AVID: GPU-enabled Visual Analytics with GPU-FAST-PROCLUS

Jakob Rødsgaard Jørgensen
jakobrj@cs.au.dk
Department of Computer Science
Aarhus University
Denmark

Ira Assent
ira@cs.au.dk
Department of Computer Science
DIGIT Centre for Digitalisation, Big Data and Data Analytics
Aarhus University
Denmark

Hans-Jörg Schulz
hjschulz@cs.au.dk
Department of Computer Science
Aarhus University
Denmark

ABSTRACT

GPU-FAST-PROCLUS is a GPU-parallelized algorithm for projected clustering based on the k-medoids approach. It speeds up clustering to allow for real-time interaction – even for datasets of millions of items. Interactivity allows users to quickly determine sensible clustering parameters such as the number of clusters k, provided a suitable visualization is available. Yet, as clustering and visualization are usually decoupled, cluster results are funneled from the GPU back to the CPU, only to be mapped onto appropriate graphics, which are then rendered on the GPU again. This introduces a bottleneck that hinders fluid interaction with clustering.

As a solution to this, we propose AVID (Analysis and Visualization In Device). Following the principle "What happens on the GPU, stays on the GPU", AVID removes the round trip to the CPU and keeps clustering results on the GPU to render them on the GPU directly. By doing so, users can interactively tune projected clustering parameters and observe the effects without noticeable delay. In our demo system, we showcase the efficiency of our data management strategies for projected clustering as well as the efficacy of data visualization.

1 INTRODUCTION

Projected clustering aims to identify groups of similar objects in subspace projections of the full-dimensional space. Efficient algorithms for projected clustering are crucial as the number of possible subspace projections is exponential in the number of dimensions. Projected clustering algorithms must be provided with predefined parameters, but the best parameters are rarely known in advance. The choice of sensible parameters generally requires a human in the loop [4].

To enable interactive, human-in-the-loop parametrization of clustering, the effects of a change in parameters must be observable at interactive framerates. This usually means that results must be computed in around $100ms$ to reduce the *temporal separation* [13, p.140] between parameter change and visualization change,and thus providing the necessary "fluidity" [3]. In Jørgensen et al. [6], we present GPU-FAST-PROCLUS, a GPU-parallelized algorithm that computes projected clusters under the definition of the well-known PROCLUS approach [2], which extends k-medoids clustering to subspace projections. GPU-FAST-PROCLUS runs on a million points in around $100ms$, and therefore theoretically allows for real-time interaction [11]. Yet, in order to visualize the results of GPU-FAST-PROCLUS to allow their interactive exploration under different parameterizations and in different projections – similar to the works by Tatu et al. [12] or Yuan et al. [15] – we would need to visualize these millions of points. To do so, the data would be clustered on the GPU (Graphics Processing Unit), then be transferred back to the CPU and mapped onto graphics primitives using some graphics framework, only to be then rendered again on the GPU.

To prevent the bottleneck of the CPU, we propose to compute both the cluster analysis and the visualization as a combined pipeline directly on the GPU. While GPU-based visualization is widely used [5, 10, 14], GPU-based Visual Analytics combining computational analysis and visualization on the GPU is still very rare with only a handful of systems having been published – e.g., [1, 7, 9]. To the best of our knowledge, no such purely GPU-based solution exists for computing and visualizing projected clusterings. Hence, we propose and demonstrate AVID (Analysis and Visualisation In Device), a real-time interactive data visualization for GPU-FAST-PROCLUS.

2 PROCLUS AND GPU-FAST-PROCLUS

PROCLUS [2] is an axis-parallel projected clustering algorithm, inspired by the k-medoids algorithm CLARANS [8]. Given a dataset and the parameters

- number of clusters k,
- average number of dimensions l, and
- scalars A and B.

PROCLUS returns a cluster assignment for each point in some axis-aligned subspace projection for the respective cluster. To that end, PROCLUS proceeds in three phases:

(1) Greedily picking potential medoids $M \subset Data$.
(2) Iteratively improving the best set of current medoids $m \subset M$ that yields the best projected clustering
(3) Further refining the best clustering.

The final result are k projected clusters within on average l-dimensional subspace. E.g., if we have $k = 3$ and $l = 4$, clusters could exist within subspaces of 2, 3, or 7 dimensions.

Our GPU-FAST-PROCLUS approach [6] provides efficient GPU-parallelization of PROCLUS clustering and even supports reusing computations between parameter settings, which is important in practice when determining the best set of parameters for a dataset and analysis task at hand. In Jørgensen et al. [6], we also provide an experimental evaluation on both real-world and synthetic datasets, and with varying size, dimensionality, distribution, and parameter settings. In the following, we provide a brief overview, with more details given in [6].

Speed-up is achieved by maintaining the distances $Dist$ from all points to all previously used medoids. Furthermore, the computation of scores $Z_{i,j}$, which indicate the suitability of medoid m_i in dimension j, is reorganized. The most expensive part of computing $Z_{i,j}$ is the sum of distances $H_{i,j}$ from each medoid m_i to all points that are within that medoid's sphere of influence L_i along each dimension j. The sphere of influence L_i is all points within a δ_i radius of m_i, where δ_i is the distance from m_i to the closest of the current medoids m_j. Since L_i is not likely to change much between iterations, the previously computed $H_{i,j}$ are stored, and updated only with the change ΔL_i in sphere L_i. To map between potential medoids M and the current medoids m, we use the index $MIdx$. All proposed reuses of computations can also be reused across parameter settings, provided that the selected sample of potential medoids M remains fixed.

In total, GPU-FAST-PROCLUS achieves up to 5000× speed-up over PROCLUS and can perform data analysis on a million points in around 100ms. Therefore, GPU-FAST-PROCLUS is admissible for real-time interaction [11]. Existing visualization frameworks, however, require data transfer to and from main memory and CPU involvement, dramatically increasing runtime beyond what users will accept in interactive settings.

3 ANALYSIS AND VISUALIZATION IN DEVICE

To leverage the speed of GPU-FAST-PROCLUS, we implement a "What happens on the GPU, stays on the GPU" data visualization supporting efficient visualization and interactive exploration of GPU-FAST-PROCLUS, called AVID (Analysis and Visualization In Device). We first discuss the implementation and then the data visualization.

3.1 Implementation

To the best of our knowledge, there does not exist a GPU-based data visualization framework that allows visualizing data directly located on the GPU. This implies that visualizing the result of GPU-FAST-PROCLUS must be funneled through the CPU and back to the GPU to be displayed. We, therefore, see the need to implement a visualization that does not use the standard data visualization frameworks to bypass the CPU. AVID is implemented using OpenGL[1], GLEW[2], GLUT[3], and CUDA[4]. OpenGL, GLEW, and GLUT are application programming interfaces (APIs) for graphics rendering that can leverage the GPU to achieve acceleration. Furthermore, OpenGL and CUDA support resource interpolation between the interfaces which implies that the graphics displayed by OpenGL can be manipulated using CUDA kernels or the CUDA API directly, which is used to implement GPU-FAST-PROCLUS.

Like any graphics framework, OpenGL, GLEW, and GLUT support drawing dots, lines, polygons, and text. However, they do not directly support our needs for interactive data visualization. To enable brushing and selection of elements in the visualization, we have implemented layout components as a tree-structure and mouse move and click listeners for each component. The content of each plot, e.g., points, bars, and grid cells, is the part of the visualization that is related to the data and results, and therefore the part that must be computed on the GPU. For each content, the visual mapping to color, scale, and location is done in parallel for all data points using CUDA kernels and rendered directly on the GPU. This implies that the data and results never leave the GPU. For demonstration purposes, we also implement a second version where visualization is done on the CPU, but the data analysis and rendering are still performed on the GPU. The user can switch between implementations to experience the difference in runtime.

[1]https://www.opengl.org/
[2]http://glew.sourceforge.net/
[3]http://freeglut.sourceforge.net/
[4]https://developer.nvidia.com/cuda-downloads

To reuse computations between parameter settings, we keep $Dist$, $DistFound$, H, M, $MIdx$, $\delta^{t'}$, each $|L_i^{t'}|$, and the previous results in GPU-memory during the lifetime of the system.

3.2 Effective and efficient visualization

Using these as a foundation, we can construct an interactive data visualization for GPU-FAST-PROCLUS. A common workflow, when exploring parameters for projected clustering, is for the user to select an initial set of parameters, wait for the result, plot the result in a scatter plot matrix, maybe zoom in to see a single scatter plot, adjust the parameters and repeat the whole process. Optimally, the user could see all possible results at once. However, having $k \times l$ d-dimensional scatter plot matrices, either requires a huge display or makes the scatter plots so small that they become indistinguishable. Instead of this tedious workflow, we propose a data visualization that has an overview, a detailed view, and previews of different parameter settings side by side. This allows the user to apply filters, selection in, and changing all views at once.

Layout: The layout consists of several components and can be seen in Figure 1. In the top left, we have an overview in form of a scatter plot matrix. The scatter plots in the lower triangle of the matrix provide the same information as the upper triangle and are therefore replaced with heat maps to allow the user to more clearly identify dense areas. In the scatter plots, the points in each cluster are assigned a unique color corresponding to that cluster, but only of the dimensions that the scatter plot represent are both in the subspace that the cluster represent. We further assign a gray color to all points, which are not part of a cluster within the subspace shown by a scatter plot.

Next, we have two menus for selecting parameters l and k, where l is the average number of dimensions in the subspace of the k clusters. These are located below and to the right of the overview, and we refer to them as l-menu and k-menu, respectively. Both menus consist of three scatter plots, each showing a change in the associated parameter and two buttons with an arrow. This allows the user to quickly see part of the result for the surrounding parameter settings, before changing the parameters for the whole view.

At the bottom right corner of the overview, we have a column chart displaying the size of each cluster and the outliers in the current result.

To the right, we have a detailed view that shows a larger version of a scatter plot selected in the scatter plot matrix. Below the detailed view, there is a set of tools: a button for recomputing the results, a switch to change where the visualization is being computed, and a display of the time it takes to compute each frame.

Selecting: In the scatter plot matrix, the user can select a scatter plot to show in the detailed view and the menus. The selected scatter plot is highlighted with light blue in the scatter plot matrix as well as the dimension id. In the menus, the user can click the buttons to increment or decrement the parameters previewed in the three scatter plots. When a scatter plot in the menu is clicked, the associated parameter is selected and all views are updated. Again, the selected scatter plot is marked with blue.

Filtering and linking: The user can filter points within a specific area by brushing the data points shown in the detailed view, or to a specific cluster by clicking on the bars in the column chart. Both filters can be applied at the same time and both are linked to all other scatter plots and heatmaps.

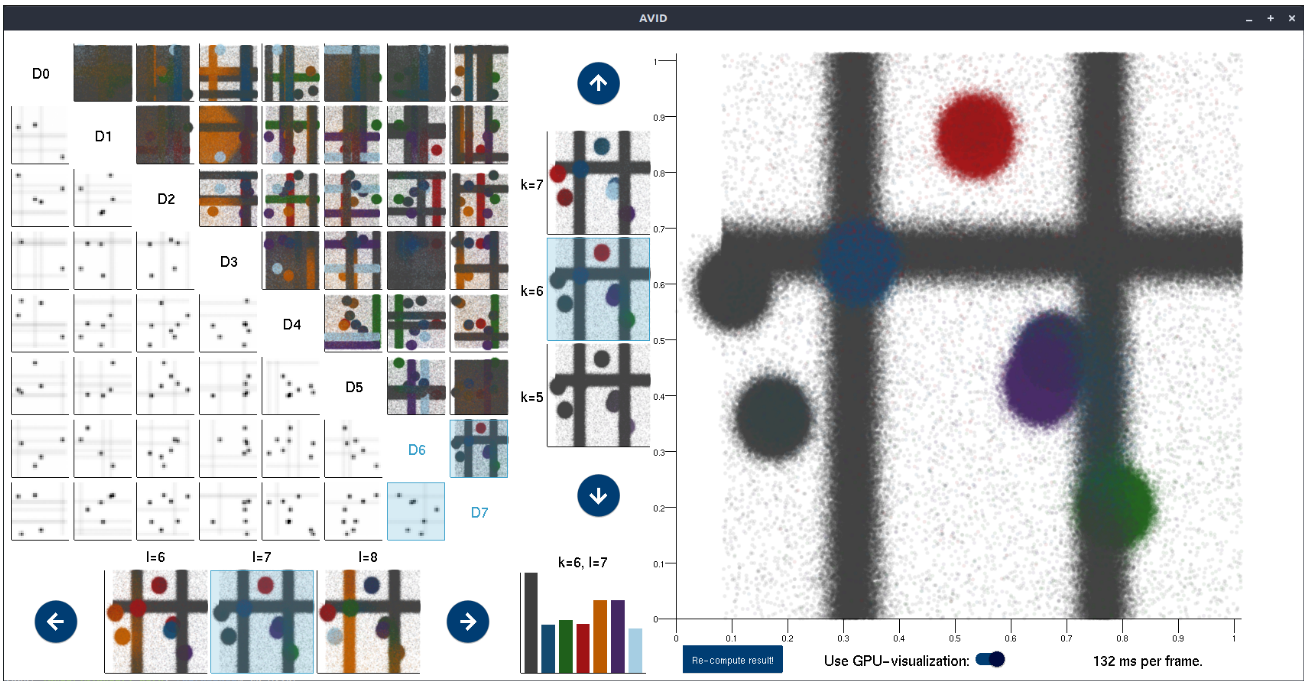

Figure 1: Layout of the data visualization AVID.

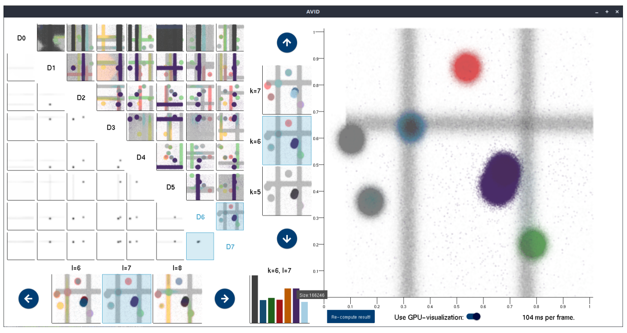

Figure 2: AVID showing a cluster that should be split in two.

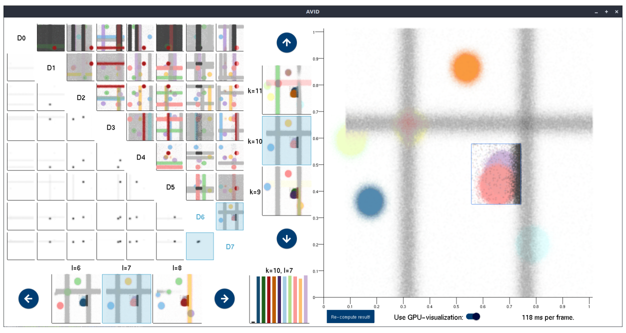

Figure 3: AVID showing brushing of two clusters.

4 DEMONSTRATION PLAN

For the demonstration of AVID, we use a synthetic dataset to show the different mechanisms. However, the user can provide AVID with any dataset $Data \in \mathbb{R}^{n \times d}$ as a CSV-file. As examples, we provide real-world datasets with the source code. We here describe the demonstration, but a short video and source code is provided at https://au-dis.github.io/publications/AVID/.

We present the architecture and the frameworks used in AVID, as described in Section 3.1. Furthermore, we present the layout and options for selecting and filtering, as outlined in Section 3.2. To illustrate that AVID is fast enough for real-time interaction, we demonstrate how to use these mechanics to find a good set of parameters. An example could be the following (due to random initialization in PROCLUS, there can be variations):

Initially, we start with $k = 6$ and $l = 7$ as seen in Figure 1. We pick a scatter plot to show in the detailed view. We identify a cluster that maybe should have been split into two. Using the column chart, we filter to show only that cluster. The filter is applied in all plots and a look at the overview confirms that the cluster should be split in two, see Figure 2. In the k-menu, we investigate the previews. When changing the previews, a projected clustering is performed but happens so fast that it is not noticeable. We find at $k = 10$ the cluster is split, pick this parameter value and all views are updated accordingly. To update the previews for $l = 6$ and $l = 8$, two clusterings are computed using GPU-FAST-PROCLUS, and updated seamlessly. To investigate the old cluster, we filter using brushing in the detailed view, see Figure 3. In the overview, we see that it has indeed been split into two clusters. However, the clusters are also identified in subspaces that contains dimensions where the clusters are not dense. We conclude that l is too high and we look at the preview scatter plots in the l-menu. Again, changing the previews requires computing new projected clusterings, but happens seamlessly. We pick $l = 6$ and in the overview, we see that the clusters only appear in the scatter plots where the clusters are dense, see Figure 4. Again, the last update requires two projected clusterings but occurs smoothly.

At last, we switch to computing the visualization on the CPU, to show the difference in interaction. Now the time to compute

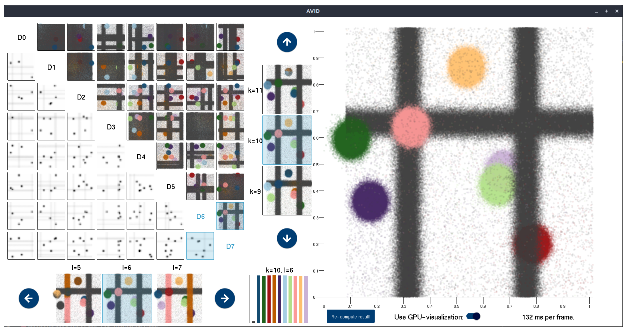

Figure 4: AVID showing the final clustering.

one frame increased from around a hundred milliseconds to a second.

5 CONCLUSIONS

We presented a demo of GPU-FAST-PROCLUS and a new data visualization that supports the exploration of parameters for the projected clustering algorithm PROCLUS. The fast GPU-parallelized version of PROCLUS makes the interaction real-time. To avoid funneling the results from the GPU through the bottleneck of the CPU and main memory just to transfer the visualization back to the GPU to be displayed, we implemented the pipeline for both clustering analysis and computing the data visualization fully on the GPU. This means that what happens on the GPU stays on the GPU. The proposed solution is a case study for GPU-FAST-PROCLUS, but could easily be adapted to fit any GPU-parallelized (projected) clustering approach.

ACKNOWLEDGMENTS

This work was supported by Independent Research Fund Denmark.

REFERENCES

[1] Yaw Adu-Gyamfi. 2019. GPU-Enabled Visual Analytics Framework for Big Transportation Datasets. *Journal of Big Data Analytics in Transportation* 1, 2-3 (2019), 147–159. https://doi.org/10.1007/s42421-019-00010-y

[2] Charu C Aggarwal, Joel L Wolf, Philip S Yu, Cecilia Procopiuc, and Jong Soo Park. 1999. Fast algorithms for projected clustering. In *Proc. of ACM SIGMOD International Conference on Management of Data*. ACM, 61–72. https://doi.org/10.1145/304182.304188

[3] Niklas Elmqvist, Andrew Vande Moere, Hans-Christian Jetter, Daniel Cernea, Harald Reiterer, and TJ Jankun-Kelly. 2011. Fluid interaction for information visualization. *Information Visualization* 10, 4 (2011), 327–340. https://doi.org/10.1177/1473871611413180

[4] Alex Endert, M. Shahriar Hossain, Naren Ramakrishnan, Chris North, Patrick Fiaux, and Christopher Andrews. 2014. The human is the loop: New directions for visual analytics. *Journal of Intelligent Information Systems* 43, 3 (2014), 411–435. https://doi.org/10.1007/s10844-014-0304-9

[5] Jean-Daniel Fekete and Catherine Plaisant. 2002. Interactive information visualization of a million items. In *Proc. of the IEEE Symposium on Information Visualization (InfoVis)*. IEEE, 117–124. https://doi.org/10.1109/INFVIS.2002.1173156

[6] Jakob Rødsgaard Jørgensen, Katrine Scheel, Ira Assent, Ajeet Ram Pathak, and Anne C Elster. 2022. GPU-FAST-PROCLUS: A Fast GPU-parallelized Approach to Projected Clustering. In *EDBT*.

[7] Todd Mostak. 2016. Using GPUs to accelerate data discovery and visual analytics. In *Proc. of the Future Technologies Conference (FTC)*. IEEE, 1310–1313. https://doi.org/10.1109/FTC.2016.7821771

[8] Raymond T. Ng and Jiawei Han. 2002. CLARANS: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering* 14, 5 (2002), 1003–1016. https://doi.org/10.1109/TKDE.2002.1033770

[9] Eric Papenhausen, Bing Wang, Sungsoo Ha, Alla Zelenyuk, Dan Imre, and Klaus Mueller. 2013. GPU-accelerated incremental correlation clustering of large data with visual feedback. In *Proc. of the IEEE International Conference on Big Data*. IEEE, 63–70. https://doi.org/10.1109/BigData.2013.6691716

[10] Donghao Ren, Bongshin Lee, and Tobias Höllerer. 2017. Stardust: Accessible and Transparent GPU Support for Information Visualization Rendering. *Computer Graphics Forum* 36, 3 (2017), 179–188. https://doi.org/10.1111/cgf.13178

[11] Ben Shneiderman. 1994. Dynamic queries for visual information seeking. *IEEE Software* 11, 6 (1994), 70–77. https://doi.org/10.1109/52.329404

[12] Andrada Tatu, Fabian Maaß, Ines Färber, Enrico Bertini, Tobias Schreck, Thomas Seidl, and Daniel Keim. 2012. Subspace search and visualization to make sense of alternative clusterings in high-dimensional data. In *Proc. of the IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 63–72. https://doi.org/10.1109/VAST.2012.6400488

[13] Christian Tominski and Heidrun Schumann. 2020. *Interactive Visual Data Analysis*. A K Peters/CRC Press. https://doi.org/10.1201/9781315152707

[14] Daniel Weiskopf. 2007. *GPU-based interactive visualization techniques*. Springer. https://doi.org/10.1007/978-3-540-33263-3

[15] Xiaoru Yuan, Donghao Ren, Zuchao Wang, and Cong Guo. 2013. Dimension Projection Matrix/Tree: Interactive Subspace Visual Exploration and Analysis of High Dimensional Data. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2625–2633. https://doi.org/10.1109/TVCG.2013.150