Optimization and Comparison of Coordinate- and Metric-Based Indexes on GPUs for Distance Similarity Searches*

Michael Gowanlock[1 [0000−0002−0826−6204]], Benoit Gallet[1 [0000−0001−9716−1502]], and Brian Donnelly[1 [0000−0002−7484−3778]]

Northern Arizona University, Flagstaff, AZ, USA
{michael.gowanlock,benoit.gallet,brian.donnelly}@nau.edu

Abstract. The distance similarity search (DSS) is a fundamental operation for large-scale data analytics, as it is used to find all points that are within a search distance of a query point. Given that new scientific instruments are generating a tremendous amount of data, it is critical that these searches are highly efficient. Recently, GPU algorithms have been proposed to parallelize the DSS. While most work shows that GPU algorithms largely outperform parallel CPU algorithms, there is no single GPU algorithm that outperforms all other state-of-the-art approaches; therefore, it is not clear which algorithm should be selected based on a dataset/workload. We compare two GPU DSS algorithms: one that indexes directly on the data coordinates, and one that indexes using the distances between data points to a set of reference points. A counterintuitive finding is that the data dimensionality is not a good indicator of which algorithm should be used on a given dataset. We also find that the intrinsic dimensionality (ID) which quantifies structure in the data can be used to parameter tune the algorithms to improve performance over the baselines reported in prior work. Lastly, we find that combining the data dimensionality and ID can be used to select between the best performing GPU algorithm on a dataset.

Keywords: Distance Similarity Search · GPGPU · Metric-based Index

1 Introduction

The distance similarity search (DSS) finds objects within a search distance of points in a dataset. The distance similarity self-join (DSSJ) refers to finding all objects in a dataset within a distance, ϵ, of each other. DSS is a building block of several algorithms, including those used for scientific data analysis [12].

The *search-and-refine* strategy reduces the computational cost of the DSS, where an index prunes the search and generates a set of candidate points, which are then refined using distance calculations to compute the final result set for

* This material is based upon work supported by the National Science Foundation under Grant No. 2042155.

each query point. Graphics Processing Units (GPUs) have high computational throughput due to their massive parallelism and are very effective at performing distance calculations. Numerous research on the DSS has demonstrated that the GPU is superior to multi-core CPUs [11,9,3,6,7].

The performance of GPU DSS algorithms is largely a function of data-dependent properties, such as the data distribution, dimensionality, sparsity, and variance. Thus, there is not a single GPU algorithm that is better than all other algorithms, which makes it challenging to select an algorithm to employ on a given workload. The two major classes of indexes are those that employ coordinate- and metric-based indexes [10]. The former constructs an index directly on the coordinates of the data points, whereas the latter uses distances to a set of reference points instead of indexing on the data coordinates.

This paper compares two state-of-the-art GPU algorithms having a coordinate-based index (GDS-Join) and a metric-based index (COSS). Because neither algorithm performs best on all datasets, this paper aims to address the following questions: (*i*) What data properties can be used to determine the number of indexed dimensions (GDS-Join) or number of reference points (COSS) to use when processing a given dataset? (*ii*) Which dataset properties can be used to select whether GDS-Join or COSS should be employed on a dataset?

2 Background: Comparison of GDS-Join and COSS

In this section, we compare GDS-Join [9] to COSS [3]. For more information, we refer the reader to those papers. GDS-Join uses a coordinate-based index; in contrast, COSS is a metric-based index which stores points in a grid based on their distance to a set of reference points. GDS-Join and COSS are similar as their grid-based indexes are GPU-friendly as they address the drawbacks of the GPU's Single Instruction Multiple Thread execution model. Both GDS-Join and COSS use batching schemes to compute batches of query points such that GPU global memory is not exceeded and to hide PCIe data transfer latency.

GDS-Join and COSS have a parameter that controls the amount of pruning they perform, such that a trade-off can be reached between index search overhead and the number of distance calculations that are computed. In this paper, this is referred to as k, which is the number of indexed dimensions for GDS-Join or the number of reference points for COSS. Thus, the two algorithms have similar GPU kernel designs except that they use coordinate- and metric-based indexing. As we will show in the evaluation, this distinction yields respective strengths and weaknesses which are a function of data-dependent properties.

GDS-Join uses two additional optimizations than the preliminary work [9]. We utilize the method by Gowanlock [8] that orders the data points from most work to least work which reduces load imbalance by assigning query points with similar amounts of work to a given warp. This is referred to as reorder-queries. Instruction-level parallelism (ILP) is also used to hide memory access latency [13]. The DSSJ in high dimensionality performs many distance calculations in the filtering phase of the algorithm. Since the pairwise components

of the distance calculation can be computed independently, we exploit ILP by partially computing parts of the distance calculation and storing these partial results in registers. We use ILP, where we define r cached elements that are used to independently compute pair-wise distance calculations. Using these optimizations, GDS-JOIN achieves speedups over the preliminary work [9] in the range 1.82–5.51× across all datasets in Section 3.

3 Experimental Evaluation

3.1 Experimental Methodology

All GPU code is written in CUDA. The C/C++ host code is compiled with the GNU compiler and the O3 optimization flag. Our platform has 2×AMD EPYC 7542 2.9 GHz CPUs (64 total cores), 512 GiB of main memory, equipped with an Nvidia A100 GPU with 40 GiB of global memory, using CUDA 11 software.

In all experiments we exclude the time to load the dataset from disk. For all GPU algorithms, we include all other time components, including constructing the index, executing the DSSJ, storing the result set on the host and other host-side operations, and perform all pre-processing optimizations. Thus, we make a fair comparison between approaches. Reported time measurements are averaged over 3 time trials, and data is stored/processed using 64-bit floating point values. Throughout the evaluation, we report the speedup of GDS-JOIN over COSS (or vice versa), defined as the ratio of two response times, $s = T_{\text{COSS}}/T_{\text{GDS-JOIN}}$.

Selectivity of the experiments: We perform experiments across datasets and ϵ values such that we do not have too few or too many total results. Thus, the values of ϵ should represent values that are pragmatically useful. We define the selectivity of the self-join as $S_D = (|R| - |D|)/|D|$, where $|R|$ is the total result set size. This yields the average number of points within ϵ, excluding a point, p_a, finding itself. We select values of $S_D \sim 0 - 1000$ across all datasets, which is a typical range used in this literature.

Datasets: We use seven real-world datasets that span $n = 18 - 384$ dimensions (Table 1), allowing us to observe how performance varies as a function of dimensionality. We normalize all datasets in the range [0, 1]. All datasets except *BigCross* [1] and *Tiny5M*[1] were obtained from the UCI ML repository[2].

Implementation Configurations: All implementations are exact (not approximate) algorithms and are parallelized using the GPU using 64-bit floating point values. **GDS-JOIN:** GPU algorithm that uses a coordinate-based index. It is configured using 32 threads per block, as we found it to achieve the best performance on our platform. The default configuration for GDS-JOIN is to use all optimizations (SHORTC, REORDERDIMS, REORDERQUERIES, and ILP), index in $k = 6$ dimensions, and set $r = 8$ as the parameter for the ILP optimization. The source code is publicly available.[3] **COSS:** GPU algorithm that employs a

[1] https://www.cse.cuhk.edu.hk/systems/hash/gqr/dataset/tiny5m.tar.gz
[2] https://archive.ics.uci.edu/ml/index.php
[3] https://github.com/mgowanlock/gpu_self_join/

Dataset	$\|D\|$	n	$[\epsilon_{min}, \epsilon_{max}]$	$[S_D^{min}, S_D^{max}]$	Speedup GDS-Join over COSS
SuSy	5,000,000	18	[0.01, 0.021]	[5.17, 1090.45]	3.28
Higgs	11,000,000	28	[0.01, 0.0555]	[0.05, 1009.02]	2.15
WEC	287,999	49	[0.002, 0.007]	[39.46, 1006.39]	0.69
BigCross	11,620,300	57	[0.001, 0.02]	[2.54, 1044.7]	1.69
Census	2,458,285	68	[0.001, 0.01]	[21.64, 1077.6]	1.39
Songs	515,345	90	[0.007, 0.0091]	[126.91, 998.19]	0.70
Tiny5M	5,000,000	384	[0.2, 0.44]	[9.72, 1019.01]	0.34

Table 1. Datasets used in the evaluation, where $|D|$ is the dataset size, and n is the dimensionality. ϵ_{min} and ϵ_{max} refers to the range of search distances used, and S_D^{min} and S_D^{max} refer to their corresponding selectivity values. The algorithms are configured as described in Section 3.1. The mean speedup (or slowdown) of GDS-Join compared to COSS is shown.

metric-based index [3]. In the evaluation, COSS is configured to use 8 threads per point, and $k = 6$ reference points.

3.2 Results

Comparison of algorithms: Table 1 compares the performance of GDS-Join and COSS, where the speedup is computed using the average response times across five different values of ϵ as shown in the table. We observe that GDS-Join achieves the greatest speedup on 4 datasets (*SuSy*, *Higgs*, *BigCross*, and *Census*) whereas COSS achieves the greatest speedup on 3 datasets (*WEC*, *Songs*, and *Tiny5M*). The two GPU algorithms have their respective niches, as performance largely depends on data properties. Furthermore, one optimization that must be selected for GDS-Join and COSS is the number of indexed dimensions and the number of reference points, respectively. This leads to the following questions: *When should* GDS-Join *or* COSS *be employed, how many dimensions (*GDS-Join*) or reference points (*COSS*) should be used, and what properties can we use to infer the best algorithm to employ?*

Index dimensionality reduction: We examine index dimensionality reduction for GDS-Join and COSS. Figure 1 plots the normalized response time of all real-world datasets as a function of k for GDS-Join, where a lower time indicates better performance. We normalize to the largest response time (yielding a value of 1 in each plot), such that we can compare the datasets across the same scale. Key observations are as follows: (*i*) Panels (b) and (e) have a parabolic shape and clearly show the trade-off between index search overhead and the number of distance calculations performed. (*ii*) Panels (a), (c), (d), and (f) are similar to the above, except that the potential overhead of computing a large number of distance calculations at low values of k is absent. This is because the data is distributed into a sufficient number of grid cells such that a large number of distance calculations can be pruned even at low values of k. This illustrates

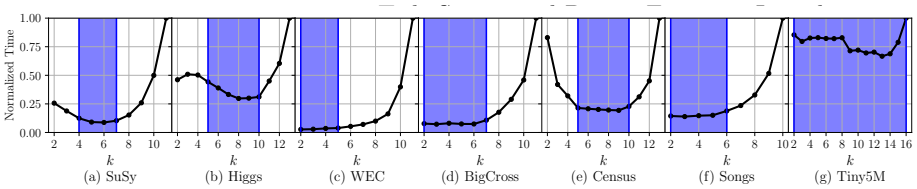

Fig. 1. The normalized response time as a function of k for all real-world datasets, where a lower normalized time is better. GDS-JOIN is executed using the median values of ϵ in Table 1; across (a)–(g) they are as follows: {0.0155, 0.03275, 0.0045, 0.0105, 0.0055, 0.00805, 0.32}. Shaded regions — the range of values of k that have a response time within 50% of the value of k that yielded the lowest response time.

how the REORDERDIMS optimization combined with indexing $k < n$ dimensions can exploit the *intrinsic dimensionality* of the data; e.g., (f) *Songs* has $n = 90$ dimensions, but yields good performance when indexed in only $k = 3$ dimensions. (*iii*) The shaded regions show values of k that yield a good response time, where k can be selected in a large range and obtain good performance on most datasets. With the exception of *WEC*, $k = 6$ yields respectable performance on all datasets. We carried out the same experiment for COSS where k refers to the number of reference points. The performance behavior between GDS-JOIN and COSS is similar so we omit discussing this.

In summary, indexing in $k < n$ dimensions (GDS-JOIN), or using $k < n$ reference points (COSS) provides a trade-off between distance comparisons and index search overhead. *Finding 1: There is not a direct correlation between data dimensionality and the number indexed dimensions (*GDS-JOIN*) or reference points (*COSS*) that should be used. Another metric for understanding data-dependent performance is needed.*

Using intrinsic dimensionality (ID) to select k: The ID is the number of dimensions required to approximately represent a dataset. Intuitively, some data dimensions are correlated, so it is possible to represent the data in fewer than n dimensions. We propose a heuristic for selecting k as described in Equation 1, where $i \geq 1$ and $n \geq k$.

$$k = 2 + \lceil c \cdot \log_2 i \rfloor. \tag{1}$$

In the equation, the base of the log is 2 — conceptually, increasing k to $k + 1$ reduces the number of distance calculations by a factor of two, which yields diminishing returns when increasing k. We take the log of the intrinsic dimensionality (i) because the efficiency of pruning is directly related to the structure of the data, with a low ID being easy to prune with a small k, while a high ID requires a larger value of k to achieve a similar degree of pruning. The notation $\lceil x \rfloor$ is the rounding function, and c is a coefficient where $c = \sqrt{\frac{|D|}{5 \times 10^6}}$. The coefficient is used to scale the number of indexed dimensions as a function of the dataset size $|D|$. This is needed because while the worst case time complexity for a sufficiently large ϵ value is $O(|D|^2)$, in practice, an average query only requires refining a fraction of the total dataset, $|D|$. Therefore, this

Dataset	n	i	k_{eqn} (Eqn. 1)	Ratio GDS-Join	Ratio COSS
SuSy	18	9	5	0.952	1.130
Higgs	28	19	8	1.301	0.882
WEC	49	15	3	1.865	1.932
BigCross	57	3	4	0.926	2.217
Census	68	13	5	1.074	1.481
Songs	90	29	4	1.290	1.426
Tiny5M	384	63	8	0.994	1.011

Table 2. The rounded intrinsic dimensionality (i) of each dataset, and k_{eqn} which is the selected value of k using Equation 1. The data dimensionality (n) is shown for comparison. The time ratio is $T_{k=6}/T_{eqn}$.

factor scales with dataset size to limit the number of indexed dimensions when a small dataset is employed and use more dimensions when processing larger datasets. Lastly, because indexing in few dimensions is inexpensive, we index in at least 2 dimensions.

We employ an ID estimator to compute i in Equation 1 that uses local Principle Component Analysis (PCA) [5,4], where it uses the k-nearest neighbor graph to estimate ID, and we employed the PCA algorithm from the scikit-dimension library[4]. To compute i, we used 100 nearest neighbors on all datasets[5].

Table 2 shows the estimated ID from the PCA method, and the value of k using Equation 1. We find that this heuristic yields a very good value of k (see Figure 1). All of the values of k yield an execution time for each dataset in the blue shaded region. Thus, ID is a good tool for determining the number of dimensions that should be indexed.

As described in Section 3.1, GDS-Join and COSS are configured by indexing in $k = 6$ dimensions and using $k = 6$ reference points, respectively, because those values were found to yield good performance across all datasets in the paper. However, $k = 6$ does not yield the best response time across all datasets. Table 2 shows the ratio of $T_{k=6}$ to T_{eqn}, which refers to the response time ratio when $k = 6$ compared to that given when using k from Equation 1. For GDS-Join, we find that there are three cases where T_{eqn} is slower ($\frac{T_{k=6}}{T_{eqn}} < 1$), but the performance loss is minor. In contrast, there are four cases where T_{eqn} yields a faster response time ($\frac{T_{k=6}}{T_{eqn}} > 1$) where substantial performance gains are achieved on *Higgs*, *WEC*, and *Songs*, yielding a ratio between 1.07–1.87×.

Examining the abovementioned ratio for COSS, we find that there is only one dataset (*Higgs*) where using k_{eqn} yields a slowdown; all other datasets yield a ratio between 1.01–2.22×. Using ID to estimate k reaches a good trade-off between index search overhead and the number of distance comparisons. It is for this reason that other work finds that using $k \approx 6$ reference points yields

[4] https://scikit-dimension.readthedocs.io/en/latest/

[5] Due to excessive execution times, we sampled *Higgs* and *Tiny5M* and ensured that 100 neighbors are sufficient across all datasets, and that sampling *Higgs* and *Tiny5M* did not adversely impact ID estimation.

Dataset	Best GPU Alg. (Table 1)	n	k_{eqn} (Eqn. 1)	n/k_{eqn}
SuSy	GDS-Join	18	5	3.60
Higgs	GDS-Join	28	8	3.50
WEC	COSS	49	3	16.33
Census	GDS-Join	68	5	13.60
Songs	COSS	90	4	22.50
Tiny5M	COSS	384	8	48.00

Table 3. The best GPU algorithm (GDS-Join or COSS) from Table 1 is shown, with the values of n and k_{eqn} from Table 2. The ratio of the n/k_{eqn} indicates whether GDS-Join or COSS should be employed. We excluded *BigCross* as neither GPU algorithm outperformed the CPU algorithms on that dataset.

good performance [2]. We also find that our heuristic yields similar values of k. *Finding 2: The ID is a much better indicator of the number of dimensions that should be indexed than the data dimensionality.*

When should GDS-Join or COSS be employed? Both algorithms have distinct niches, but it is not clear from the results thus far under what circumstances GDS-Join or COSS should be employed. By definition, metric-based indexes (COSS) are more effective than coordinate-based indexes (GDS-Join) when there is less structure in the data indicating lower ID, and so by indexing in the metric space, they are able to better prune the search in instances where coordinate-based indexes cannot.

Table 3 shows the algorithm that achieved the best performance in Table 1. Also reported is n/k_{eqn}, where k_{eqn} is a function of the ID. Intuitively, this indicates how much pruning the index can accomplish relative to all n dimensions. Because COSS outperforms GDS-Join when there is less structure in the data (lower ID), we find that when $n/k_{eqn} \geq 16$ COSS should be employed, and likewise when $n/k_{eqn} < 16$ GDS-Join should be employed, which indicates that there is more structure in the data, or a higher ID relative to the total number of data dimensions, n. Consequently, one of the GPU algorithms can be selected based on n and k_{eqn}, which is a function of data-dependent properties. *Finding 3: Datasets with greater ID and/or fewer data dimensions are best processed by coordinate-based indexes (*GDS-Join*), whereas datasets with lower ID and/or higher dimensions should be processed by metric-based indexes (*COSS*).*

4 Discussion & Conclusions

This paper examined fundamental data properties that can be used to: (*i*) improve the performance of GDS-Join and COSS; and, (*ii*) determine under which conditions GDS-Join or COSS should be employed. Algorithm selection allows for computing DSS more robustly, as we are now able to select an algorithm that performs well depending on the characteristics of a dataset. Using the

proposed heuristics, we find that GDS-Join or COSS can be selected largely based on the *intrinsic dimensionality* and the number of data dimensions.

Acknowledgements We thank Ben Karsin for his contributions to the preliminary version of this paper.

References

1. Ackermann, M.R., Märtens, M., Raupach, C., Swierkot, K., Lammersen, C., Sohler, C.: Streamkm++ a clustering algorithm for data streams. Journal of Experimental Algorithmics (JEA) **17**, 2–1 (2012)
2. Chen, L., Gao, Y., Li, X., Jensen, C.S., Chen, G.: Efficient metric indexing for similarity search. In: 2015 IEEE 31st Intl. Conf. on Data Engineering. pp. 591–602. IEEE (2015)
3. Donnelly, B., Gowanlock, M.: A Coordinate-Oblivious Index for High-Dimensional Distance Similarity Searches on the GPU. In: Proc. of the 34th ACM Intl. Conf. on Supercomputing. pp. 1–12 (2020)
4. Fan, M., Gu, N., Qiao, H., Zhang, B.: Intrinsic dimension estimation of data by principal component analysis. arXiv preprint arXiv:1002.2050 (2010)
5. Fukunaga, K., Olsen, D.R.: An algorithm for finding intrinsic dimensionality of data. IEEE Transactions on Computers **100**(2), 176–183 (1971)
6. Gallet, B., Gowanlock, M.: Heterogeneous CPU-GPU epsilon grid joins: static and dynamic work partitioning strategies. Data Science and Engineering **6**(1), 39–62 (2021)
7. Gallet, B., Gowanlock, M.: Leveraging GPU Tensor Cores for Double Precision Euclidean Distance Calculations. In: 2022 IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC). pp. 135–144 (2022)
8. Gowanlock, M.: Hybrid knn-join: Parallel nearest neighbor searches exploiting cpu and gpu architectural features. Journal of Parallel and Distributed Computing **149**, 119–137 (2021)
9. Gowanlock, M., Karsin, B.: GPU-Accelerated Similarity Self-Join for Multi-Dimensional Data. In: Proc. of the 15th Intl. Workshop on Data Management on New Hardware. pp. 6:1–6:9. ACM (2019)
10. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces (survey article). ACM Transactions on Database Systems **28**(4), 517–580 (2003)
11. Lieberman, M.D., Sankaranarayanan, J., Samet, H.: A fast similarity join algorithm using graphics processing units. In: IEEE 24th Intl. Conf. on Data Engineering. pp. 1111–1120 (2008)
12. Trilling, D.E., et al.: The solar system notification alert processing system (snaps): Design, architecture, and first data release (snapshot1). The Astronomical Journal **165**(3), 111 (2023)
13. Volkov, V.: Better performance at lower occupancy. https://www.nvidia.com/content/GTC-2010/pdfs/2238_GTC2010.pdf (2010), accessed Jan. 1, 2023