Optimizing Data Locality and Termination Criterion for t-SNE

Doğa Dikbayır
Computer Science and Engineering
Michigan State University
East Lansing, USA
dikbayir@msu.edu

Balasubramaniam Shanker
Electrical and Computer Engineering
Michigan State University
East Lansing, USA
bshanker@msu.edu

Hasan Metin Aktulga
Computer Science and Engineering
Michigan State University
East Lansing, USA
hma@cse.msu.edu

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9534303

***Abstract*—The t-Distributed Stochastic Neighbor Embedding (t-SNE) is known to be a successful method at visualizing high-dimensional data, making it very popular in the machine-learning and data analysis community, especially recently. However, there are two glaring unaddressed problems: (a) Existing GPU accelerated implementations of t-SNE do not account for the poor data locality present in the computation. This results in sparse matrix computations being a bottleneck during execution, especially for large data sets. (b) Another problem is the lack of an effective stopping criterion in the literature. In this paper, we report an improved GPU implementation that uses sparse matrix re-ordering to improve t-SNE's memory access pattern and a novel termination criterion that is better suited for visualization purposes. The proposed methods result in up to 4.63× end-to-end speedup and provide a practical stopping metric, potentially preventing the algorithm from terminating prematurely or running for an excessive amount of iterations. These developments enable high-quality visualizations and accurate analyses of complex large data sets containing up to 10 million data points and requiring thousands of iterations for convergence.**

***Index Terms*—t-SNE, data visualization, big data**

I. Introduction

In recent years, as problems that apply machine-learning and deep-learning tools become increasingly complex, they require large-scale and high-dimensional training data to produce accurate results. Knowing the global and local structures in the data sets enables researchers to develop more sophisticated models that are tailored specifically for each problem, decreasing the fine-tuning costs for the models. Such exploration of the data can be done through data visualization techniques. The high-dimensionality of the input data makes visualization tasks very challenging on modern machine learning data sets. In machine learning, data usually lie on a low-dimensional manifold embedded in the original high-dimensional space. For such data sets, preserving local, pairwise similarities in the low-dimensional representation is crucial to generate high-quality visualizations [1].

The t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm [2] is a powerful method designed to overcome the issues mentioned above. It can embed high-dimensional data sets into two or three dimensions very effectively. It does so by computing the pairwise similarity distribution in the high-dimensional data set and using this information to fit a low-dimensional representation with a matching similarity distribution. The t-SNE algorithm is proven to preserve the local structures in the data sets, resulting in high-quality visualizations. Over the years, many improvements have been proposed to optimize the execution time of t-SNE. These include using the Barnes-Hut and dual-tree based t-SNE implementations [3]. The popular Barnes-Hut t-SNE, which is implemented in widely-used libraries like sklearn API [4], was later replaced with the interpolation-based t-SNE [5], significantly improving the overall execution time. Recently, Chan *et. al.* proposed BH-tSNE-cuda, a GPU implementation for Barnes-Hut t-SNE [1] and interpolation-based FI-tSNE-cuda [6] which resulted in execution time accelerations of several orders of magnitude, enabling, for the first time, the visualization of data sets containing millions of data points. However, both BH-tSNE-cuda and FI-tSNE-cuda suffer from poor data locality in the attractive forces kernel. The attractive forces kernel serves as one of the key components as it contributes to the gradient used to minimize the divergence between the high-dimensional data set and its embedding. The pairwise similarity matrix that is used to compute the attractive forces has a sparse structure as many data-points only have a limited number of neighbors. For this reason, for large data sets that require a large number of iterations for visual stabilization, the attractive forces kernel starts dominating the overall execution time. As such, there is still room for significant performance improvements.

Furthermore, existing work, including BH-tSNE-cuda and FI-tSNE-cuda, do not provide a good stopping criterion that guarantees visually converged results. Looking solely at the average gradient norm (intuitively speaking, the higher the average gradient norm should indicate the more unsettled the points are) may result in an excessive number of iterations that continue moving data-points in the low dimensional space, even after visual stabilization is achieved. This leads to the aforementioned data locality problem being increasingly evident, since significantly more iterations than that are required in practice may be performed.

In this paper, we propose a performance optimization

This work was supported by the National Science Foundation under Grant No. CCF-1822932. This research used resources of Michigan State University's High Performance Computing Center (MSU HPCC).

978-0-7381-3366-9/21/$31.00 ©2021 IEEE

that is applicable for both BH-tSNE-cuda and FI-tSNE-cuda implementations that utilizes the sparse matrix reordering idea to improve the memory access pattern of its attractive forces kernel. We also explore an effective stopping criteria, preventing excessive computations in practice. The proposed techniques result in an optimized and generic framework for analysis of very large data sets, potentially saving significant amounts of exploration time.

The outline of this paper is as follows: In Section II, we give background information on t-SNE, define the problems present in existing implementations, and discuss the motivation behind our proposed techniques to overcome these problems. In Section III, we describe our algorithms and methods, and detail their implementation. In Section IV, we evaluate the proposed techniques on both real-world and synthetic data sets, containing up to 10 million elements. We present related literature in Section V, and finally conclude and give future directions in Section VI.

II. Background and Motivation

A. The t-distributed Stochastic Neighbor Embedding Algorithm

The t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm is a powerful dimensionality reduction technique that was designed primarily for visualization purposes. In contrast to other popular dimensionality reduction techniques, t-SNE works well with real-world data sets and is able to preserve the local structure in the low-dimensional space. t-SNE computes pair-wise similarities between high-dimensional data points $X = \{x_1, x_2, ..., x_n\}$ using the k Nearest Neighbor (k-NN) information and tries to match the probability distribution of the low-dimensional points $Y = \{y_1, y_2, ..., y_n\}$ by minimizing the Kullback–Leibler (KL) divergence between the two distributions. The theoretical background of the method is described in detail in previous works [7] [2]. Below, we give a brief summary.

The pairwise similarities between points i and j in high and low-dimensional spaces are represented as two conditional probabilities $p_{j|i}$ and $q_{j|i}$, respectively. Each denotes the probability of point j being a neighbor of point i in its respective space. For $p_{j|i}$, a Gaussian distribution is centered at each high-dimensional data point in X:

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2} \tag{1}$$

Very small values for $p_{j|i}$ (instances without any nearby points) cause the position of the instance in the low-dimensional mapping to be vague. This is prevented by using joint probabilities for high-dimensional data points:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{2}$$

Similarly, a Student's t-distribution [1] with 1 degree of freedom is centered at each low-dimensional data point to model the probability distribution:

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}} \tag{3}$$

The gradient then can be computed using the following equation where C is the cost function for the KL-divergence between P and Q:

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1} \tag{4}$$

Eq. 4 can also be represented as,

$$F_{attr} = \sum_{j \in [1,..,N], j \neq i} p_{ij} q_{ij} Z(y_i - y_j) \tag{5}$$

$$F_{rep} = - \sum_{j \in [1,..,N], j \neq i} q_{ij}^2 Z(y_i - y_j) \tag{6}$$

$$\frac{\partial C}{\partial y_i} = 4(F_{attr} + F_{rep}) \tag{7}$$

where $Z = \sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}$ is the normalization term and F_{attr} and F_{rep} represent the attractive and repulsive forces between the data points, respectively.

B. FI-tSNE-cuda Implementation

Algorithm 1: FI-tSNE-cuda

Data:
$X = \{x_1, x_2, x_3, ..., x_N\}$
#iterations: $maxIter$
Result: $Y = \{y_1, y_2, y_3, ..., y_N\}$

```
init Y randomly;
find k-NN ;
compute sparse matrix P;
for t ← 1 to maxIter do
    Compute polynomial interpolant coefficients
    Compute FFT of the coefficients
    Compute the potentials for low-dimensional data points
    call atomicAdd() to accumulate sum F_attr = Σ_{j∈[1,..,N], j≠i} p_ij q_ij Z(y_i − y_j)
    ∂C/∂Y = 4(F_attr + F_rep);
    apply forces on Y
```

FI-tSNE-cuda (FFT-accelerated Interpolation-based t-SNE on GPU) is the fastest implementation of t-SNE. Algorithm 1 demonstrates the steps of the procedure. First, points are initialized in the low-dimensional space randomly. Then, the approximate k-nearest neighbor information for the high-dimensional data set is calculated on the GPU using the FAISS library [8]. The elements of the **P** matrix, which contain the p_{ij} values, are computed using the k-NN information and equation 2. The **P** matrix is stored as a sparse matrix using

[1] https://en.wikipedia.org/wiki/Student%27s_t-distribution

the coordinate (COO) format in cuSPARSE. The COO format stores only the location (row and column indices) information of the non-zero pairs (neighbors) in the sparse matrix, ignoring the zeros (non-neighbors). At each step of the optimization loop (lines 4-16), the repulsive and attractive forces are used to calculate the KL-divergence gradient (Equation 4). The details of the repulsive force computation (lines 5-7) can be found in [5]. The repulsive forces also leverage batched cuFFT calls on the GPU which improves GPU utilization significantly [6]. The attractive forces are accumulated over j (line 8) for the non-zero interactions using CUDA *atomicAdd()* calls, which are highly optimized hardware instructions in recent GPU architectures. Finally, the gradient is computed and the resulting forces are applied to the low-dimensional points, moving them to their new positions in the low-dimensional space.

The FI-tSNE-cuda code provides a full GPU based pipeline to compute t-SNE orders of magnitudes faster than previously reported implementations. Despite its excellent performance, the sparsity of the affinity matrix $\mathbf{P}$ used in the attractive forces computation often results in a poor memory access pattern. The severity of this performance problem is proportional to the size and variation in the input data set. As the size of the input data set increases, more irregular memory accesses are performed at each iteration. As the variation in the data set increases, t-SNE requires more iterations to converge, thus keeps reusing the previously described poor memory access pattern which exacerbate the impact of the poor data access patterns even further.

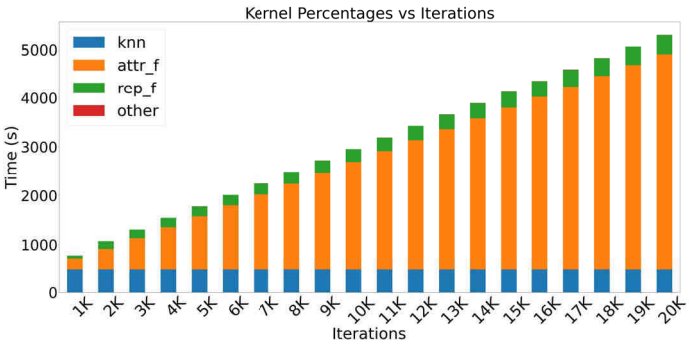

Fig. 1: The kernel percentage relative to the total execution time of the t-SNE algorithm for Deep10M data set consisting of 10 million 96-dimensional points.

The k-NN computation is performed only once, before the iterative process begins. Since the number of iterations required to reach visual stabilization varies between the input data sets, we focus on the performance of the iterative loop and attractive forces kernel. Figure 1 shows each kernel's percentages with respect to the total execution times for different number of iterations. Although the k-NN computation step dominates the total execution time for lower number of iterations, we can clearly see that the attractive forces computation is the most expensive kernel in the long-run, accounting to more than 75% of the end-to-end execution time. This serves as a motivation to focus on this portion of the implementation.

C. When to Stop?

The visual representations presented in previous works on t-SNE are generated using a fixed number of iterations, usually 1000. As mentioned before, this is an important issue because most data sets, especially large ones, require a large number of steps to produce good visuals. Figure 2 compares three 2D visualizations of the same data set containing 1 million 300-dimensional data points from the GloVe word embeddings [9] at different iterations.

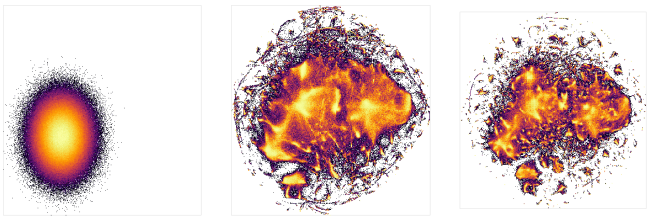

Fig. 2: 1 million GloVe word embeddings at iterations 1000, 2000 and 3000.

As it can be easily observed from this example, the three representations are qualitatively very different from each other. At 1000 iterations, even the most evident global structures are completely absent, making the visual useless. As t-SNE continues executing, more details are revealed about the data set in 2D. These details can be potentially important, depending on the target machine-learning application. Therefore being able to make an informed decision on the number of iterations to run the t-SNE algorithm is an important consideration, both from a data analysis point of view, as well as from a performance point of view.

III. Methods and Implementation

In this section, we propose new methods to overcome the data locality and stopping criterion issues described above.

A. Sparse Matrix Reordering

Sparse matrix reordering is a well studied topic in the high-performance and scientific computing community aiming at improving data-locality of large-scale applications. Methods such as the Reverse Cuthill-McKee (RCM) [10] and Approximate Minimum Degree algorithms [11] are known to improve the memory access patterns of sparse linear algebra kernels such as sparse matrix vector multiplication (SpMV) by reducing the bandwidth of the matrix.

For reordering $\mathbf{P}$, a scalable and efficient algorithm is desirable to minimize the permutation overhead. Parallel implementations of the popular RCM algorithm for both distributed [12] and shared-memory [13] systems are available in the literature and are able to provide decent speedups over the sequential version, while preserving the reordering quality. However, the RCM algorithm is breadth-first-search based, thus only considers the connections between the vertices of the

graph. This non-agglomerative approach fails to capture the community structures in the data set, which are very common for real-world data sets. Rabbit Reordering algorithm [14] on the other hand is a parallel, community-detection based, sparse matrix reordering algorithm. Rabbit Reorder rearranges the non-zero elements in the input sparse matrix as recursive dense-blocks around the diagonal. The denser the blocks get, the finer grain the corresponding community is. This approach improves cache utilization which is important for GPUs due to their hierarchical memory architectures. The finer grain blocks are loaded into higher level caches, while the global clusters reside in lower level cache memory. Moreover, Rabbit Reorder scales and performs better than the state-of-the-art parallel reordering methods, thus is practical to use for the case of large-scale data sets with t-SNE.

We integrated the multi-core Rabbit Reordering algorithm into the FI-tSNE-cuda codebase. Before starting the gradient optimization iterations, we rearrange the elements of $\mathbf{P}$ using the reordering kernel. We perform this operation only once, since the $\mathbf{P}$ matrix is computed at the beginning of the algorithm (as part of K-NN) and does not change during the course of gradient iterations. It is worth mentioning that the resulting embedding of the t-SNE algorithm is independent from this optimization. Since there are no changes in the KNN computation and accessed elements of the $\mathbf{P}$ matrix during attractive forces computation, the sequence of calculations performed in the optimization loop remain identical for matrix reordered version of t-SNE. Unfortunately, the tSNE-cuda library does not support true deterministic runs due to high amount of parallelism and third-party dependencies [2]. Thus, we cannot perform a direct comparison by feeding the same random seed to vanilla and matrix reordered t-SNE. Instead, we run matrix reordered t-SNE on data sets with known ground-truth labels and structures (see Table I), to present some form of empirical evidence to our claim. The resulting embeddings are listed in Figure 3. Even for the slightly large data sets DBPedia and Yahoo, the amount of noise is low and most of the data points belonging to the same category are tightly grouped together, as an expected result of the standard t-SNE algorithm.

Table I: Datasets with known ground-truth labels.

Name	Dims	Size	Categories
MNIST [3]	784	70K	10
Fashion-MNIST [4]	784	70K	10
CIFAR-10 [5] [15]	1024	60K	10
AG's News [6] [15]	100	120K	4
DBPedia [7] [15]	100	560K	14
Yahoo [8] [15]	100	1.4M	10

[2]https://github.com/rapidsai/cuml/issues/2980
[3]http://yann.lecun.com/exdb/mnist/
[4]https://github.com/zalandoresearch/fashion-mnist
[5]https://www.cs.toronto.edu/ kriz/cifar.html
[6]http://www.di.unipi.it/gulli/AG_corpus_of_news_articles.html
[7]https://wiki.dbpedia.org
[8]https://webscope.sandbox.yahoo.com/

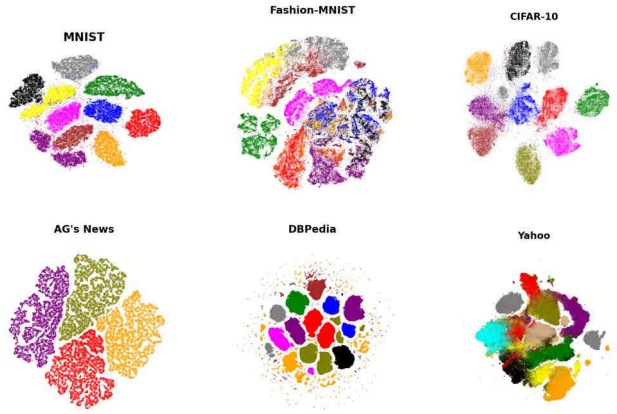

Fig. 3: Visualization of the datasets with known ground-truth labels in Table I using matrix reordered tSNE.

Figure 4 shows the sparsity patterns for $\mathbf{P}$ matrices with and without Rabbit Reordering using randomly sub-sampled 500 thousand points from the GoogleNews[9] [16] and Deep1B [10] [17] data sets. For both data sets, the temporal and spatial locality significantly improves as we observe dense blocks forming around the diagonal. Nevertheless, we still observe off-diagonal regions being filled. This is because the real-world data sets have many more inter-cluster connections which account for the noise present in the reordered sparsity patterns of the GoogleNews and Deep1B data sets. In Section IV, we evaluate the performance of the reordering on data sets containing up to 10 million data points. We discuss the effect of the reordering on both attractive forces computation and end-to-end application performance. Overall, we observe it to be a highly effective optimization.

B. A Better Termination Criterion

Modern machine learning tasks are very difficult, requiring complex models and large number of samples to prevent over-fitting the training data. This issue pushed the scientific community to design and curate data sets consisting of millions to billions of samples [18], [19]. Despite their huge sizes, visualization of these data sets is often done on a limited screen or image resolution. Therefore, for very large data sets, many points map to the same pixel. These overlaps, cause distortions in the visualization of the data. To overcome these problems, visualization libraries such as Datashader [20] use binning techniques to represent the data set by highlighting regions that contain a large number of points. Techniques like these compress the original data set, while proving themselves to be visually effective. This, however, makes the stabilization problem we discussed in Section II-C more difficult in a practical setup.

To the best of our knowledge, there is no method developed specifically for determining when to terminate the t-SNE

[9]https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/
[10]http://sites.skoltech.ru/compvision/noimi/

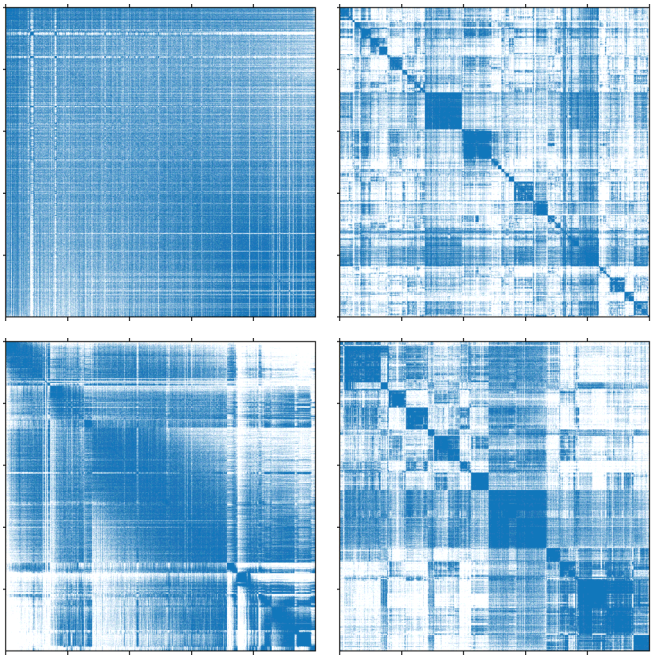

Fig. 4: Original (left) and Rabbit reordered (right) sparsity patterns of the **P** matrix for GoogleNews (first row) and Deep1B (second row) data sets.

algorithm. Existing work on t-SNE such as [6] and [5] run the algorithm for a fixed number of iterations and do not discuss the visual stabilization issue. For large data sets, however, being able to terminate the algorithm at the right moment is critical for obtaining good results without excessive computations. The latest implementation of FI-tSNE-cuda [6] has an optional parameter to determine the minimum value of the gradient norm to stop the iterative optimization loop. This parameter could easily be used to implement a patience-based early-termination mechanism, where the iterations stop if the gradient norm does not change sufficiently in a user-determined patience-interval. However, it can also become inefficient for real-world scenarios. The gradient norm is related to the forces applied to each individual point in the embedding. This causes the gradient norm to potentially be too fine granular as a visual stabilization criterion. The t-SNE algorithm is primarily utilized for exploring structures in data sets and for visualization purposes. Therefore, it is important to note that we focus on visual stabilization in this section and our analysis should not be interpreted as a numerical convergence analysis. Local (but relatively significant) changes may occur deep into the optimization cycle, even when these changes are not visually noticeable in the global structure.

Figure 5 shows the value of the gradient norm (averaged over a window of ten iterations) for t-SNE ran on the MNIST data set for 2000 iterations, and also shows three snapshots of the mapped 2D data points from iterations 300, 800 and 2000. By looking at the snapshots from iterations 300 and 800, we clearly see that the visualization drastically changes. MNIST contains 10 clusters, but, at iteration 300, we don't see all of these clusters because the algorithm did not visually stabilize yet. If we iterate for 500 more steps to step 800, we can clearly see the 10 different clusters, and members of the clusters assigned correctly. However, the gradient norm plot tells a different story. There is a steep decline around iteration 1750, which should indicate a significant change in the resulting embedding. However, if we look at the snapshots from iterations 800 and 2000, we do not see any major changes in the global structure. In other words, the necessity of the iterations between 800 and 2000 seems arguable from a practical perspective.

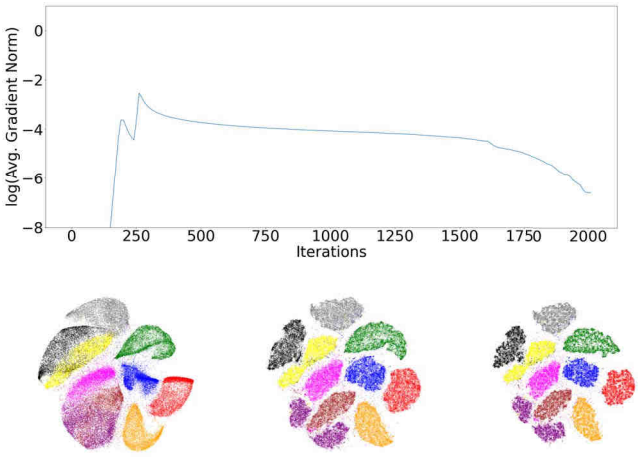

Fig. 5: The gradient norm and snapshots from iterations 300, 800 and 2000 in order.

We address the issue described above by introducing an alternative termination metric that is potentially better suited for visualization. Even with very high resolution configurations, plotting data sets containing millions of points without any overlaps is not possible. Following this intuition, we implement a simple binning approach that transforms the set of point coordinates into a 2D histogram. We divide the entire 2D space into n square bins such that $n << N$, where N is the total number of points. Each bin contains the frequency of the data points within its borders. This way, we are able to monitor the global structure of the visualization in a computationally inexpensive manner.

The 2D histogram is essentially a compressed representation of the visual and can be used as an early termination criterion, replacing the standard numerical methods. To do this, we take the Frobenius Norm (which is the matrix analog of the L2-norm of a vector) of the difference matrix between two consecutive histograms and inspect its change in a log-scale plot.

Figure 6 shows the stabilization of the proposed metric for the MNIST data set [21]. The histogram difference metric forms a clear plateau that stabilizes much earlier than iteration 1800 as indicated by the gradient norm metric. When we inspect the range of iterations where the histogram difference metric indicates stabilization (at around 600 iterations and

after), we observe visually stabilized representations of the data set. For instance, comparing the snapshots from iterations 500 (in Figure 6), and 800 and 2000 (in Figure 5), we don't see major structural changes, which supports our hypothesis. In the next section, we provide more empirical evidence in this regard.

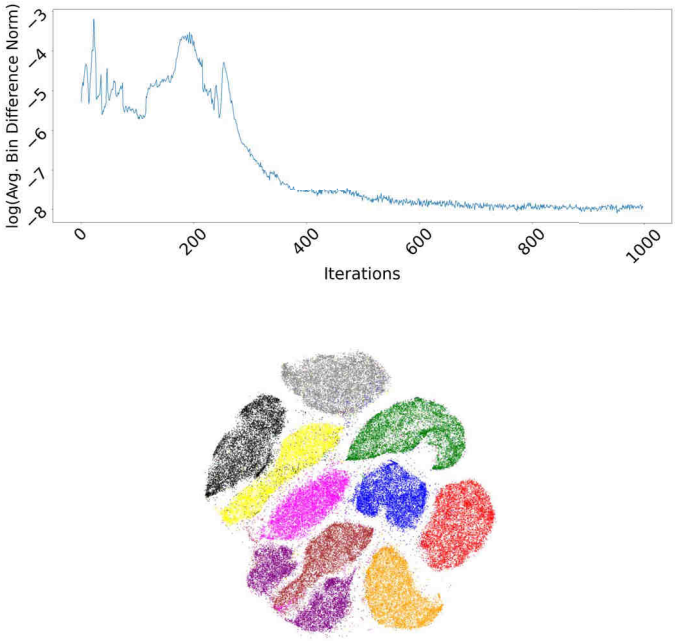

Fig. 6: The bin difference norm and MNIST snapshot at iteration 500.

We note that calculating the Frobenius norm of the histogram difference matrix is a computationally inexpensive operation compared to the t-SNE iterations and one does not have to calculate it at every iteration of the optimization loop. As such, the proposed method requires a very small overhead. Moreover, it can easily be parallelized as the required computations are simple data-parallel loops. We observe that for very large data sets, an OpenMP based parallel implementation on multi-core processors may not be able to provide real-time feedback. Therefore, we developed a simple GPU kernel that computes the Frobenius norm of the 2D histogram differences. Our implementation has negligible overhead compared to the main t-SNE loop. We again provide detailed performance evaluation results in the next section.

IV. Evaluation

In this section we evaluate the performance of the proposed methods on both synthetic and real-world data sets. The baseline implementations that we compare our optimized version to are FI-tSNE and BH-tSNE. We give more details about these algorithms in the Matrix Reordering subsection. To make the comparisons fair with respect to the results presented in literature, we fixed the values 200 and 50 for the learning rate and perplexity parameters of t-SNE, respectively, which are the default values in the vanilla t-SNE code. These parameters are important and should be carefully picked for different target data sets. However, as we mentioned previously in Section III, our optimizations do not really affect the internal mechanics of the iterative optimizations. Therefore, we do not expect a fine-tuning experiment to be in the scope of this work. Our focus in these evaluation is to analyze the speedups gained by optimizing data locality and the efficiency of the proposed stopping metric over average gradient norm for visual stabilization purposes, in experiments with fixed control parameters.

A. Experimentation Environment

All of the experiments are run on a single node with two Intel Xeon Gold 6148 CPUs (each with 20 cores) and an NVIDIA V100 32GB GPU. The V100 GPU has 5120 CUDA cores and 900GB/s memory bandwidth.

B. Data sets

- **Amazon** : The Amazon Electronics reviews data set [11]. It consists of 1689188 text reviews, embedded into 100-dimensional word vectors using FastText [22]. The pre-trained FastText model can be downloaded from here [12]
- **GloVe** : A very popular natural language processing data set, containing 2.2 million 300-dimensional word vectors trained with the GloVe [9] algorithm.
- **GoogleNews**: A text corpus containing about 100 billion words. The corpus is trained on a *word2vec* [23] model and there are in total 3 million word vectors with 300 dimensions.
- **Deep1B**: The Deep1B is a data set containing deep descriptors for 1 billion images from the Web. These descriptors are pulled from the last fully-connected layer of the GoogleNet DNN [24]. The data set creators also compressed the dimensionality from 1000 to 96. We sample 10 million deep descriptors to use in our experiments.
- **SIFT1B** : SIFT1B is a data set containing 1 billion 128-dimensional SIFT features [25] used for image recognition. We randomly sample 10 million data points for our experiments
- **Synthetic**: A 10 million point, 50-dimensional synthetic data set from ten different Gaussian distributions.

C. Stopping Criterion

In Section 2, we showed that using the gradient norm metric as a stopping criterion may result in executing a significant number of unnecessary iterations on a small instance. In this subsection, we test the proposed stopping criterion on our evaluation data sets. We compare the proposed metric to the average gradient norm by analyzing the output at different iterations. The number of iterations required for achieving visual stabilization differs depending on the target data set. This number might depend on multiple factors such as data

[11] http://jmcauley.ucsd.edu/data/amazon/
[12] https://dl.fbaipublicfiles.com/fasttext/vectors-wiki/wiki.en.vec

set size, dimensionality and structural complexity. While these properties play a role in the optimization, this subsection simply focuses on the comparison of the proposed metric with the average gradient norm metric as a stopping criterion for visual purposes. We do not present a convergence analysis for t-SNE.

We start by comparing the plots of the proposed method and gradient norm for 20 thousand iterations. We utilize these plots to determine snapshots to take from the execution and do further analysis. At this scale, standard plotting tools like the `matplotlib` library fail to represent inner dynamics in the resulting embeddings. We use the `Datashader` [13] library which is utilized in plotting large amounts of data. `Datashader` plots the density of data-points instead of explicitly drawing each one on the screen, enabling a more detailed comparison.

The proposed bin difference metric is able to capture the visual stabilization for all of the data sets, even for a data set like MNIST (Section III) that does not have many overlaps in the resulting embedding image due to its small size. Since its impact is the most evident, we evaluate the proposed metric on the larger data sets, namely the Deep10M and SIFT10M data sets. These two data sets are used to evaluate similarity queries between word and image descriptor vectors, respectively. To the best of our knowledge, there are no known visualizations of these two data sets at this scale. The data is also unlabeled. Therefore, we cannot show the categories on the plots, or present a known structure. Nevertheless, the two metrics show very different stabilization behaviours for these cases as can be seen in Figure 7. We utilize these two data sets as stretch tests for our optimizations as they contain a massive amount of data points.

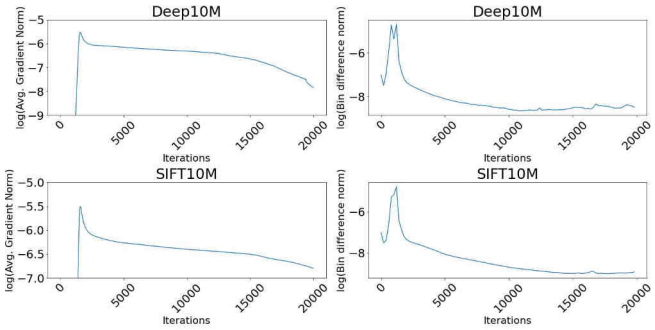

Fig. 7: Gradient Norm (left) compared to Bin Difference Norm (right), 20K iterations

Figure 8 shows the snapshots of iterations 4000, 8000, 12000 and 16000 for the Deep10M data set run. We can see that at iteration 4000, the algorithm is far from convergence. At iteration 8000, the global structure is obtained. Between iterations 8000 and 12000, we still see some noticeable changes, especially inside the south-central region of the global structure but there are no significant changes between

[13] https://datashader.org/

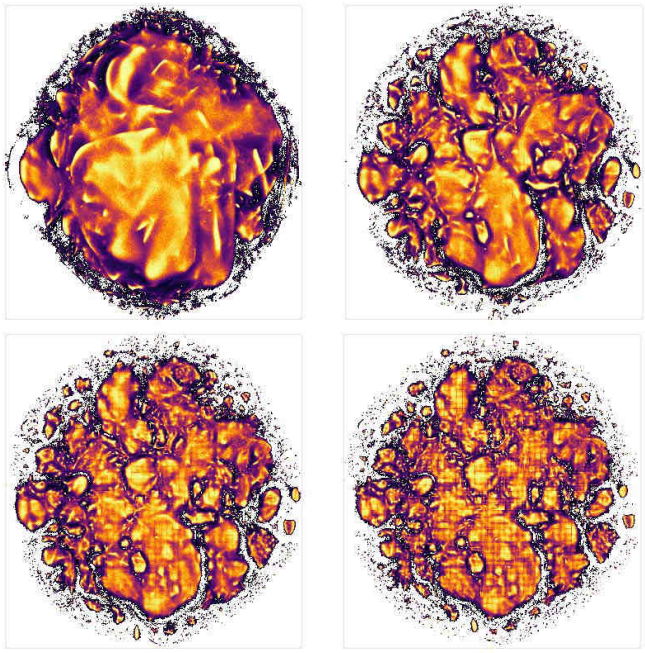

Fig. 8: The Deep10M Data set visualization for iterations 4000, 8000, 12000 and 16000

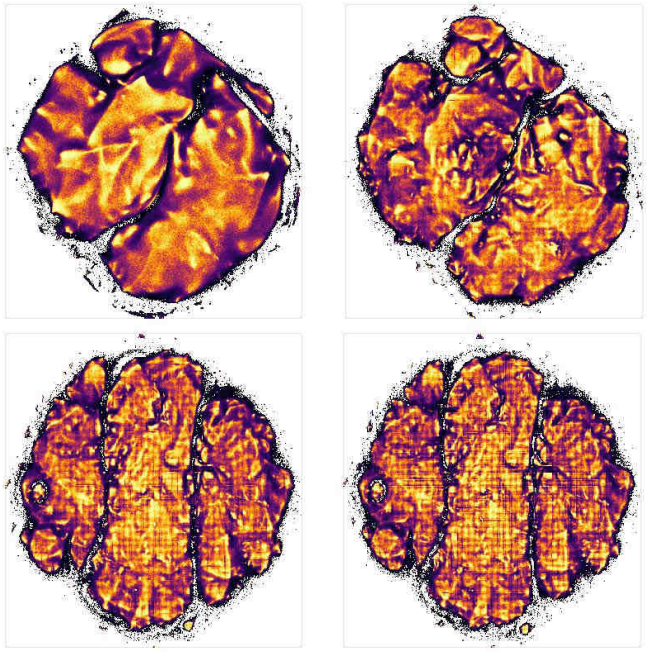

Fig. 9: The SIFT10M data set visualization for iterations 6000, 12000, 15000 and 18000

iterations 12000 and 16000. By looking at the average gradient norm in Figure 7, capturing this information about the stabilization of the algorithm seems difficult. The proposed metric however, better reflects these observations as it indicates stabilization at around 10000 iterations.

For the SIFT10M data set, we observe a slightly later stabilization than Deep10M despite the two data sets having

similar sizes and dimensions. The proposed histogram difference metric is able to capture the transitions in the visuals once again while the gradient norm seems to continue decreasing again, even after 20000 iterations. There is a clear transition between iterations 6000-12000 and 12000-15000 which is reflected as a constant decrease in the histogram difference norm plot in Figure 7. Again, we observe no significant changes between iterations 15000 and 18000, and the histogram difference metric indicates stabilization at around 15000 iterations. The bin-difference metric consistently drops to a certain value and shows little change after that point (note the log scale on the y-axis), while the average gradient norm metric steeply declines even well after the global structure of the embedding has stabilized. The same behaviour can be seen in Section III in the MNIST data set experiment which contains only 0.7% of the elements in SIFT10M and Deep10M data sets and has a much simpler structure.

It is also worth mentioning that the change in the gradient norm metric stays almost constant for a very long period of time, for both runs. This might render the application of patience-based early termination methods harder since the patience-interval would require to be very large to avoid local minima and premature termination. This behaviour is also observed in Section III with the MNIST data set, containing only 70000 elements.

D. Matrix Reordering

We analyze the effect of reordering the rows and columns of the $\mathbf{P}$ matrix on the data sets described above. We also evaluate the effect of the reordering for two different implementations: FI-tSNE and BH-tSNE. We already explained the FI-tSNE algorithm in detail in Section II. The BH-tSNE algorithm utilizes a Barnes-Hut tree to compute the repulsive forces and the compressed sparse row (CSR) based *SPMV* implementation in the cuSPARSE library to compute the attractive forces, instead of the COO based *SPMV* in FI-tSNE that relies on CUDA *atomicAdd* [1] operations. We believe this comparative evaluation could be helpful since some related work such as [26] in literature consider the Barnes-Hut version of the algorithm as a baseline.

Since the number of iterations required for the algorithm to converge differs from data set to data set, we present the speedups for a fixed number of iterations. From our experiments with the listed data sets, we discovered that most of them visually stabilize between 10 thousand and 20 thousand iterations. We run the algorithm for 20 thousand iterations to stretch-test and evaluate the overall speedup achieved by reordering the sparse matrix. Note that this is a significantly higher number of iterations than that is used in previous work, but it is necessary to obtain proper convergence due to the large size of the data sets as we demonstrated in the previous subsection.

In Figure 10 and Figure 11 we break down the end-to-end execution time into different parts. For FI-tSNE-cuda, these parts include the k nearest neighbors (*knn*), the attractive forces (*attr_f*), repulsive forces (*rep_f*), binning for histogram differences (*bin*) and matrix reordering (*reorder*) parts. All kernels which are responsible for an insignificant percentage are accumulated into "other". For BH-tSNE-cuda, we consider the kernels evaluated in [26]. These parts are mostly the same with FI-tSNE-cuda, while having the Barnes-Hut tree building (*build_tree*) part in addition.

First of all, comparing the solid bars in Figures 10 and 11, we can see that vanilla FI-tSNE-cuda clearly outperforms the vanilla BH-tSNE-cuda, even without any matrix reordering. For both versions, we observe that rabbit-reordering (hatched bars) delivers important end-to-end speedups. The percentage of the attractive forces computation time constitutes a significant portion of the vanilla BH-tSNE-cuda time, more than 75% of the total time for all runs. This results in better overall speedups in all experiments for the BH-tSNE case. Moreover, "reorder" time is an insignificant fraction of the overall execution time for both evaluations, and the improved memory access pattern amortizes its overhead even in the worst-case scenario, i.e., for the GloVe data set.

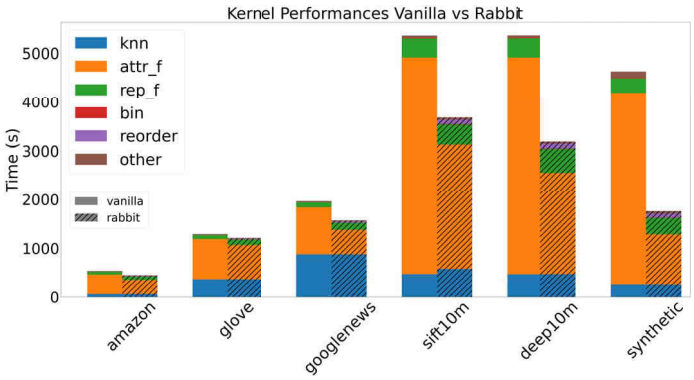

Fig. 10: Performance breakdown of the kernels for vanilla (solid bars) and rabbit re-ordered (hatched bars) runs for FI-tSNE.

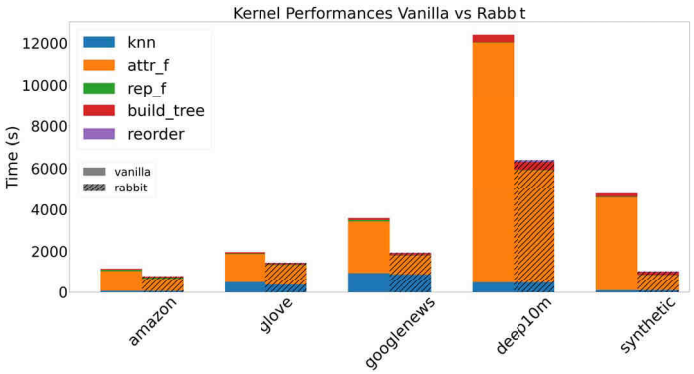

Fig. 11: Performance breakdown of the kernels for vanilla (solid bars) and rabbit re-ordered (hatched bars) runs for BH-tSNE.

The above table summarizes the impact of improving data locality through matrix reordering both for BH-tSNE and FI-

Table II: Speedup values after matrix reordering is applied to FI-tSNE and BH-tSNE, both for attractive forces computation and overall end-to-end running time.

	Amazon	*GloVe*	*GN*	*SI0M*	*D10M*	*Synth*
attr_f_FI	1.40×	1.17×	1.94×	1.74×	2.13×	3.82×
end-to-end_FI	1.20×	1.06×	1.25×	1.50×	1.68×	2.63×
attr_f_BH	1.65×	1.42×	2.70×	-	2.12×	5.97×
end-to-end_BH	1.46×	1.35×	1.85×	-	1.94×	4.63×

tSNE-cuda. We note that we sample 5 million points from the synthetic data set and do not include SIFT10M results for BH-tSNE-cuda due to high requirement of resources and long computation time. For both implementations, among all kernels, attractive forces kernel (*attr_f*) is most affected by the reordering as computations with the $\mathbf{P}$ matrix constitute the majority of the operations there. For this kernel, matrix reordering gives 1.17× to 3.82× speedup for the FI-tSNE-cuda case and 1.42× to 5.97× speedup for the BH-tSNE-cuda case, typically yielding higher speedups for larger data sets. Since *attr_f* is the most expensive part, these speed-ups translate to good end-to-end execution time improvements, too, especially in the BH-tSNE-cuda case. The proposed locality optimization results in 1.20x to 2.63x overall speedup for FI-tSNE-cuda while giving 1.46× to 4.63× improvement for the BH-tSNE-cuda case.

It is also worth mentioning that the GPU device used in this experiment could have a significant effect on the speedup. Since V100 has significantly more memory bandwidth than its counterparts and supports very fast atomic operations, the effects of the poor data-locality are diminished in comparison to most other NVIDIA GPU devices. Therefore, we expect that these improvements will be magnified on GPU devices having less memory bandwidth.

V. Related Work

The t-SNE algorithm has been extensively studied since its first appearance. The size of the target data set, however, has many implications for the state-of-the-art implementations. Here we discuss the current state of the GPU implementation and the stopping criterion, both of which are key factors for utilizing t-SNE for large data sets.

As we discussed and demonstrated, the number of iterations is one of the key hyper-parameters for t-SNE. However, to the best of our knowledge, this aspect of t-SNE has not been studied in detail before. In the BH-tSNE-cuda [1] and FI-tSNE-cuda [6] papers, the authors do not provide a convergence analysis for the experiments they perform. However, they present performance scalability tests where the number of iterations was fixed to 1000 by default in their implementation. While 1000 iterations can be more than enough for small data sets like MNIST, it is very likely that it will cause a premature termination in large data sets, far before the visual stabilization occurs. In this regard, our proposed histogram difference metric is a novel contribution, as we tried to demonstrate its merit by examining the visual stabilization of the low-dimensional embeddings along with plots of histogram difference and gradient norm metrics over iterations.

To improve the memory access pattern of BH-tSNE-cuda, Meyer et al. replace the Barnes-Hut implementation, which is based on [27], with an alternate version using $ImplicitTree$ representation [28]. This representation improves the memory access pattern, and hence the overall performance of the application [26]. While the authors are able to achieve around 2× end-to-end speed-up for the Amazon Electronics data set, the method is not applicable for FI-tSNE-cuda, which is the state-of-the-art t-SNE implementation. Since the performance of the FI-tSNE-cuda is significantly better (more than 2×) than BH-tSNE-cuda overall, the applicability of the optimization becomes critically important. As we demonstrated in Section IV, if we compare vanilla BH-tSNE-cuda (the baseline in [26]) to matrix reordered FI-tSNE-cuda, we observe combined end-to-end speedups ranging from 2× to more than 4×, for real-world data sets.

VI. Conclusion

In recent years, GPU implementations of the t-SNE algorithm sped up the execution time by several orders of magnitude, enabling the visualization of data sets containing millions of data points. We investigated and optimized the t-sne-cuda code for very large data sets containing up to 10 million points. The size of such data sets makes the t-SNE computation very expensive, even on GPUs. There are two main factors affecting the overall computation cost: Number of iterations and the sparse memory access pattern of the $\mathbf{P}$ matrix.

Currently, the only available stopping criterion is the gradient norm. While this metric can be used to terminate the algorithm, we discover that it is prone to the local changes occurring very late in the execution, making the metric not practical. We proposed a new stopping criterion using 2D binning difference norm on the low-dimensional embeddings produced by t-SNE. We implemented the proposed metric on GPU using CUDA, achieving negligible overhead in the t-SNE loop. We finally compared the proposed metric to the gradient norm and discussed its effectiveness for visual stabilization.

We also addressed the inefficient memory access pattern issue in the attractive forces kernel, both for BH-tSNE-cuda and FI-tSNE-cuda implementations. Since many real-world data sets have community structures, we integrated the multi-core Rabbit Order matrix-reordering algorithm to FI-tSNE-cuda and BH-tSNE-cuda codebase to improve the data locality of the $\mathbf{P}$ matrix. The optimizations resulted in up to 1.68× and 1.94× end-to-end speedup for FI-tSNE-cuda and BH-tSNE-cuda, respectively, for very large real-world data sets, while amortizing the reordering overheads for all runs.

The proposed methods, both for stopping the algorithm and improving the memory access pattern, could serve as a generic framework to analyze very large real-world data sets more efficiently. As future work, we plan to investigate methods to reduce memory utilization of the program and possibly integrate multi-GPU solutions to support even larger data sets.

REFERENCES

[1] D. M. Chan, R. Rao, F. Huang, and J. F. Canny, "t-sne-cuda: Gpu-accelerated t-sne and its applications to modern data," 2018.

[2] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [Online]. Available: http://www.jmlr.org/papers/v9/vandermaaten08a.html

[3] L. van der Maaten, "Accelerating t-sne using tree-based algorithms," *Journal of Machine Learning Research*, vol. 15, no. 93, pp. 3221–3245, 2014. [Online]. Available: http://jmlr.org/papers/v15/vandermaaten14a.html

[4] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[5] G. C. Linderman, M. Rachh, J. G. Hoskins, S. Steinerberger, and Y. Kluger, "Fast interpolation-based t-sne for improved visualization of single-cell rna-seq data," *Nature Methods*, vol. 16, no. 3, p. 243–245, Feb 2019. [Online]. Available: http://dx.doi.org/10.1038/s41592-018-0308-4

[6] D. M. Chan, R. Rao, F. Huang, and J. F. Canny, "Gpu accelerated t-distributed stochastic neighbor embedding," *Journal of Parallel and Distributed Computing*, vol. 131, pp. 1 – 13, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S074373151830875X

[7] L. V. D. Maaten, "Fast optimization for t-sne."

[8] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," 2017.

[9] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *In EMNLP*, 2014.

[10] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th National Conference*, ser. ACM '69. New York, NY, USA: Association for Computing Machinery, 1969, p. 157–172. [Online]. Available: https://doi.org/10.1145/800195.805928

[11] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM J. Matrix Anal. Appl.*, vol. 17, no. 4, p. 886–905, Oct. 1996. [Online]. Available: https://doi.org/10.1137/S0895479894278952

[12] A. Azad, M. Jacquelin, A. Buluc, and E. G. Ng, "The reverse cuthill-mckee algorithm in distributed-memory," 1 2016. [Online]. Available: https://www.osti.gov/biblio/1439186

[13] K. I. Karantasis, A. Lenharth, D. Nguyen, M. J. Garzarán, and K. Pingali, "Parallelization of reordering algorithms for bandwidth and wavefront reduction," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. IEEE Press, 2014, p. 921–932. [Online]. Available: https://doi.org/10.1109/SC.2014.80

[14] J. Arai, H. Shiokawa, T. Yamamuro, M. Onizuka, and S. Iwamura, "Rabbit order: Just-in-time parallel reordering for fast graph analysis." in *IPDPS*. IEEE Computer Society, 2016, pp. 22–31. [Online]. Available: http://dblp.uni-trier.de/db/conf/ipps/ipdps2016.html#AraiSYOI16

[15] C. Fu, Y. Zhang, D. Cai, and X. Ren, "Atsne: Efficient and robust visualization on gpu through hierarchical optimization," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 176–186. [Online]. Available: https://doi.org/10.1145/3292500.3330834

[16] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Googlenews-vectors-negative300.bin.gz - efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013. [Online]. Available: https://code.google.com/archive/p/word2vec/

[17] A. B. Yandex and V. Lempitsky, "Efficient indexing of billion-scale datasets of deep descriptors," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2055–2063.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[19] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: re-rank with source coding," 2011.

[20] J. A. Cottam, A. Lumsdaine, and P. Wang, "Abstract rendering: out-of-core rendering for information visualization," in *Visualization and Data Analysis 2014*, P. C. Wong, D. L. Kao, M. C. Hao, and C. Chen, Eds., vol. 9017, International Society for Optics and Photonics. SPIE, 2014, pp. 218 – 230. [Online]. Available: https://doi.org/10.1117/12.2041200

[21] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[22] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, April 2017, pp. 427–431.

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015. [Online]. Available: http://arxiv.org/abs/1409.4842

[25] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94

[26] B. H. Meyer, A. T. R. Pozo, and W. M. N. Zola, "Improving barnes-hut t-sne scalability in gpu with efficient memory access strategies," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[27] M. Burtscher and K. Pingali, "Chapter an efficient cuda implementation of the tree-based barnes hut n-body algorithm."

[28] W. M. Nunan Zola, L. C. E. Bona, and F. Silva, "Fast gpu parallel n-body tree traversal with simulated wide-warp," in *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2014, pp. 718–725.