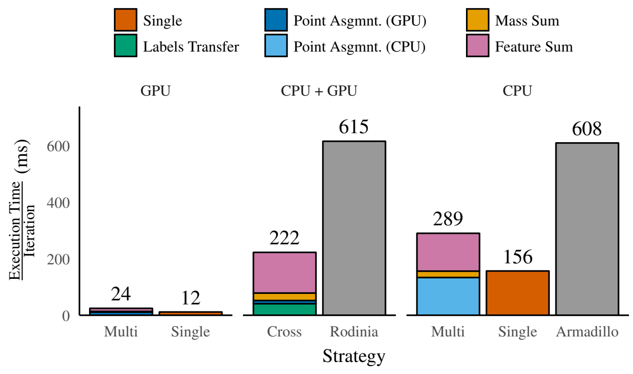

Figure 2: Execution time breakdowns for different *k*-Means strategies on CPU and GPU for 4 features and 4 clusters.

On GPU, we observe that our Single-Pass strategy runs 2× faster than the Multi-Pass strategy and 18.5× faster than the Cross-Processing strategy. As Single-Pass reads half as much data as Multi-Pass, its execution time is halved. Further, the Multi-Pass strategy is 9.3× faster than the Cross-Processing strategy. We deduce that updating centroids on the CPU and transferring intermediate results on every iteration slows down the Cross-Processing strategy compared to the Single-Pass and Multi-Pass strategies.

In contrast, on CPU, our Single-Pass strategy has 1.8× and 1.4× higher throughput than the Multi-Pass and Cross-Processing strategies, respectively. The Cross-Processing strategy outperforms the Multi-Pass strategy by 1.3×.

We also observe a factor 2.7× difference between our implementation of the Cross-Processing strategy and Rodinia's implementation of the same strategy. Likewise, there is a factor 2.1× difference between our Multi-Pass strategy and Armadillo's implementation. We attribute these differences to the hand-tuning of our implementations for these specific processors.

We summarize that our Single-Pass strategy for GPUs shows promising results that merit further investigation.

4 CONCLUSION & OUTLOOK

In this paper, we introduce a GPU-optimized strategy for *k*-means. We highlight two fundamental problems of previous approaches: *cross-processing* and *multi-pass* execution. Our proposed solutions center around an efficient algorithm for updating centroids on GPUs. In our presented results, we show that our Single-Pass strategy for GPUs achieves up to 2× and 18.5× higher throughput than the Multi-Pass and Cross-Processing strategies, respectively. We will extend our work to present our solutions in more detail. In particular, we will show both data- and compute-intensive parameters (i.e., small and large *k*). Furthermore, we will explore data sets with different numbers of features. Finally, we will demonstrate that our Single-Pass strategy is feasible for data sets larger than GPU memory.

Acknowledgments
This work was funded by the EU projects SAGE (671500) and E2Data (780245), DFG Priority Program "Scalable Data Management for Future Hardware" (MA4662-5), and the German Ministry for Education and Research as BBDC (01IS14013A).

REFERENCES

[1] 2018. Amazon EC2 Pricing. (May 8 2018). https://aws.amazon.com/ec2/pricing/on-demand

[2] 2018. Microsoft Azure Pricing. (May 8 2018). https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/

[3] David Arthur and Sergei Vassilvitskii. 2007. k-Means++: The advantages of careful seeding. In *ACM-SIAM*. 1027–1035.

[4] Hong-tao Bai, Li-li He, Dan-tong Ouyang, Zhan-shan Li, and He Li. 2009. k-Means on commodity GPUs with CUDA. In *WRI CSIE*. 651–655.

[5] Sebastian Breß et al. 2017. Generating custom code for efficient query execution on heterogeneous processors. *CoRR* abs/1709.00700 (2017).

[6] Sebastian Breß, Henning Funke, and Jens Teubner. 2016. Robust query processing in co-processor-accelerated databases. In *SIGMOD*. 1891–1906.

[7] Feng Cao, Anthony K. H. Tung, and Aoying Zhou. 2006. Scalable clustering using graphics processors. In *WAIM*. 372–384.

[8] Christophe Cassou. 2008. Intraseasonal interaction between the Madden–Julian Oscillation and the North Atlantic Oscillation. *Nature* 455, 7212 (Sept. 2008), 523–527.

[9] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC*. 44–54.

[10] M Dall, DCS Beddows, Peter Tunved, Radovan Krejci, Johan Ström, H-C Hansson, YJ Yoon, Ki-Tae Park, S Becagli, R Udisti, et al. 2017. Arctic sea ice melt leads to atmospheric new particle formation. *Scientific reports* 7, 1 (2017), 3318.

[11] Reza Farivar, Daniel Rebolledo, Ellick Chan, and Roy H. Campbell. 2008. A parallel implementation of k-means clustering on GPUs. In *PDPTA*. 340–345.

[12] Henning Funke, Sebastian Breß, Stefan Noll, Volker Markl, and Jens Teubner. 2018. Pipelined query processing in coprocessor environments. In *SIGMOD*. ACM.

[13] Jesse Hall and John Hart. 2004. GPU acceleration of iterative clustering. In *GPGPU*. 45–52.

[14] Bingsheng He, Mian Lu, Ke Yang, Rui Fang, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. 2009. Relational query coprocessing on graphics processors. *TODS* 34, 4 (2009).

[15] Max Heimel, Michael Saecker, Holger Pirk, Stefan Manegold, and Volker Markl. 2013. Hardware-oblivious parallelism for in-memory column-stores. *PVLDB* 6, 9 (2013), 709–720.

[16] Nathaniel D Heintzman, Rhona K Stuart, Gary Hon, Yutao Fu, Christina W Ching, R David Hawkins, Leah O Barrera, Sara Van Calcar, Chunxu Qu, Keith A Ching, et al. 2007. Distinct and predictive chromatin signatures of transcriptional promoters and enhancers in the human genome. *Nature Genetics* 39, 3 (2007), 311.

[17] Joseph Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. 2012. The MADlib analytics library or MAD skills, the SQL. *PVLDB* 5, 12 (2012), 1700–1711.

[18] Tomas Karnagel, René Müller, and Guy M. Lohman. 2015. Optimizing GPU-accelerated group-by and aggregation. In *ADMS*. 13–24.

[19] Kristin M. Kleisner, Michael J. Fogarty, Sally McGee, Analie Barnett, Paula Fratantoni, Jennifer Greene, Jonathan A. Hare, Sean M. Lucey, Christopher McGuire, Jay Odell, Vincent S. Saba, Laurel Smith, Katherine J. Weaver, and Malin L. Pinsky. 2016. The effects of sub-regional climate velocity on the distribution and spatial extent of marine species assemblages. *PLOS ONE* 11 (02 2016), 1–21.

[20] You Li, Kaiyong Zhao, Xiaowen Chu, and Jiming Liu. 2010. Speeding up k-means algorithm by GPUs. In *IEEE CIT*. 115–122.

[21] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* 28, 2 (1982), 129–136.

[22] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob.*, Vol. 1. 281–297.

[23] Disa Mhembere, Da Zheng, Carey E. Priebe, Joshua T, Vogelstein, and Randal Burns. 2017. knor: A NUMA-optimized in-memory, distributed and semi-external-memory k-means library. In *HPDC*.

[24] Carlos Ordonez. 2004. Programming the k-means clustering algorithm in SQL. In *SIGKDD*. 823–828.

[25] Linnea Passing, Manuel Then, Nina Hubig, Harald Lang, Michael Schreier, Stephan Günnemann, Alfons Kemper, and Thomas Neumann. 2017. SQL- and operator-centric data analytics in relational main-memory databases. In *EDBT*. 84–95.

[26] Holger Pirk, Stefan Manegold, and Martin L. Kersten. 2014. Waste not…Efficient co-processing of relational data. In *ICDE*. 508–519.

[27] Holger Pirk, Oscar Moll, Matei Zaharia, and Sam Madden. 2016. Voodoo - A vector algebra for portable database performance on modern hardware. *PVLDB* 9, 14 (2016), 1707–1718.

[28] Conrad Sanderson and Ryan Curtin. 2016. Armadillo: a template-based C++ library for linear algebra. *Journal of Open Source Software* (2016).

[29] Arul Shalom, Manoranjan Dash, and Minh Tue. 2008. Efficient k-means clustering using accelerated graphics processors. In *DaWaK*. 166–175.

[30] Michael Shindler, Alex Wong, and Adam W. Meyerson. 2011. Fast and accurate k-means for large datasets. In *NIPS*. 2375–2383.

[31] Sarah A Vitak, Kristof A Torkenczy, Jimi L Rosenkrantz, Andrew J Fields, Lena Christiansen, Melissa H Wong, Lucia Carbone, Frank J Steemers, and Andrew Adey. 2017. Sequencing thousands of single-cell genomes with combinatorial indexing. *Nature Methods* 14, 3 (2017), 302.

[32] Fuhui Wu, Qingbo Wu, Yusong Tan, Lifeng Wei, Lisong Shao, and Long Gao. 2013. A vectorized k-means algorithm for Intel Many Integrated Core architecture. In *APPT*. 277–294.

[33] Chongzhi Zang, Tao Wang, Ke Deng, Bo Li, Sheng'en Hu, Qian Qin, Tengfei Xiao, Shihua Zhang, Clifford A. Meyer, Housheng Hansen He, Myles Brown, Jun S. Liu, Yang Xie, and X. Shirley Liu. 2016. High-dimensional genomic data bias correction and data integration using MANCIE. *Nature Communications* 7 (April 2016), 11305.

[34] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: An efficient data clustering method for very large databases. In *SIGMOD*. 103–114.