SING: Sequence Indexing Using GPUs

Botao Peng
LIPADE, Université de Paris
Institute of Computing Technology,
Chinese Academy of Sciences
botao.peng@parisdescartes.fr

Panagiota Fatourou
FORTH ICS
Dept. of Comp. Science, Univ. of Crete
faturu@csd.uoc.gr

Themis Palpanas
LIPADE, Université de Paris
French University Institute (IUF)
themis@mi.parisdescartes.fr

***Abstract*—Data series similarity search is a core operation for several data series analysis applications across many domains. This has attracted lots of interest that led to the development of several indexing techniques. Nevertheless, these techniques fail to deliver the similarity search time performance that is needed for interactive exploration, or analysis of large data series collections. We propose SING, the first data series index designed to take advantage of Graphics Processing Units (GPUs). SING is an in-memory index that uses CPU+GPU co-processing (as well as SIMD, multi-core and multi-socket architectures), in order to accelerate similarity search. Our experimental evaluation with synthetic and real datasets shows that SING is up to 5.1x faster than the state-of-the-art parallel in-memory approach, and up to 62x faster than the state-of-the-art parallel serial scan algorithm. SING achieves exact similarity search query times as low as 32msec on 100GB datasets, which enables interactive data exploration on very large data series collections.**

***Index Terms*—Data series, Indexing, Modern hardware, GPU**

I. Introduction

[Motivation] Several applications across many diverse domains, such as in finance, astrophysics, neuroscience, engineering, multimedia, and others [1]–[4], continuously produce big collections of data series[1], which need to be processed and analyzed [5]–[11].Often times, this is part of an exploratory process, where users ask a query, review the results, and then decide what their subsequent queries, or analysis steps should be [4]. The most common type of query that different analysis applications need to answer on these collections of data series is similarity search [1], [12]–[14]. The continued increase in the rate and volume of data series production, with collections that grow to several petabytes in size [1], [3], [4], renders traditional, serial-execution data series indexing technologies [15]–[21] inadequate [12], [22], [23]. For this reason, several recent efforts have focused on the development of parallel [24]–[26] and distributed [27]–[30] indexing techniques. In this work, we will not consider distributed solutions. However, the study of techniques that combine parallelization (in a single node) and distribution (across nodes) is an interesting research direction.

In this work, we focus on designing an efficient parallel indexing and exact query answering scheme for *in-memory* data series processing. The necessity for fast in-memory data series computation appears in real scenaria, e.g., in Airbus [26]. Although Airbus stores petabytes of data series, reasoning about the behavior of aircraft components or pilots [31] requires experts to run analytics on subsets of the data (e.g., relevant to landings from Air France pilots) that fit in memory.

[State-of-the-Art Solutions] ParIS+ [24], [25] is a recently-designed *disk-based* data series indexing scheme that takes advantage of modern hardware parallelization. However, its performance is dominated by the I/O cost it encounters for processing the disk-resident data. This cost is too high (e.g., 15sec for answering an 1-Nearest Neighbor exact query on a 100GB dataset) for keeping the user's attention (i.e., 10sec), let alone for supporting interactivity in the analysis process (i.e., 100msec) [32]. MESSI [26] is an *in-memory* parallel index, built upon the lessons learned from ParIS+, achieving for the first time interactive exact query answering times (at ∼50msec), being 6-11x faster than ParIS+.

The performance improvements of ParIS+ and MESSI were achieved by exploiting the parallelism opportunities offered by the multi-socket, multi-core, and SIMD architectures. However, these indices did not take advantage of the parallel computation power of Graphics Processing Units (GPUs). This is the research direction that we study here.

[Challenges] We note that the programming approach for GPU algorithms is distinct from that of CPU, and requires different design and techniques in order to achieve the desired performance improvement. The following constraints of a GPU determine the algorithmic choices we need to make. First, the on-board GPU memory is rather limited, and cannot hold the entire dataset. Note that the GPU we use in this work has 12GB of RAM, while our datasets are one order of magnitude larger, occupying 100GB; much larger data series collections are very common in practice [3], [4], [12], [13].

Second, the solution of moving at query time all, or subsets of the raw data into the GPU (e.g., in batches, or streaming) for further processing is also challenging because of the slow interconnect speeds: in our system (detailed in Section IV-A) that uses a PCI-Express 3.0 x16 bus, this speed was measured at 10GB/sec. Given that query answering times are in the order of 50-100msec, the above data transfer rate means that not only can we not afford to move the entire dataset to the GPU (that would need 10sec), but even moving small *ad hoc* subsets of data required by queries (i.e., those not pruned) incurs a prohibitively high time cost (e.g., the raw data for an average 0.4% of a 100GB dataset would need >40msec).

[1]A data series, or data sequence, is an ordered sequence of data points. If the ordering dimension is time then we talk about time series, though, series can be ordered over other measures. (e.g., angle in astronomical radial profiles, frequency in infrared spectroscopy, mass in mass spectroscopy, etc.).

The above considerations imply that existing tree-based approaches (such as the state-of-the-art MESSI [26]), or simple GPU adaptations of those, cannot outperform modern *multi-core* parallel solutions. In fact, we point out that previous GPU solutions for query answering and similarity search only compared to CPU baselines with up to 2 cores, and without the use of SIMD [33]–[35].

[Our Approach and Contributions] In this work, we describe SING (Sequence Indexing Using GPUs), the first (in-memory[2]) data series index that uses the GPU's parallelization opportunities, as well as the SIMD, multi-core and multi-socket architectures of modern hardware, in order to accelerate exact similarity search.

SING provides a novel similarity search algorithm that runs on top of the MESSI tree index [26]. This algorithm ensures effective CPU+GPU co-processing (i.e., collaboration of both the CPU and GPU resources of the system) to produce the exact query answers. Overall, SING reduces considerably not only the amount of work that needs to be performed, but also the execution time required to complete the necessary work. Our experimental evaluation shows that SING outperforms by a large margin the current state-of-the-art parallel (i.e., SIMD and multi-core) solutions in a variety of settings, and continues to do so even when we use all 16 cores of our system.

II. Preliminaries

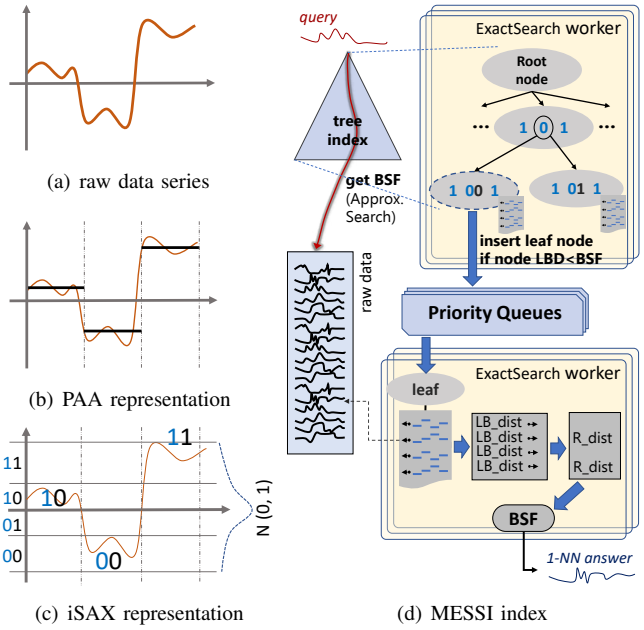
[Data Series] A data series, $S = \{p_1, ..., p_n\}$, is defined as a sequence of points, where each point $p_i = (v_i, t_i)$, $1 \leq i \leq n$, is associated to a real value v_i and a position t_i. The position corresponds to the order of this value in the sequence. We call n the *size*, or *length* of the data series. We note that all the discussions in this paper are applicable to high-dimensional vectors, in general.

[Similarity Search] Analysts perform a wide range of data mining tasks on data series including clustering [36], classification and deviation detection [37], [38], and frequent pattern mining [39]. Existing algorithms for executing these tasks rely on performing fast similarity search across the different series. Thus, efficiently processing nearest neighbor (NN) queries is crucial for speeding up the above tasks.

NN queries are formally defined as follows: given a query series S_q of length n, and a data series collection $\mathcal{S}$ of sequences of the same length, n, we want to identify the series $S_c \in \mathcal{S}$ that has the smallest distance to S_q among all the series in the collection $\mathcal{S}$. (In the case of streaming series, we first create subsequences of length n using a sliding window, and then index those.)

In this work, we use Euclidean Distance (ED) [40]; though, Dynamic Time Warping (DTW) [41] can be easily supported, as well, with no changes to the index structure [26].

[iSAX] The iSAX representation (or summary) is based on the Piecewise Aggregate Approximation (PAA) representation [42], which divides the data series in segments of equal length, and uses the mean value of the points in each segment in order to summarize a data series. Figure 1(b) depicts an example of PAA representation with $w = 3$ segments (depicted with the black horizontal lines), for the data series depicted in Figure 1(a).



(a) raw data series (b) PAA representation (c) iSAX representation (d) MESSI index

Fig. 1. The iSAX representation, and the MESSI index

Based on PAA, the indexable Symbolic Aggregate approXimation (iSAX) representation was proposed [43] (and later used in several different data series indices [21], [24], [37], [44], [45]). This method first divides the (y-axis) space in different regions, and assigns a bit-wise symbol to each region. In practice, the number of symbols is small: iSAX achieves very good approximations with as few as 256 symbols, the maximum alphabet cardinality, $|alphabet|$, which can be represented by eight bits [19]. It then represents each one of the w segments of the series with the symbol of the region the PAA falls into, forming the word $10_2 00_2 11_2$ shown in Figure 1(c) (subscripts denote the number of bits used to represent the symbol of each segment).

[MESSI Index] Based on the iSAX summary [43], the MESSI index was developed [46], which proposed techniques and algorithms specifically designed for modern hardware and in-memory data [26]. MESSI makes use of variable cardinalities for the iSAX summaries (i.e., variable degrees of precision for the symbol of each segment) in order to build a hierarchical tree index (see Figure 1(d)), consisting of three types of nodes: (i) the root node points to several children nodes, 2^w in the worst case: when the series in the collection cover all possible iSAX summaries (following previous work [46], we use $w = 16$); (ii) each inner node contains the iSAX summary of all the series below it, and has two children; and (iii) each leaf node contains the iSAX summaries of all the series inside it, and pointers to the raw data (in order to be able to prune false positives and produce exact, correct answers),

[2]For disk-resident data, previous work has shown that the computations are disk I/O bound [25].

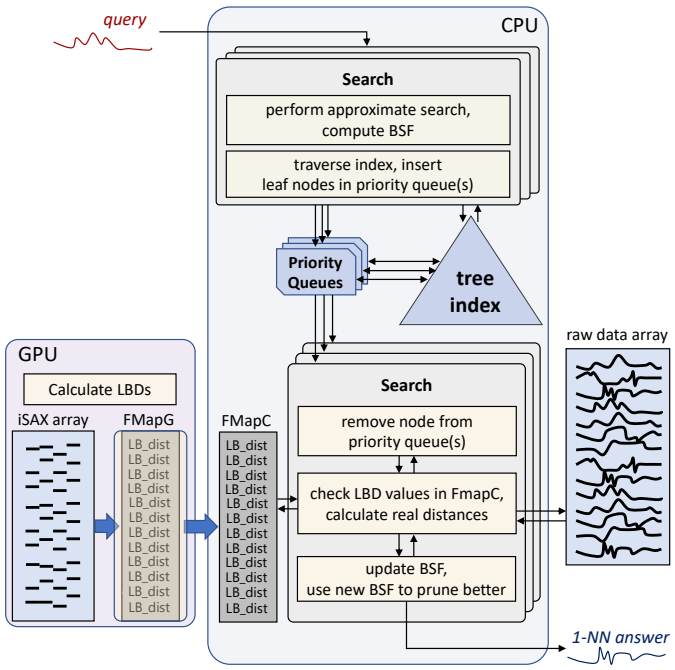


Fig. 2. M+G flowchart for query answering.

which reside on disk. When the number of series in a leaf node becomes greater than the maximum leaf capacity, the leaf splits: it becomes an inner node and creates two new leaves, by increasing the cardinality of the iSAX summary of one of the segments (the one that will result in the most balanced split of the contents of the node to its two new children [19], [21]). The two refined iSAX summaries (new bit set to *0* and *1*) are assigned to the two new leaves. In our example, the series of Figure 1(c) will be placed in the outlined leaf node of the index (Figure 1(d)). We define the distance of a query series to a node as the distance between the query (raw values, or iSAX summary) and the iSAX summary of the node.

III. The SING Data Series Index

SING adjusts the tree index that is created by MESSI. It also uses a iSAX array which is produced as the iSAX array in ParIS+. SING then transfers the iSAX array (or an appropriate part of it) in the GPU memory, so that it is already in the GPU at query answering time. We first present our basic solution, called M+G, and then we present SING.

A. The M+G Solution

To answer a query in M+G, a CPU thread performs an approximate search in order to get a first estimate of the answer (refer to Figure 2). This estimate is stored in a variable, called Best-So-Far (BSF). Then, the PAA of the query and BSF are transferred in the GPU and the GPU threads are instructed to calculate the distance between the iSAX summary of each entry of SAX and the PAA of the query. The GPU outputs a float map (i.e., an array of float values) containing the lower bound distance for those iSAX summaries stored in the iSAX array. This float map is output to the CPU threads using streaming. Specifically, it is split into chunks and as soon as the data in a chunk becomes ready, the chunk is output to the CPU threads. The element of each row of the float map corresponds to the same-numbered row of the raw data array. This computation comprises the *lower bound distance calculation* phase and it is executed entirely in GPU.

At the same time (i.e. concurrently to the lower bound distance calculation phase), several CPU threads traverse the tree and create a number of priority queues the same way it is done in MESSI. Then, the CPU threads wait until the lower bound distance calculation has finished. Afterwards, the CPU threads start processing the priority queues. Specifically, each thread chooses a priority queue to work on and repeatedly deletes the node with the highest priority from it. If the node cannot be pruned, then for each element of the node, the thread checks the lower bound distance stored in the float map for this element. In this way, the CPU thread does not have to calculate the lower bound distance by itself as is the case in MESSI. If the lower bound distance is larger than the current value of BSF, the data series is pruned. Otherwise, the real distance computation is performed. If any of these real distance computations results in a value smaller than the current value of BSF, then the BSF is updated to store the smaller value. This process continues until all nodes in the priority queues have either been processed or be pruned.

B. The SING Solution

SING is an optimized version of M+G. Below, we describe the different design decisions made in SING.

SING follows a different strategy for building the iSAX array than M+G. Specifically, as soon as the tree index is constructed, SING performs a recursive traversal of the tree, which visits the tree leaves from the leftmost to the rightmost (in this order)[3] and stores in the iSAX array, the iSAX summaries of all the data series stored in the leaves (in order). This allows SING to access consecutive elements of the float map computed by the GPU instead of performing random accesses in it. This reduces the number of cache misses caused during the priority queue processing phase that the float map is accessed in order to check whether the elements contained in deleted nodes of the priority queues can be pruned. This is so since the elements of a node are examined in order, and therefore, it is beneficial to have the lower bound distances for those elements stored in the float map in this order.

Based on previous experiments [26], a big number of root subtrees can be pruned. SING query answering approach exploits this idea to apply an initial pruning technique which determines which part of the iSAX array is actually necessary to be processed by the GPU. This is done as follows. Before instructing the GPU to calculate lower bound distances, the CPU threads compute the lower bound distances between the query PAA and each of the tree root children. In this way, they identify a collection of consecutive subtrees that cannot

[3]An inorder traversal of the root subtrees from the leftmost subtree to the rightmost subtree would accomplish this task.

be pruned. Note that the iSAX summaries of the data series stored in the leaves of each sequence of consecutive subtrees of this collection reside in one or more consecutive chunks of the iSAX array. The GPU is then instructed to calculate lower bound distances for each of these chunks. Therefore, the GPU threads calculate lower bounds only for the series in the subtrees that cannot be pruned. Consequently, the number of lower bound distance calculations that are performed by the GPU in SING is much smaller than in M+G, and thus, the GPU execution time is also reduced.

Experiments show that, even after the optimization described above, the GPU computation time is often higher than the time the CPU threads require to create the priority queues. Thus, in M+G, the search workers (CPU threads) wait for the GPU to complete its computation which is wasteful in terms of computational resources. To overcome this problem, in SING, the search workers start processing the elements in the priority queues they create without waiting for the GPU computation to complete. As soon as a search worker discovers that the priority creation phase has been completed by all CPU threads, it proceeds immediately to the priority queue processing phase. It chooses a priority queue to work on and calls *DeleteMin()* to process the root element, i.e. the highest priority element in the queue. It first checks if the GPU computation has already performed the lower bound distance calculations for the elements of this node. If not, the search worker calculates these lower bounds itself, and then moves on to the next node to work on. If yes, the search worker uses the lower bound distances calculated by the GPU to prune if possible, decides whether further examination of the elements of the node is necessary (by calculating real distances), and moves on to the next node to work on.

To achieve this overlap of the queue processing phase with the GPU computation, the float map used by the GPU is split into chunks, and the GPU outputs each chunk to the CPU threads as soon as the lower bound distance calculations of the elements stored in the chunk have been performed. We denote by N_l the total number of chunks that the iSAX array (and the corresponding GPU float map) comprise.

IV. Experimental Evaluation

A. Setup

We used a server with 2x Intel Xeon Gold 6134 CPUs with 8 cores each, 320GB RAM and a Titan Xp GPU with 3840 NVIDIA CUDA Cores (12GB RAM). All algorithms were implemented in C and C++, and compiled using GCC v7.4.0 and NVCC 10.1 on Ubuntu Linux v18.04. We use NVCC to compile the GPU part of the code as a lib file; we then link this lib to the main function of the program.

[Algorithms] We compared SING to the following algorithms:
(i) MESSI [26], the state-of-the-art modern hardware data series index.
(ii) UCR Suite-P, our parallel implementation of the state-of-the-art optimized serial scan technique, UCR Suite [41]. In UCR Suite-P, every thread is assigned a part of the in-memory data series array, and all threads concurrently and independently process their own parts, performing the real distance calculations in SIMD, and only synchronize at the end to produce the final result. (We do not consider the non-parallel UCR Suite in our study, since it is $\sim$300x slower.)
(iii) UCR Suite GPU [47], an extension of UCR Suite, where all computations take place in the GPU.
(iv) Finally, we compare to M+G, our baseline solution based on MESSI, described in Section III.

SING and M+G use both the CPU and GPU, while MESSI and UCR Suite-P can only use the CPU. All algorithms operated exclusively in memory: the index tree and raw data were already loaded in the main memory of the system, and the SAX array was already loaded in the GPU memory (for SING and M+G). The code for all algorithms is online [48].

[Datasets] In order to evaluate the performance of the proposed approach, we use several synthetic datasets (produced by a random walk data series generator [12], [13], [49]) for a fine grained analysis, and two real datasets from diverse domains. Unless otherwise noted, the series have a size of 256 points, which is a standard length used in the literature, and allows us to compare our results to previous work. All our datasets are Z-normalized[4]. For our first real dataset, *Seismic*, we used the IRIS Seismic Data Access repository [52] to gather 100M series representing seismic waves from various locations, for a total size of 100GB. The second real dataset, *SALD*, includes neuroscience MRI data series [53], for a total of 200M series of size 128, of size 100 GB. Note that in all cases, the raw data and the index are stored in the CPU memory, while the iSAX representations for all series in the dataset are stored in the GPU memory[5].

In both cases, we used as queries 100 series that were not part of the datasets (produced using our synthetic series generator, since these datasets do not come with query workloads). In all cases, we repeated the experiments 10 times and we report the average values. We omit reporting the error bars, since all runs gave results that were very similar (less than 3% difference). Queries were always run in a sequential fashion, one after the other, in order to simulate an exploratory analysis scenario, where users formulate new queries after having seen the results of the previous one.

We note that in all cases, the answers produced by our algorithms are the exact, correct answers; the same is true for the competitors we compare against.

B. Results

Figures 3 and 4 report the execution times of MESSI, M+G and SING, as we vary the number of cores up to 8

[4]Z-normalization transforms a series so that it has a mean value of zero, and a standard deviation of one. This allows similarity search to be effective, irrespective of shifting (i.e., offset translation) and scaling [50]. Therefore, similarity search can return results with similar trends, but different absolute values. Moreover, minimizing the Euclidean distance on Z-normalized data is equivalent to maximizing their Pearson's correlation coefficient [51]. For these reasons, Z-normalization is extensively used in both the literature [12], [13], [49] and in practice [3], [4].

[5]For a 100GB dataset, the iSAX summaries occupy less than 2GB of GPU memory. This means that our 12GB GPU memory could support query answering using SING for datasets as large as 600GB.

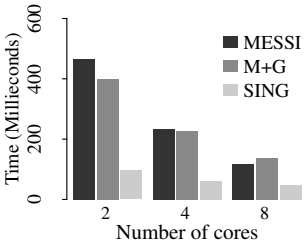


Fig. 3. Query answering time, varying cores (1 socket; 100GB Synthetic).

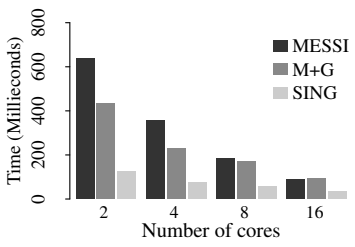


Fig. 4. Query answering time, varying cores (2 sockets; 100GB Synthetic).

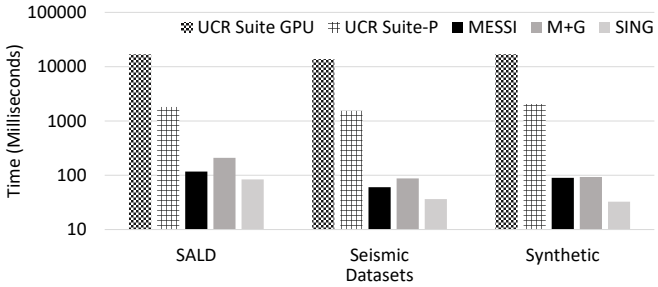


Fig. 5. Query answering time, varying datasets (16 cores, 2 sockets; 100GB).

in one socket, and up to 16 in two sockets. We observe that the performance of all algorithms improves when we use more cores. This improvement is more pronounced for MESSI, which starts from much higher execution times for small numbers of cores. M+G can only beat the performance of MESSI when using a small number of cores (in either 1, or 2 sockets). In these cases, having the GPU calculate the lower bounds removes a heavy burden from the CPU, and translates to execution time savings. On the other hand, SING consistently outperforms the competitors across the board. SING is 5.1x faster than MESSI for 2 cores in 2 sockets. Even when we use all 16 cores of our system, SING is still 2.8x faster than MESSI. SING only needs 32ms to answer an exact similarity search query on a 100GB dataset.

Finally, we report the results of the comparison to the state-of-the-art parallel serial scan algorithm, UCR Suite-P[6]. Figure 5 (log-scale y-axis) reports the query answering time for three different datasets, with SING being up to 62x faster than UCR Suite-P. UCR Suite GPU is significantly slower, due to the cost of transferring the raw data in the GPU (recall that the raw data size is much bigger than the GPU memory).

V. Related Work

There has been a flurry of activity, especially during the last years, related to the development of scalable data series similarity search techniques [15], [17]–[21], [24]–[29], [45], [46], [54]. Nevertheless, none of these techniques considered the use of GPUs for performing part of the computations. Note that in this work, we focus on indexing structures specialized to data series, since other techniques cannot provide comparable performance in this high-dimensional context [12].

Changkyu Kim et al. [55] have designed a tree base index on GPU but only for single dimension, integer key data. Gieseke et al. [56] propose the Buffer k-d Tree to process NN queries on a GPU. The goal of this approach is to efficiently process together large batches of queries. In contrast, we focus on exploratory search, where queries arrive one by one: the results of an analyst's query determine what the next query will be. The use of GPUs in order to support spatio-temporal queries has been examined in the past [33], [57]. Doraiswamy et al. [57]. In a more recent work, Li et al. [33], design an update-efficient GPU accelerated grid index for k-NN queries for road networks. We note that all the works addressing spatio-temporal queries propose and use indices designed for a 2-dimensional space, and there is no straight-forward way to apply it on data series.Moreover, earlier studies have shown that the indices used in these cases (such as grid-based, Rtrees, or Kd-Trees) do not perform well for the high-dimensional data series collections [12], [21]. Previous work has considered the use of GPUs for speeding up similarity search using Locality Sensitive Hashing (LSH) [34], [58], [59]. However, all these works only support approximate query answering. In our work, we focus on exact query answering that is required by several applications [4]. Zhu et al. [35], [60], present a GPU implementation of *Matrix Profile*, an algorithm used to identify data series motifs (i.e., frequent subsequences). Zimmerman et al. [61] extend the above work by describing a solution that operates on a cluster of distributed GPUs. We observe that Matrix Profile is used to reason about short subsequences within a long data series, while we are interested in similarity search between a query series and a dataset containing a large number of data series. Doruk Sart et al. [47] implement a GPU-based brute force similarity search algorithm that we compare against (i.e., UCR Suite GPU).

VI. Conclusions

Data series similarity search remains an important and challenging problem. We propose SING, the first data series index that answers similarity search queries with CPU+GPU co-processing. In our experiments with several synthetic and real datasets, SING extends considerably the scalability of similarity search: it is up to 5.1x faster at query answering time than the state-of-the-art parallel in-memory approach, up to 62x faster than the state-of-the-art parallel serial scan algorithm, and achieves exact similarity search times at interactive speeds: as low as 32msec on 100GB datasets.

[Acknowledgements] Supported by Chinese Scholarship Council, FMJH Program PGMO, EDF, Thales, HIPEAC 4, and NVIDIA Corporation for donating a Titan Xp GPU. Part of work performed while P. Fatourou was visiting LIPADE, and while B. Peng was visiting CARV, FORTH ICS.

References

[1] T. Palpanas, "Data series management: The road to big sequence analytics," *SIGMOD Record*, 2015.

[6]Note that this algorithm was developed for subsequence matching, while in our case we are solving the problem of whole matching [12].

[2] K. Zoumpatianos and T. Palpanas, "Data series management: Fulfilling the need for big sequence analytics," in *ICDE*, 2018.

[3] A. J. Bagnall, R. L. Cole, T. Palpanas, and K. Zoumpatianos, "Data series management," *Dagstuhl Reports*, 9(7), 2019.

[4] T. Palpanas and V. Beckmann, "Report on the first and second interdisciplinary time series analysis workshop (ITISA)," *SIGREC*, 48(3), 2019.

[5] K. Kashino, G. Smith, and H. Murase, "Time-series active search for quick retrieval of audio and video," in *ICASSP*, 1999.

[6] L. Ye and E. Keogh, "Time series shapelets: a new primitive for data mining," in *SIGKDD*. ACM, 2009.

[7] P. Huijse, P. A. Estevez, P. Protopapas, J. C. Principe, and P. Zegers, "Computational intelligence challenges and applications on large-scale astronomical time series databases," *CIM*, 2014.

[8] U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco, "Practical data prediction for real-world wireless sensor networks," *TKDE*, 2015.

[9] P. Boniol, M. Linardi, F. Roncallo, and T. Palpanas, "Automated Anomaly Detection in Large Sequences," in *ICDE*, 2020.

[10] P. Boniol and T. Palpanas, "Series2Graph: Graph-based Subsequence Anomaly Detection for Time Series," *PVLDB*, 2020.

[11] M. Linardi, Y. Zhu, T. Palpanas, and E. J. Keogh, "Matrix Profile Goes MAD: Variable-Length Motif And Discord Discovery in Data Series," in *DAMI*, 2020.

[12] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim, "The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art," *PVLDB*, 2018.

[13] ——, "Return of the Lernaean Hydra: Experimental Evaluation of Data Series Approximate Similarity Search," *PVLDB*, 2019.

[14] K. Echihabi, K. Zoumpatianos, and T. Palpanas, "Scalable machine learning on high-dimensional vectors: From data series to deep network embeddings," in *WIMS*, 2020.

[15] D. Rafiei and A. Mendelzon, "Similarity-based queries for time series data," in *SIGMOD*, 1997.

[16] I. Assent, R. Krieger, F. Afschari, and T. Seidl, "The ts-tree: efficient time series search and retrieval," in *EDBT*, 2008.

[17] J. Shieh and E. Keogh, "isax: disk-aware mining and indexing of massive time series datasets," *DMKD*, 2009.

[18] P. Schäfer and M. Högqvist, "Sfa: a symbolic fourier approximation and index for similarity search in high dimensional datasets," in *EDBT*, 2012, pp. 516–527.

[19] A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. Keogh, "Beyond One Billion Time Series: Indexing and Mining Very Large Time Series Collections with iSAX2+," *KAIS*, vol. 39, no. 1, 2014.

[20] Y. Wang, P. Wang, J. Pei, W. Wang, and S. Huang, "A data-adaptive and dynamic segmentation index for whole matching on time series," *VLDB*, 2013.

[21] K. Zoumpatianos, S. Idreos, and T. Palpanas, "Ads: the adaptive data series index," *VLDB J.*, 2016.

[22] A. Gogolou, T. Tsandilas, T. Palpanas, and A. Bezerianos, "Progressive similarity search on time series data," in *EDBT*, 2019.

[23] A. Gogolou, T. Tsandilas, K. Echihabi, A. Bezerianos, and T. Palpanas, "Data Series Progressive Similarity Search with Probabilistic Quality Guarantees," in *SIGMOD*, 2020.

[24] B. Peng, T. Palpanas, and P. Fatourou, "Paris: The next destination for fast data series indexing and query answering," *IEEE BigData*, 2018.

[25] ——, "Paris+: Data series indexing on multi-core architectures," *TKDE*, 2020.

[26] ——, "Messi: In-memory data series indexing," in *ICDE*, 2020.

[27] D. E. Yagoubi, R. Akbarinia, F. Masseglia, and T. Palpanas, "Dpisax: Massively distributed partitioned isax," in *ICDM*, 2017.

[28] D.-E. Yagoubi, R. Akbarinia, F. Masseglia, and T. Palpanas, "Massively distributed time series indexing and querying," *TKDE (to appear)*, 2018.

[29] J. Wu, P. Wang, N. Pan, C. Wang, W. Wang, and J. Wang, "Kv-match: A subsequence matching approach supporting normalization and time warping," in *ICDE*, 2019.

[30] O. Levchenko, B. Kolev, D. E. Yagoubi, D. E. Shasha, T. Palpanas, P. Valduriez, R. Akbarinia, and F. Masseglia, "Distributed algorithms to find similar time series," in *ECML/PKDD*, 2019.

[31] A. Guillaume, "Head of Operational Intelligence Department Airbus. Personal communication." 2017.

[32] J.-D. Fekete and R. Primet, "Progressive analytics: A computation paradigm for exploratory data analysis," *CoRR*, 2016.

[33] C. Li, Y. Gu, J. Qi, J. He, Q. Deng, and G. Yu, "A gpu accelerated update efficient index for knn queries in road networks," in *ICDE*. IEEE, 2018.

[34] J. Zhou, Q. Guo, H. Jagadish, L. Krcal, S. Liu, W. Luan, A. K. Tung, Y. Yang, and Y. Zheng, "A generic inverted index framework for similarity search on the gpu," in *ICDE*. IEEE, 2018.

[35] Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-C. M. Yeh, G. Funning, A. Mueen, P. Brisk, and E. Keogh, "Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins," in *ICDM*. IEEE, 2016.

[36] T. Rakthanmanon, E. J. Keogh, S. Lonardi, and S. Evans, "Time series epenthesis: Clustering time series streams requires ignoring some data," in *ICDM*, 2011, pp. 547–556.

[37] J. Shieh and E. Keogh, "iSAX: disk-aware mining and indexing of massive time series datasets," *DMKD*, no. 1, 2009.

[38] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *CSUR*, 2009.

[39] A. Mueen, E. J. Keogh, Q. Zhu, S. Cash, M. B. Westover, and N. B. Shamlo, "A disk-aware algorithm for time series motif discovery," *DAMI*, 2011.

[40] R. Agrawal, C. Faloutsos, and A. N. Swami, "Efficient similarity search in sequence databases," in *FODO*, 1993.

[41] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *SIGKDD*, 2012.

[42] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality reduction for fast similarity search in large time series databases," *KAIS*, 2001.

[43] J. Shieh and E. Keogh, "i sax: indexing and mining terabyte sized time series," in *SIGKDD*, 2008.

[44] H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas, "Coconut: A scalable bottom-up approach for building data series indexes," *PVLDB*, 2018.

[45] M. Linardi and T. Palpanas, "Scalable, variable-length similarity search in data series: The ulisse approach," *PVLDB*, 2019.

[46] T. Palpanas, "Evolution of a Data Series Index," *CCIS*, vol. 1197, 2020.

[47] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul, "Accelerating dynamic time warping subsequence search with gpus and fpgas," in *ICDM*, 2010.

[48] http://helios.mi.parisdescartes.fr/ themisp/sing/, 2020.

[49] K. Zoumpatianos, Y. Lou, I. Ileana, T. Palpanas, and J. Gehrke, "Generating data series query workloads," *VLDB J.*, 2018.

[50] E. J. Keogh and S. Kasetty, "On the need for time series data mining benchmarks: A survey and empirical demonstration," *DAMI*, 2003.

[51] A. Mueen, S. Nath, and J. Liu, "Fast approximate correlation for massive time-series data," in *SIGMOD*, 2010.

[52] "Incorporated Research Institutions for Seismology – Seismic Data Access," http://ds.iris.edu/data/access/, 2016.

[53] "Southwest university adult lifespan dataset (sald)," http://fcon_1000.projects.nitrc.org/indi/retro/sald.html, 2018.

[54] M. Linardi and T. Palpanas, "Scalable data series subsequence matching with ulisse," *VLDBJ*, 2020.

[55] C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. D. Nguyen, T. Kaldewey, V. W. Lee, S. A. Brandt, and P. Dubey, "Fast: fast architecture sensitive tree search on modern cpus and gpus," in *SIGMOD*, 2010.

[56] F. Gieseke, J. Heinermann, C. Oancea, and C. Igel, "Buffer kd trees: processing massive nearest neighbor queries on gpus," in *ICML*, 2014.

[57] H. Doraiswamy, H. T. Vo, C. T. Silva, and J. Freire, "A gpu-based index to support interactive spatio-temporal queries over historical data," in *ICDE*, 2016.

[58] J. Pan and D. Manocha, "Fast gpu-based locality sensitive hashing for k-nearest neighbor computation," in *SIGSPATIAL*, 2011, pp. 211–220.

[59] ——, "Bi-level locality sensitive hashing for k-nearest neighbor computation," in *IEEE ICDE*, 2012, pp. 378–389.

[60] Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-C. M. Yeh, G. Funning, A. Mueen, P. Brisk, and E. Keogh, "Exploiting a novel algorithm and gpus to break the ten quadrillion pairwise comparisons barrier for time series motifs and joins," *KAIS*, 2018.

[61] Z. Zimmerman, K. Kamgar, N. S. Senobari, B. Crites, G. Funning, P. Brisk, and E. Keogh, "Matrix profile xiv: Scaling time series motif discovery with gpus to break a quintillion pairwise comparisons a day and beyond," in *SoCC*, 2019, pp. 74–86.