Parallel Nonlinear Dimensionality Reduction Using GPU Acceleration

Yezihalem Tegegne[1](✉), Zhonglin Qu[1], Yu Qian[2], and Quang Vinh Nguyen[3]

[1] School of Computer, Data and Mathematical Sciences, Western Sydney University, Penrith, Australia
{19201971,18885806}@student.westernsydney.edu.au
[2] J. Craig Venter Institute, La Jolla, CA, USA
mqian@jcvi.org
[3] MARCS Institute and School of Computer, Data and Mathematical Sciences, Western Sydney University, Penrith, Australia
q.nguyen@westernsydney.edu.au

Abstract. Dimensionality reduction is usually an essential step in data mining and classical machine learning from high-dimensional data. Uniform Manifold Approximations Projections (UMAP) is a recently developed nonlinear dimensionality reduction method that is being widely applied in biomedical informatics. However, the UMAP implementation is still not efficient enough for processing the recent big omics data from biomedicine. This paper proposes and implements a method that reduces UMAP runtime using GPU-acceleration on the GPU-RAPIDS platform. Our experiments showed that the parallel UMAP implementation performed hundred times faster than the original UMAP implementation on a cluster computer, while maintaining the effectiveness on identifying leukemic cells from clinical flow cytometry data.

Keywords: Dimensionality reduction · UMAP · GPU-RAPIDS · GPU · cuML · Flow cytometry · Machine learning

1 Introduction

Classical data pre-processing techniques [1] have become slow for big data generated in biomedicine and real world applications, due to both increasing data volume and high data dimensionality. Extracting and visualizing patterns from high-dimensional big data through traditional pairwise comparisons using scatterplots or dot plots can be inefficient and sometimes even infeasible due to curse of dimensionality.

Dimensionality reduction methods project high-dimensional data to low-dimensional data, usually 2D or 3D for straightforward analysis and interpretation of patterns such as data clusters. Linear projection methods, such as Principle Component Analysis (PCA), have been recognized as a basic yet effective method to support high throughput cell biology [2, 3]. While the linear projection approach does not usually capture non-linear relationships which may exist in omics data [4], it can be used as an early process before applying more advanced dimensionality reduction methods [5]. Non-linear dimensionality reduction methods have recently gained popularity in biomedical data analysis due to their ability to identify novel non-linear patterns. T-distributed Stochastic Neighbour Embedding t-SNE [6] (or viSNE [7]) and Uniform Manifold Approximations Projections (UMAP) [8, 9] are among the most commonly used techniques [4, 5]. For example viSNE has been used to identify abnormal T-cell populations in routine clinical flow cytometry data [10]. The recently introduced UMAP has gained popularity due to its fast run time to scale to handle large numbers of cells and dimensions [5, 9].

The non-linear dimensionality reduction methods, such as UMAP are normally implemented to run on Central Processing Units (CPUs) and they are not optimized for parallelization. For example, the UMAP method was originally designed to be compatible with Scikit-learn, NumPy, and Pandas on the CPUs to transfer and visualize data [8]. Scikit-learn [11] is a python module integrating a wide range of state-of-the-art supervised and unsupervised machine learning algorithms that also support data processing and manipulation, such as classification, regression, clustering, model selection. NumPy [12] is an open python module that provides mathematical computation on large multi-dimensional matrices and arrays. And Pandas [13] is an open-source library for data analysis such as reading and writing data between in-memory data structure and different formats such as CSV, text files, and database.

However, these libraries are only support of a portion of their CPU equivalents and work with the data sets that fit in the device memory [14]. For example, running a big data set with Pandas and applying a massive matrix computation with NumPy are time taking. This makes CPU-based dimensionality reduction methods relatively ineffective to process data sets with millions or more items, which limits their capability for real-time or near real-time analysis. Therefore, it is required a new and better mechanism to perform UMAP much faster than the current implementation system to handle such huge omics data sets.

We introduce a new method that optimises the implementation of UMAP algorithm with parallel processing on GPU-RAPIDS platform. Graphical Processing Units (GPUs) are more effective and powerful for parallel processing thanks to their high number of cores per unit in comparison with CPUs. We improve the computational speed significantly which is a hundred times faster or more. While UMAP is a random process and each time the layout is slightly different, our method also preserves similar outputs in comparison with the original CPU-based algorithm.

2 GPU-RAPIDS Platform for UMAP

2.1 GPU-RAPIDS Platform

The advancement of GPU computing is Nvidia's CUDA solution which provides both the CUDA programming language and the enabled hardware computational engine. This platform has a highly parallel architecture with hundreds of cores and very high memory bandwidth, which enables the parallel computation of machine learning models [15]. RAPIDS [16] is an open-source developed by Nvidia and contains APIs and python libraries that allow for executing algorithms python jobs on GPU to speed up machine learning algorithms. And for this, GPU acceleration has become more and more important. In addition, to execute and accelerate tasks using GPU, RAPIDS provides interoperability with other open-source tools for machine learning and deep learning workflows [17]. We use cuDF [18] instead of Pandas, and cuML [19] instead of Scikit-learn to implement UMAP on the GPU-RAPIDS. CuDF is a Python GPU DataFrames for data handling and manipulation. CuML is a Python GPU machine learning library that contains machine learning algorithms that Scikit-Learn has, all in a very similar format but on GPU.

2.2 Constructing k-NN Graph: UMAP-Learn vs CuML-UMAP

Most dimension reduction algorithms can be categorized as either matrix factorization or graph layout. The current implementation system UMAP-Learn uses the nearest neighbours descent algorithm for the construction of an approximate nearest neighbours graph and stochastic gradient descent (SGD) algorithm for optimization. Within our knowledge, UMAP-Learn is only implemented on a single core without the GPU-accelerated version. Tree-based approximate variants, such as algorithms available in Scikit-Learn, also do not have a straightforward port to the GPUs. This is due to the bottleneck of computing extraction of k-NN graph for a large data sample. In order to construct the initial high-dimensional graph, UMAP builds a "fuzzy simplicial complex" graph which represents a weighted graph [8]. The edge weight reflects the likelihood that two nodes are connected to form an edge.

2.3 Get the Nearest Neighbours

A value in a data set is mathematically a vector, and most of the time the rows are made up of numerical values. We calculate the distance between two rows or two vectors to draw a straight line. The straight-line distance between two vectors can be calculated by using Euclidean distance formula, which is the square root of the sum of the squared difference between two vectors using the following formula:

$$\sqrt{\sum_{i}^{N}(x_{1i}-x_{2i})^2}$$

Where x_1 and x_2 are the first and the second rows of the data respectively, and i is the index to a specific column as we sum across all the columns in the data set and N is the total number of dimensions.

From the distance computation process above, we determine the k-closest instances for the data. We next identify the nearest neighbours in the algorithm. Firstly, we extend a radius outward from each data point to connect other points. When the radii overlap, a weighted k-nearest neighbour (k-NN) graph is constructed. The construction of the graph can be performed via a nearest neighbour or approximate nearest neighbour search algorithm. The computational time for this k-NN graph may take up to 26% of the total time of UMAP-learn [20], which is claimed the second-largest computational bottleneck of the UMAP algorithm, next to the optimization of the embedding process.

2.4 k-NN Graph Optimisation on GPU

The first phase of UMAP algorithm is the construction of a weighted k-NN graph. The bottleneck of a k-NN implementation on a single core is the distance calculation and the selection of k nearest neighbour observation. This problem could be addressed by changing a sequential program into a parallel process, which increases the speed of computation. Implementing k-NN on GPU enables a dramatic computational performance improvement due to the high parallelization nature of the arithmetic operation. Under UMAP-learn k-NN is set to automatically chooses the algorithm based on the size and structure types of the input data set, such as either a brute force search or k-d tree or ball tree search. However, this process is slow and time taking.

The distance calculation process can be fully parallelized on GPU-RAPIDS, hence every thread can involve in the computation of finding the distance between the nearest data points. For large data points, many threads and blocks are launched to perform the computation simultaneously. Once the distance is calculated, the distance kernel is invoked, and the results are stored in shared memory.

The next task is to find the best k-nearest neighbours based on the calculated distance. Then each thread considers one distance simultaneously to calculate and to find the k-nearest neighbours. This first step process saves a significant amount of time when running the UMAP algorithm.

2.5 Embedding Optimisation

Embeddings are robust representations of data modalities like text, images, sound, etc. Essentially, they are vectors of relatively lower dimensions, that can capture original information and it gets a lot easier if we operate on the data using their embeddings representations. The curse of dimensionality complicates research and development works such as in Omics data analytics [21]. UMAP is very efficient at identifying subpopulations in large high dimensional data sets. It scales well with both input dimension and embedding dimension. As mentioned in previous sections, UMAP is comprised of two important steps including first computing a graph representing our data, and second learning an embedding for the graph. The UMAP algorithm uses a binary cross-entropy as a cost function and utilises a stochastic gradient descent to minimise the cost function to optimise the step performance.

The first step of the embedding optimization stage is to initialise the array of output embedding which can follow both random and spectral initialisation strategies. The current implementation uses a spectral embedding of the fuzzy union through the nearest-neighbours variant of the Laplacian eigenmaps algorithm. Our new RAPIDS UMAP method uses spectral clustering implementation from cuGraph, a RAPIDS library that collects GPU accelerated graph algorithms [22]. Overall, we enable parallel processing to perform different UMAP computation steps simultaneously.

UMAP uses binary cross-entropy (CE) as a cost function associated with the model, which can measure how well a model performs on the training data with an aim to minimize the cost function in a way that it comes as close to zero as possible. Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function that minimizes a cost function. Hence gradient descent can be slow to run on very large datasets due to one iteration of the gradient descent algorithm requires a prediction for each instance in the training dataset, it can take a long time when we have many millions of instances. To overcome this problem UMAP uses a modified version of gradient decent called stochastic gradient descent, which calculate the gradient using just a random small part of the observations instead of all of them using batch and this approach can reduce computation time. UMAP computation and update operations have been fused into a single kernel and parallelized so that each thread processes one edge of the fuzzy union. The CUDA kernel is scheduled iteratively for n_epochs which means *n* times of complete passes of the entire training dataset passing through the training or learning process of the algorithm to compute and apply the gradient updates to the embeddings in each epoch. The dependencies between the vertices in the updating of the gradients make this step non-trivial to parallelize efficiently. This decreases the potential for coalesced memory access and creates the need for atomic operations when applying gradient updates. When the k-NN graph is pre-computed, this step can comprise up to 50% [23] of the remaining time spent in the computation. Therefore, the parallelization of the k-NN graph construction process can speed up the performance of the UMAP algorithm significantly.

3 Benchmark Experimental Comparison

We run our new UMAP algorithm on the RAPIDS-GPU platform using various MINST data sets to evaluate its output quality and its computational time. Our experiments indicated that the new RAPIDS-GPU UMAP method has outperformed the CPU method which is over a hundred times faster while producing comparable outputs (see Table 1).

We present two experiments on the F-MNIST [24] and MNIST [25] digits data sets (collected from https://github.com/zalandoresearch/fashion-mnist) in this paper. Our experiments were carried in the Comet platform, an eXtreme Science and Engineering Discovery Environment (XSEDE) cluster with 24 nodes (https://www.sdsc.edu/support/user_guides/comet.html). Each node uses Intel Xeon E5-2680v3 processors, 128 GB DDR4 DRAM, and 320 GB of SSD local scratch memory. The GPU nodes contain four NVIDIA GPUs each. We only use one node in our computational experiment.

Table 1. Datasets used in experiment and computational time for UMAP methods.

Dataset	Rows	Columns	Running time (in seconds) CPU UMAP	Running time (in seconds) RAPIDS UMAP	Improvement
Fashion Minst	70,000	784	260	1.9	>135 times
Minist (Digits)	70,000	784	240	2.3	>104 times
Omics data sets	1.5 million	16	4000	30	>130 times
	5 million	16	too long	960	

3.1 F-MNIST Data Set

F-MNIST (or Fashion MNIST) [24] is a 10-class dataset of 70,000 28 $\times$ 28 pixels grayscale images of fashion items (clothing, footwear, and bags), i.e., 70,000 items with 784 dimensions for classification into 10 categories.

We run the experiments on the existing UMAP method on the CPU and on the new GPU-RAPIDS UMAP method respectively. The UMAP projection aims to identify groups of garments in the two-dimensional space. The vectors of pixels for each image are used as input and the UMAP maps them to the two-dimensional space for each image. Here we can see clearly, the important structural properties of the data have been retained while the known classes have been cleanly pulled apart and isolated. For example, the pants, t-shirts and bags are both retained their shape and internal structure (see blue and red items respectively in Fig. 1).

Figures 1a and 1b show the output from the standard UMAP and from the new GPU-enabled UMAP-RAPIDS methods, respectively. The colours are coded according to the clothing category of the original input (e.g. boots are blue, sneakers are green, sandals are yellow, and trousers are red colours). The visualisation clearly shows that a similar output between the existing UMAP and the new GPU enabled UMAP methods. However, the run time on the GPU method is much faster than the current implementation of UMAP on the CPU. Particularly, the running time of UMAP under GPU-RAPID (Fig. 1b) is 1.9 s in comparison to over 260 s on the original CPU platform.

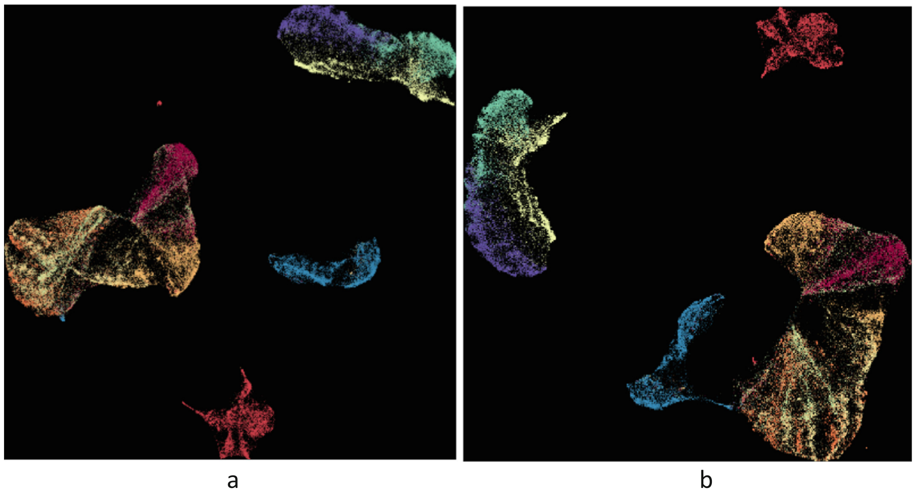

Fig. 1. The two-dimensional visualisations of UMAP methods on Fashion MNIST dataset, where (a) shows the existing UMAP output (run on CPU) and (b) shows the new UMAP output (run on RAPIDS-GPU platform). It takes over 260 s to generate the output in Fig. 1(a) in comparison with just 1.9 s to generate the output in Fig. 1(b). (Color figure online)

3.2 MNIST Digits Data Set

The MNIST [25] digits data set contains handwritten digits (from 0 to 9) with 70,000 images of digits with 784 dimensions. We use UMAP to identify digits in the data set. After applying UMAP on the data, we find that without any labels, UMAP manages to separate well the data. The outputs are similar between UMAP-learn and UMAP-RAPIDS. Figure 2 shows there are clear 10 clusters that are colour-coded by digit types (0 to 9). However, the performance is much faster for UMAP-RAPIDS (2.3 s) which is more than 100 times faster than the original UMAP (240 s).

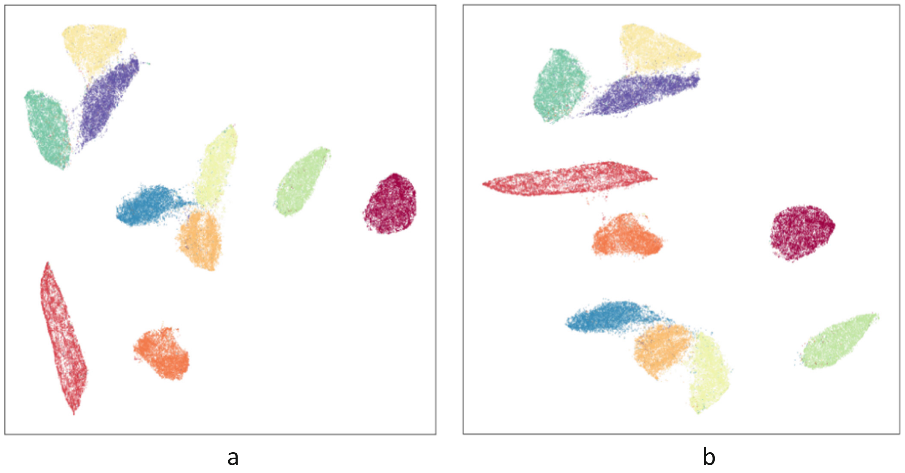

Fig. 2. The two-dimensional visualisations of UMAP methods on the MNIST digits dataset, where Fig. 2a shows the output of UMAP on CPU) and Fig. 2b shows the output of the new UMAP on the RAPIDS-GPU platform. The visualisation shows 10 groups shown in different colours representing 10 digits. It took 240 s to generate the output in Fig. 2a in comparison with just 2.3 s to generate the output in Fig. 2b. (Color figure online)

4 A Case Study on Leukemia Diagnostic Data Analysis

We applied our method to generate UMAP visualization of high-dimensional clinical flow cytometry data for diagnosis of Chronic Lymphocytic Leukemia (CLL). The dataset we used is downloaded from FlowRepository (http://flowrepository.org/) as used and published in a previous study [26]. The goal is to demonstrate the effectiveness of the UMAP dimensionality reduction for supporting visual identification of CLL leukemic cells so that the CLL cases can be visually separated from the non-CLL cases. Each sample contains 10 protein molecule markers and 6 scatter parameters (16 dimensions in total) measured on approximate 100,000 cells (rows). The markers in the reagent panel included CD3, CD5, CD10, CD19, CD22, CD38, CD43, CD45, CD79b and CD81. The scatter attributes include area/height/width for both forward scatter and side scatter parameters (FSC-A/W/H and SSC-A/H/W). To identify CLL cases, we focused on identifying cells with the leukemic phenotype CD5^{+}, CD19^{+}, CD10^{-}, and CD79b$^{\mathrm{dim}}$ from the flow cytometry data as defined in the study in [27]. Our analytical workflow of the CLL case studies is illustrated in Fig. 3.

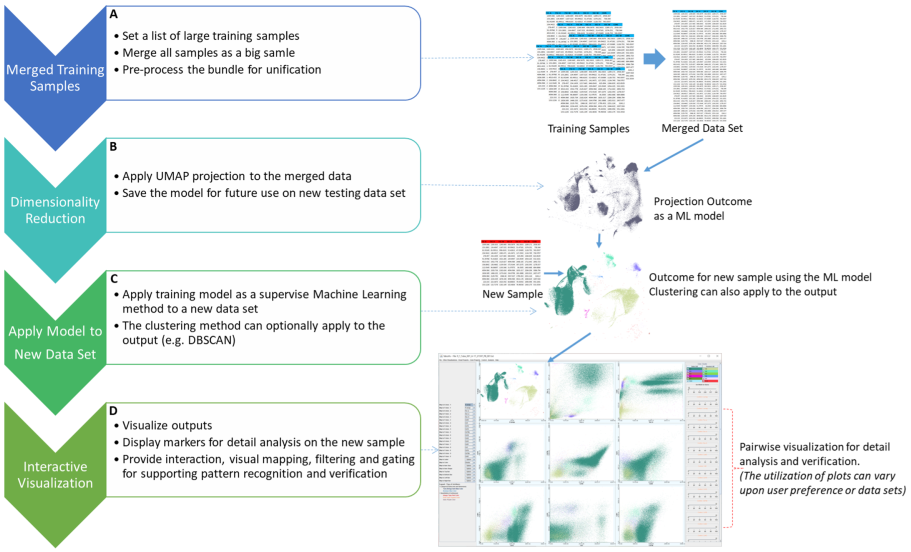
Merged Training Samples
A
• Set a list of large training samples
• Merge all samples as a big samle
• Pre-process the bundle for unification
Training Samples
Merged Data Set
Dimensionality Reduction
B
• Apply UMAP projection to the merged data
• Save the model for future use on new testing data set
Projection Outcome as a ML model
Apply Model to New Data Set
C
• Apply training model as a supervise Machine Learning method to a new data set
• The clustering method can optionally apply to the output (e.g. DBSCAN)
New Sample
Outcome for new sample using the ML model
Clustering can also apply to the output
Interactive Visualization
D
• Visualize outputs
• Display markers for detail analysis on the new sample
• Provide interaction, visual mapping, filtering and gating for supporting pattern recognition and verification
Pairwise visualization for detail analysis and verification.
(The utilization of plots can vary upon user preference or data sets)

Fig. 3. The flow diagram of the analytics process in the CLL case study includes A) the data processing and merging, B) creating training model(s) with UMAP, C) produce the transformation on the new sample(s) using the saved model, and D) provide interactive visualisation with scatterplots for better comprehension, analysis and verification of the outcome.

We first merged a set of selected CLL samples into one large template for training purpose, resulting in a matrix with the same number of dimensions and the concatenations of millions of rows, with each row representing a cell. We next applied the UMAP non-linear dimensionality reduction method to the merged data. This process projected the 16-dimensional data to a two-dimensional data space for visualization. The template together with UMAP parameters used is then saved for supervised projection of new samples. We repeated this process on different sets and sizes of training samples to generate multiple projection models. This process took most of the computation time since it each time processed a big set of merged data.

For each new diagnostic sample in the analysis, we applied the saved UMAP model as a supervised machine learning process to classify the cellular events onto the 2D UMAP space. A data clustering method, such as DBSCAN [28], is optionally applied to the 2D output for identifying and color-coding of the cell subpopulations. Finally, we implemented interactive visualisation to provide 2D plotting visual presentation on both the UMAP-transformed space and the pairwise combination of original markers with biological meaning side by side for visual pattern recognition and verification. The visualisation utilises the TabuVis tool [29, 30] to provide multiple scatterplots, visual mapping, filtering and interaction enabling effective visual analysis of the data.

We have run the experiments on different sizes of pooled data and training samples. In the first experiment, we carried out the training experiment with 15 CLL samples. After merging all training samples into one large data set (i.e. 1.5 million rows and 16 dimensions), we have applied the UMAP method into the processed data set. The running time on our new UMAP method on the GPU-RAPIDS platform takes approximately 30 s. This is more than a hundred times faster in comparison with over 4000 s when running the same process on a regular CPU system.

In the second experiment, we carried out the training experiment with much larger data sets of 50 CLL samples. We hypothesise that having a much larger number of samples in the training would potentially produce more consistent outcomes and expose better subpopulations in the testing samples. After merging all training samples into a large data set (i.e. 5 million rows and 16 dimensions), we applied the UMAP method into the processed data set. The running time on our new UMAP method on the GPU-RAPIDS platform takes approximately 16 min. Unfortunately, the same process on the CPU system takes multiple days to run so that we had to abort this experiment.

From the saved training models, we can run a new sample independently on a selected model to identify the cancer cells in the data set. This step is much quicker than the training process because we do not need to compute on the much larger merged data. We were able to carry out the computational analysis within a few seconds, which is crucial to enable the real-time diagnostics and analysis of the CLL cancel data. Particularly, our experiments show that the running time GPU-RAPIDS on reading the saved model is 0.4 s for reading a UMAP model with 15 merged samples and less than 1 s for the UMAP model with 50 merged samples respectively. It also takes approximately 3 s for transforming a new CLL data set (100,000 rows and 16 dimensions). The running time for DBSCAN is also less than 1 s. Therefore, the whole computational analysis process takes less than 5 s.

The outcomes of our new UMAP method on GPU-RAPIDS are consistent with the UMAP by CPU method, while the processing time using GPU is more than a hundred times faster than the latter. We can identify quickly the CLL cases as well as the non-CLL cases from the whole dataset in a few minutes. Further from the study in [27], our results are also validated by an domain expert to ensure the accuracy in the diagnosis outcomes of CLL cases.

Figure 4 illustrates the outputs of the projection methods using the training model with 15 CLL samples. We use pairwise scatterplots to show the UMAP output and the selected markers and scatters, where Fig. 4a uses the regular UMAP and Fig. 4b uses our new UMAP method on GPU-RAPIDS. In each figure, the first plot shows the projection output (x-axis and y-axis) and DBSCAN clustering (colours) of the new CLL sample, and other plots show pairwise scatterplots of selected markers and scatter parameters of the new sample, particularly (FSC-A, SSC-A), (SSC-H, CD45), (CD5, CD19), (CD10, CD79b), (CD38, CD22), (CD3, CD81), (CD3, CD19) and (CD79b, CD81) on each x-axis and y-axis respectively. The visualization also highlights the identified CLL cancer cells in red (CD5+, CD19+, CD10−, CD79bdim). The cancer cells are also highlighted simultaneously on other pairwise scatterplots for complete verification and analysis. Both UMAP implementations, as shown in Figs. 4a and 4b, support visual identification of the CLL cells for the new patients effectively.

Figure 5 presents another example of our projection methods using the same training model as in Fig. 4 with 15 CLL samples on a CLL case with high tumour burden, comparing the original UMAP (Fig. 5a) and the GPU-accelerated UMAP implementation on GPU-RAPIDS (Fig. 5b). Both methods can consistently identify the CLL cells for the testing sample. The detailed views show clearly the CLL cells in the 3 pairwise 2D plots (CD5, CD19), (CD10, CD79b) and (CD38, CD22).

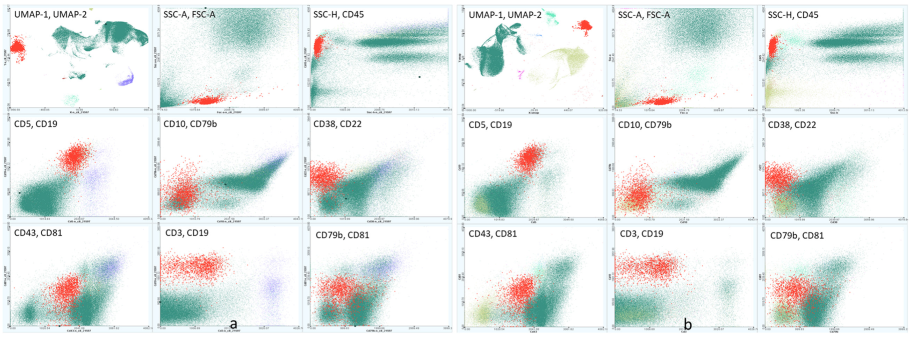

Fig. 4. An example of scatterplots visualizations showing the UMAP projection outcome using the training model with 15 CLL samples. The UMAP outputs are shown in the first panel, and the selected markers and scatters are shown in the other panels. Figures 4a and 4b present the results from the UMAP algorithms on CPU and GPU-RAPIDS respectively.

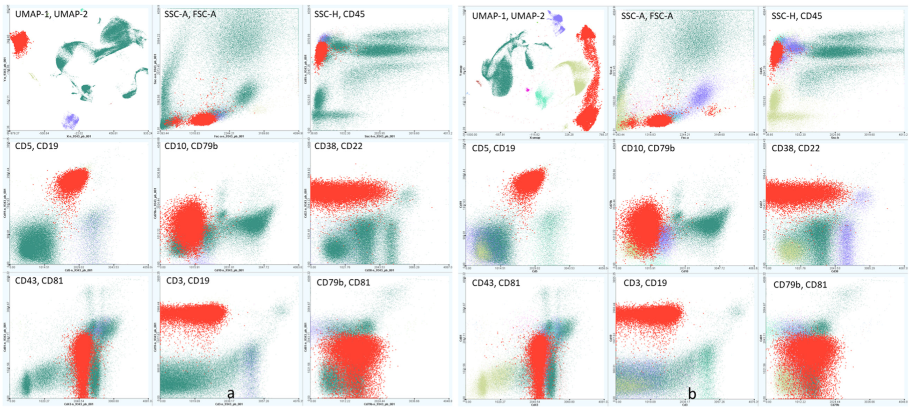

Fig. 5. An example scatterplot visualization showing the UMAP projection on a sample with a large number of cancer cells.

Figure 6 presents the outputs of our UMAP implementation when more training samples were used for the UMAP-based classification. 50 CLL samples were merged to create a UMAP template. The original UMAP implementation was too slow to process such a big input file. Figure 6a indicated 8.9% of CLL cells were found in the testing sample. In Fig. 6b, only a few cells were seen in the UMAP cancer cell region. The results are consistent with the clinical diagnosis labels where the case in Fig. 6a is CLL positive and the case in Fig. 6b non-CLL.

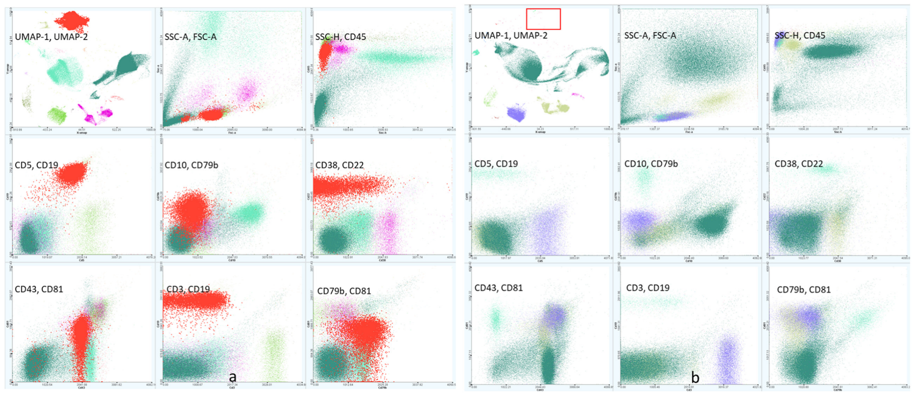

Fig. 6. Two examples of scatterplots visualizations showing the UMAP projection using 50 CLL training samples. Figure 6a show near 8.9% CLL cancer cells in the sample, and Fig. 6b shows a non-CLL case where it shows no cancer cells in the region.

5 Conclusion

We have designed and implemented the UMAP dimensionality reduction on the GPU-RAPIDS platform, which significantly improved the speed of the original UMAP.

We optimised the acceleration and parallel processing with GPUs to overcome to computational bottleneck in the graph construction and distance calculation from the UMAP algorithm. Using MNIST/F-MNIST benchmark datasets, our experiments show that the new method can run a hundred times faster or better, and it also produces stable and consistent outputs in comparison with the original UMAP. We demonstrate the effectiveness of our work with case studies on identifying the cancer cells for clinical diagnosis of leukemia, which will ultimately help real-time application of machine learning diagnosis for biomedicine and the results are also meaningful for other graph and manifold learning algorithms that use GPUs.

References

1. Bendall, S.C., Nolan, G.P., Roederer, M., Chattopadhyay, P.K.: A deep profiler's guide to cytometry. Trends Immunol. **33**, 323–332 (2012)
2. Haghverdi, L., Buettner, F., Theis, F.J.: Diffusion maps for highdimensional single-cell analysis of differentiation data. Bioinformatics **31**, 2989–2998 (2015)
3. Ringnér, M.: What is principal component analysis? Nat. Biotechnol. **26**(3), 303–304 (2008)
4. Konstorum, A., Jekel, N., Vidal, E., Laubenbacher, R.: Comparative analysis of linear and nonlinear dimension reduction techniques on mass cytometry data. bioRxiv 273862 (2018)

5. Luecken, M.D., Theis, F.J.: Current best practices in single-cell RNA-seq analysis: a tutorial. Mol. Syst. Biol. **15**(6), e8746 (2019)
6. Maaten Lvd, Hinton, G.: visualizing data using t-SNE. J. Mach. Learn. Res. **9**, 2579−2605 (2008)
7. Amir, E.D., et al.: viSNE enables visualization of high dimensional single-cell data and reveals phenotypic heterogeneity of leukemia. Nat. Biotechnol. **31**(6), 545–552 (2013)
8. McInnes, L., Healy, J., Melville, J.: UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv:180203426 [statML] (2018)
9. Becht, E., et al.: Dimensionality reduction for visualizing single-cell data using UMAP. Nat. Biotechnol. **37**(1), 38–44 (2018)
10. DiGiuseppe, J.A., Cardinali, J.L., Rezuke, W.N., Pe'er, D.: PhenoGraph and viSNE facilitate the identification of abnormal T-cell populations in routine clinical flow cytometric data. Cytometry B Clin. Cytometry **94**(5), 588–601 (2018)
11. Pedregosa, F.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
12. NumPy (2022). https://numpy.org
13. team Tpd: pandas-dev/pandas: Pandas. In: latest edn: Zenodo (2020)
14. Yuan, G., Palkar, S., Narayanan, D., Zaharia, M.: Offload annotations: bringing heterogeneous computing to existing libraries and workloads. In: Annual Technical Conference (ATC 20), pp. 293–306 (2020)
15. Adu-Gyamfi, Y.: GPU-enabled visual analytics framework for big transportation datasets. J. Big Data Anal. Transp. **1**(2–3), 147–159 (2019). https://doi.org/10.1007/s42421-019-00010-y
16. RAPIDS: The Platform Inside and Out (2022). https://developer.download.nvidia.com/video/gputechconf/gtc/(2019)/presentation/s9577-rapids-the-platform-inside-and-out.pdf
17. Aguerzame, A., Pelletier, B., Waeselynck, F.: GPU acceleration of PySpark using RAPIDS AI. In: DATA (2019)
18. Lindholm, E., Nickolls, J., Oberman, S., Montrym, J.: NVIDIA tesla: a unified graphics and computing architecture. IEEE Micro **28**(2), 39–55 (2008)
19. Ocsa, A.: SQL for GPU data frames in RAPIDS Accelerating end-to-end data science workflows using GPUs. In: LatinX in AI Research at ICML (2019)
20. Nolet, C.J., Lafargue, V., Raff, E., Nanditale, T., Oates, T., Zedlewski, J., Patterson, J.: Bringing UMAP Closer to the Speed of Light with GPU Acceleration. arXiv:200800325 [csLG] (2020)
21. Catchpoole, D., Kennedy, P., Skillicorn, D., Simoff, S.: The curse of dimensionality: a blessing to personalized medicine. Proc. Am. Soc. Clin. Oncol. **28**(34), e723–e724 (2010)
22. Hricik, T., Bader, D., Green, O.: Using RAPIDS AI to accelerate graph data science workflows. In: IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–4. IEEE (2020)
23. Nolet, C.J., Lafargue, V., Raff, E., Nanditale, T., Oates, T., Zedlewski, J., Patterson, J.: Bringing UMAP Closer to the Speed of Light with GPU Acceleration (2020)
24. Xiao, H., Rasul, K., Vollgraf, R.J.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms (2017)
25. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
26. Ji, D., et al.: machine learning of discriminative gate locations for clinical diagnosis. Cytometry A **97**(3), 296–307 (2020). PMID: 31691488; PMCID: PMC7079150
27. Scheuermann, R.H., Bui, J., Wang, H.-Y., Qian, Y.: Automated analysis of clinical flow cytometry data: a chronic lymphocytic leukemia illustration. Clin. Lab. Med. **37**(4), 931–944 (2017). PMID: 29128077; PMCID: PMC5766345
28. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD, pp. 226–231 (1996)

29. Nguyen, Q.V., Qian, Y., Huang, M.L., Zhang, J.: TabuVis: a tool for visual analytics multidimensional datasets. Sci. China Inf. Sci. **56**, 052105:052101–052105:052112 (2013)
30. Nguyen, Q.V., Simoff, S., Qian, Y., Huang, M.L.: Deep exploration of multidimensional data with linkable scatterplots. In: 9th International Symposium on Visual Information Communication and Interaction, pp. 43–50. ACM, Dallas, Texas (2016)