PADDi: Highly Scalable Parallel Algorithm for Discord Discovery on Multi-GPU Clusters

Y. A. Kraeva[1, *] **and M. L. Zymbler**[1, **]

(Submitted by A. M. Elizarov)

[1] *South Ural State University, Chelyabinsk, 454080 Russia*

Received November XX, 2024; revised November XX, 2024; accepted January XX, 2025

Abstract—Currently, in a wide spectrum of subject domains, time series data mining requires the efficient subsequence anomaly discovery in a very long time series, which cannot be entirely placed in RAM. At present, one of the best approaches to solving such a problem is to formalize the anomaly as a discord, a given-length subsequence that is maximally far away from its non-overlapping nearest neighbor. In the article, we introduce a novel parallel algorithm called PADDi (PALMAD-based Anomaly Discovery on Distributed GPUs), which discovers arbitrary-length discords in a very long time series on a high-performance cluster with nodes, each of which is equipped with multiple GPUs. The algorithm exploits two-level parallelism: first, when the time series is divided into equal-length fragments stored on disks associated with the cluster nodes, and second, when a fragment is split into equal-length segments to be processed by GPUs of the respective node. To implement data exchanges between nodes and calculations on GPUs within a node, we employ MPI and CUDA technologies, respectively. The algorithm performs as follows. Firstly, in each segment processed by one GPU, the algorithm selects potential discords and then discards false positives, resulting in the local candidate set. Next, local candidate sets are sent among cluster nodes in an "all-to-all" manner, resulting in a global candidate set. Then, each cluster node refines the global candidates within its fragment, obtaining the local resulting set of true positive discords. Finally, each cluster node sends the local resulting sets to a master node, which outputs the end result as the intersection of the received local resulting sets. Extensive experiments over real-world and synthetic million-length time series on various configurations of two high-performance clusters with up to 64 nodes and different models of GPU onboard showed that our algorithm's scalability remains linear without stagnation or degradation.

2010 Mathematical Subject Classification: 62M10, 65Y05

Keywords and phrases: *time series, anomaly, discord discovery, parallel algorithm, high-performance cluster, GPU, CUDA.*

INTRODUCTION

Over the past decade, time series anomaly discovery remains one of the most topical problems for researchers and practitioners in a wide spectrum of subject domains [1]: Internet of Things [2], smart cities [3] and buildings [4], monitoring of HPC-systems [5], etc. Within the time series anomaly discovery scope, one of the most challenging task is subsequence anomaly detection, where we need to find successive points in time whose collective behavior is unusual, although each point individually is not necessarily an outlier.

Focusing on the subsequence anomaly detection task, we apply the discord concept [6], which is currently considered one of the best approaches to formalize and discover subsequence anomaly [7, 8]. Discord is intuitively defined as a given-length subsequence that is maximally far away from its

* E-mail: kraevaya@susu.ru

** E-mail: mzym@susu.ru

non-overlapping nearest neighbor. We can point out three following key serial discord discovery algorithms: HOTSAX, DRAG, and MERLIN. HOTSAX [6] discovers discords in a time series, which is entirely placed in RAM. DRAG [10] introduces the range discord concept (a discord that is at least a user-defined threshold away from its non-overlapping nearest neighbor) and implements the range discord discovery for the case when a time series is stored on disk. Finally, MERLIN [11], as opposed to its predecessors, discovers discords of every possible length in a specified range determining the above-threshold automatically.

In our latest study, we introduced the PD3 [12] and PALMAD [13] algorithms, which accelerate, respectively, DRAG and MERLIN on a graphics processing unit (GPU). However, this does not allow us to discover arbitrary-length discords in a very long time series that is stored on the disk (and cannot be entirely placed in RAM).

The article pushes our previous research forward and contributes as follows:

- We introduce a novel parallel algorithm called PADDi (PALMAD-based Anomaly Discovery on Distributed GPUs), which discovers arbitrary-length discords in a very long time series on a high-performance cluster with nodes, each of which is equipped with multiple GPUs.
- We carry out extensive experiments to evaluate PADDi over real-world and synthetic million-length time series on various configurations of two high-performance clusters with different models of GPU, where our algorithm's scalability remains linear without stagnation or degradation. We establish a repository [14], which contains the algorithm's source code and supplemental data to facilitate the reproducibility of our research.

The remainder of the article is organized as follows. Section 1 briefly discusses related works. In Section 2, we introduce the notation and formal definitions, along with a short description of the serial and parallel algorithms our study is based on Section 3 introduces PADDi, our novel parallel algorithm for discord discovery in the time series. In Section 4, we discuss the results of the experimental evaluation of PADDi. Finally, in Conclusions, we summarize the results obtained and suggest directions for further research.

1. RELATED WORK

Currently, the discord concept proposed by Keogh *et al.* [6], is considered one of the best approaches to time series anomaly discovery [7, 8]. Discord is intuitively defined as a given-length subsequence that is maximally far away from its non-overlapping nearest neighbor. In the above-cited article, Keogh *et al.* introduced the HOTSAX (Heuristically Ordered Time series using Symbolic Aggregate ApproXimation) algorithm for discords discovery for the case when a time series is entirely placed in RAM. The algorithm encodes the time series subsequences by the SAX transform [9] and scans all the pairs of subsequences. During the scanning, HOTSAX calculates the distance between subsequences and finds the maximum among the distances to the nearest neighbor. Herewith, the algorithm discards unpromising subsequences without calculating distances. However, HOTSAX suffers from the fact that the length of a time series to discover discords in is limited by the RAM size.

In [10], Keogh *et al.* proposed the improvement of HOTSAX, the DRAG (Discord Range Aware Gathering) algorithm, which discovers discords in a time series placed on a disk, not in RAM. DRAG introduces the range discord concept, where such a discord is at least a user-defined threshold away from its non-overlapping nearest neighbor. The algorithm performs in two phases, where, on each phase, it linearly scans the time series at once. In the first phase, candidate selection, DRAG collects potential range discords, whereas in the second phase, discord refinement, it prunes the false positives. However, DRAG does not provide a user with a formal way to choose the threshold to ensure efficient discord discovery. In addition, DRAG (like HOTSAX) discovers discords of a fixed, not arbitrary length, which would be much more useful for a user who is not an expert in the subject domain.

In [11], Keogh *et al.* introduced the MERLIN algorithm, which removes the above-mentioned limitations of DRAG. MERLIN discovers discords of every possible length in a specified range through repeated calls of DRAG and adaptive selection of the threshold. However, MERLIN is a serial algorithm, and its parallelization could significantly increase the efficiency of discord discovery.

Research that addresses the parallelization of the fixed-length discord discovery can be briefly described as follows. In our previous work [15, 16], we proposed a way to accelerate HOTSAX on Intel many-core CPUs and GPUs. In [17], we introduced a parallelization scheme for DRAG on a high-performance cluster with Intel many-core CPUs. As it was shown in the experiments, such an algorithm significantly outruns two DRAG-based rivals, DDD (Distributed Discord Discovery) [18] and PDD (Parallel Discord Discovery) [19], since they employ intensive data exchanges between cluster nodes. We can also point out two following algorithms for GPU: first, by Zhu *et al.* [20], which employs the computational patterns, and second, KBF_GPU (Brute-Force for K-distance discord) [21] that discovers K-distance discords, some modification of the classic discord concept.

To the best of our knowledge, no research has addressed the accelerating of the arbitrary-length discords discovery with GPU or any other parallel hardware architecture except our algorithm PALMAD (Parallel Arbitrary Length MERLIN-based Anomaly Discovery) introduced earlier in [13]. PALMAD basically follows MERLIN, the original serial algorithm, but employs our derived recurrent formulas to calculate the mean values and standard deviations of subsequences. Further, we use such data in calculations of the Euclidean distance between subsequences and, eventually, significantly reduce the costs of entire calculations. Furthermore, PALMAD repeatedly calls PD3 (Parallel DRAG-based Discord Discovery), our developed parallel version of the original DRAG algorithm proposed earlier in [12]. However, PALMAD makes it possible to discover arbitrary-length discords on one GPU just for the case when time series should be entirely placed in RAM, not for a very long time series that is stored on the disk. This requires modifications to our developments for a high-performance cluster with nodes, each of which is equipped with several GPUs, thus, in this article, we consider such a challenging task.

2. PRELIMINARIES

Prior to introducing the proposed algorithm for unsupervised labeling of long time series, below, in Subsections 2.1 and 2.2, respectively, we first give basic definitions and notation, and then briefly describe underlying developments our approach is based on.

2.1. Formal Definitions and Notation

Time series and subsequence. A *time series* is a chronologically ordered sequence of real-valued numbers

$$T = \{t_i\}_{i=1}^{n}, \quad t_i \in \mathbb{R}.$$

A *subsequence* $T_{i,m}$ of a time series T is its subset of m successive elements that starts at the ith position

$$T_{i,m} = \{t_k\}_{k=i}^{i+m-1}, \quad 1 \le i \le n-m+1, \quad 3 \le m \ll n.$$

Discords. Given a time series T and its two subsequences $T_{i,m}$ and $T_{j,m}$, we say that they are *non-self match* to each other at distance $\mathrm{Dist}(T_{i,m},\, T_{j,m})$ if $|i-j| \ge m$. Let us denote a non-self match of a subsequence $C \in S_T^m$ by M_C.

Given a time series T, its subsequence $D \in S_T^m$ is said to be the *discord* if D has the largest distance to its nearest non-self match:

$$\forall C \in S_T^m \quad \min_{M_D \in S_T^m} \mathrm{Dist}(D,\, M_D) > \min_{M_C \in S_T^m} \mathrm{Dist}(C,\, M_C).$$

Given the positive real number r, the discord at a distance of at least r from its nearest non-self match is called the *range discord*. That is, the range discord $D \in S_T^m$ w.r.t. the parameter r meets the following:

$$\min_{M_D \in S_T^m} \mathrm{Dist}(D,\, M_D) \ge r.$$

Further, for brevity, we use the term "discord", meaning range discord, unless otherwise is clearly indicated.

Distance function. A *distance function* for any two m-length subsequences is a nonnegative and symmetric function $\mathrm{Dist} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$. We employ the Euclidean metric that is defined as follows. Let us have $X, Y \in S_T^m$, then the Euclidean distance between the subsequences is calculated as below:

$$\mathrm{ED}(X, Y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}.$$

In our study, the algorithms deal with subsequences that have been previously z-normalized to have a mean of zero and a standard deviation of one. The z-normalization of a subsequence $X = \{x\}_{i=1}^m \in S_T^m$ is defined as a subsequence $\hat{X} = \{\hat{x}\}_{i=1}^m$, where

$$\hat{x}_i = \frac{x_i - \mu_X}{\sigma_X}, \quad \mu_X = \frac{1}{m}\sum_{i=1}^{m} x_i, \quad \sigma_X = \sqrt{\frac{1}{m}\sum_{i=1}^{m} x_i^2 - \mu_X^2}.$$

In our study, to provide the highest possible performance of discord discovery, we employ the squared z-normalized Euclidean distance as the $\mathrm{Dist}(\cdot\,,\cdot)$ function. Hereinafter, we denote by $\hat{\mathrm{ED}}$ the Euclidean distance between two z-normalized subsequences: $\hat{\mathrm{ED}}(X, Y) = \mathrm{ED}(\hat{X}, \hat{Y})$. To compute $\hat{\mathrm{ED}}^2(\cdot, \cdot)$, we employ the technique proposed in [22] that allows for faster calculation than in above classic formula:

$$\mathrm{Dist}(X, Y) = \hat{\mathrm{ED}}^2(X, Y) = 2m\left(1 - \frac{\langle X, Y\rangle - m\cdot\mu_X\cdot\mu_Y}{m\cdot\sigma_X\cdot\sigma_Y}\right).$$

2.2. Previous building-block algorithms

Serial algorithms: DRAG and MERLIN. DRAG [10] discovers given-length discords and performs in two phases: candidate selection (collecting potential range discords) and discord refinement (pruning the false positives). In the first phase, the algorithm scans through the time series, and for each subsequence it validates each candidate already in the candidate set is a discord. If a candidate fails the validation, then it is removed from this set. Finally, the new subsequence is either added to the candidate set, or it is pruned. In the second phase, the algorithm scans through the time series, calculating the distance between each subsequence and each candidate. If the distance is less than the given threshold, then the candidate is permanently excluded from the candidate set as a false positive. If the above distance is less than the best-so-far distance to the nearest neighbor, then the current distance to the nearest neighbor is updated. MERLIN [11] discovers discords with length in the range $minL..maxL$ through repeated running DRAG along with the adaptive selection of the threshold. At each step, MERLIN calculates the arithmetic mean μ and standard deviation σ of the distances from the last five discords found to their nearest neighbors, and then calls DRAG with the threshold $r = \mu - 2\sigma$. If DRAG does not discover a discord, then σ is subtracted from r until DRAG completes successfully. To set the parameter r for the first five discord lengths $minL..minL + 4$, MERLIN employs advanced formulas introduced in [11].

Parallel algorithms: PD3 and PALMAD. Our developed algorithm PD3 [12] accelerates both phases of DRAG with a GPU. PD3 exploits segment-wise processing of a time series and the above-mentioned fast calculation of the squared z-normalized Euclidean distance. Each block of GPU threads treats subsequences of its segment as local candidates and processes the subsequences located to the right of the segment and do not overlap with the candidates. The thread block scans the subsequences chunk-wisely, where the number of elements in a chunk is equal to the segment length. The refinement phase is parallelized similarly, involving, however, only segments with a non-empty set of local candidates and subsequences located to the left of the segment. PALMAD [13], our developed algorithm, parallelizes MERLIN on a GPU. PALMAD basically follows the original serial algorithm and repeatedly calls PD3. However, PALMAD employs our derived recurrent formulas for calculation of the mean values and standard deviations of subsequences, significantly reducing the costs of calculation of the distances between subsequences.

3. DISCORD DISCOVERY ON MULTI-GPU CLUSTER

Below, we introduce PADDi (PALMAD-based Anomaly Discovery on Distributed GPUs), a novel parallel algorithm for arbitrary-length discord discovery in a very long time series on a high-performance cluster with nodes, each of which is equipped with multiple GPUs. In Subsections 3.1 and 3.2, respectively, we describe our algorithm's data layout and parallelization scheme.

3.1. Data Layout

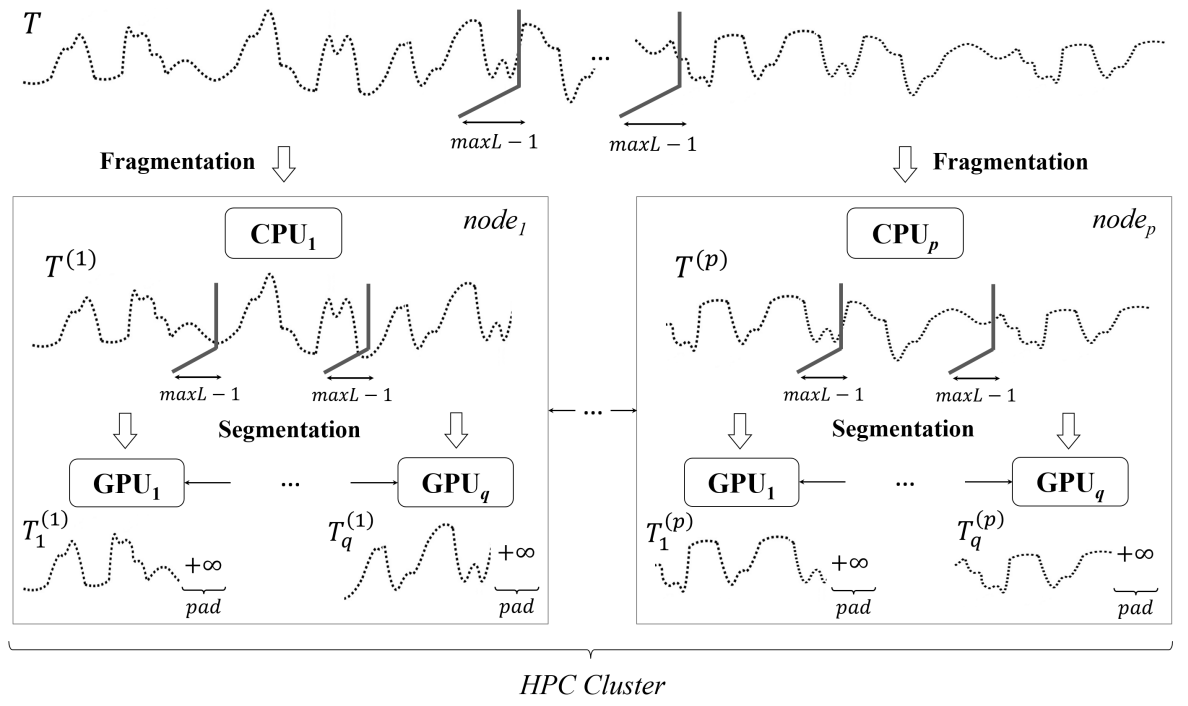


Figure 1. Data layout in PADDi

In Fig. 1, we show the data layout of our algorithm, which employs two-level data parallelism. In the first level, we split the given time series T into a set of equal-length fragments $T^{(1)}, \ldots, T^{(p)}$, where p is the number of cluster nodes, and place the fragments on disks of the nodes. We suppose that p is selected in such a way that the fragment and the algorithm's supplemental data can be entirely placed in the node RAM. To prevent loss of results at the junction of fragments, we use overlapping as follows: at the end of each fragment (except for the last one), we add $maxL - 1$ points taken from the beginning of the next fragment.

In the second level of parallelism, we divide each fragment into equal-length segments according to the number of GPUs installed on the node. To make segmentation of each fragment $T^{(i)}$, we employ the above-described overlapping technique, resulting in a set of segments $T_1^{(i)}, \ldots, T_q^{(i)}$, $1 \leq i \leq p$, where q is the number of GPUs that each cluster node is equipped with. Furthermore, each segment is aligned to ensure a load balance of the GPU threads processing its segment. The segment length is chosen as a multiple of the GPU warp size; if this is not the case, then the segment is padded on the right with dummy points that have the $+\infty$ value.

3.2. Parallelization

In Fig. 2, we show the parallelization scheme of our algorithm. For each discord length within the given range $minL..maxL$, PADDi performs in three phases: local selection, local refinement, and global refinement. Among all cluster nodes, we choose a master node, which further collects the

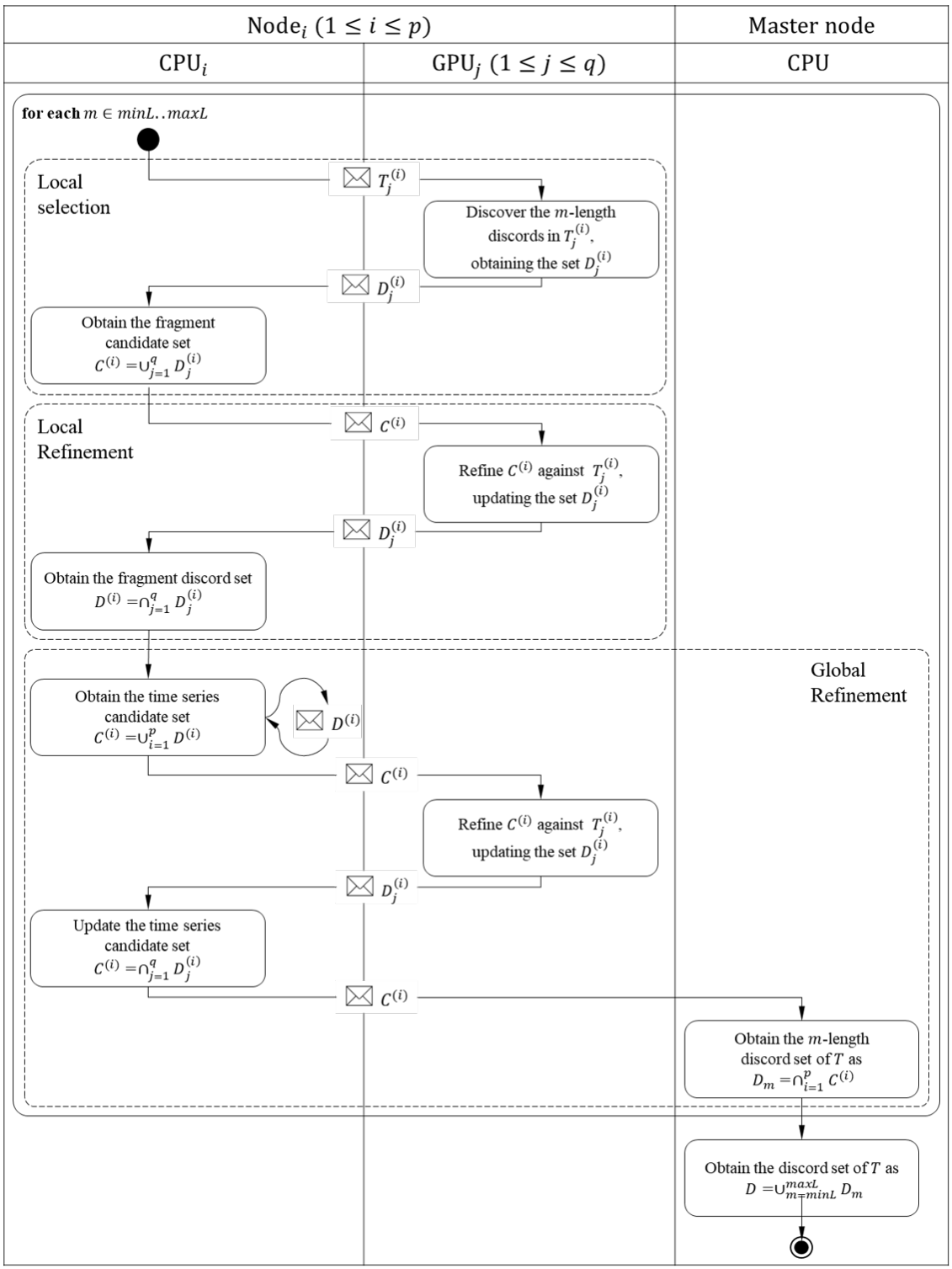


Figure 2. Parallelization scheme in PADDi

discords obtained by other nodes and outputs the final result. To implement data exchanges across the cluster nodes and calculations on GPUs installed on a node, we employ, respectively, the MPI and CUDA technologies.

In the first phase, for each fragment $T^{(i)}$, PADDi, applying our earlier developed algorithm PD3, selects potential discords within each segment $T_j^{(i)}$ and then discards false positives of them,

resulting in the discord set $D_j^{(i)}$. After that, each GPU transfers its result to the CPU of the cluster node, where the phase resulting set is obtained as $C^{(i)} = \cup_{j=1}^{q} D_j^{(i)}$.

The resulting set of the first phase, however, may contain false positives, since the discords found in one segment are not compared to the subsequences from other segments of the same fragment, and thus may not be discords within the entire fragment. This is the reason that we consider $C^{(i)}$ the fragment candidate set and proceed with the second phase, local refinement, where PADDi performs as follows. Firstly, $C^{(i)}$ is transferred to each GPU of the cluster node. Next, on each GPU, we refine $C^{(i)}$ against the segment $T_j^{(i)}$, resulting in the updated discord set $D_j^{(i)}$. Finally, each GPU transfers its result to the CPU of the cluster node, where the phase resulting set is obtained as an intersection of the partial results, $D^{(i)} = \cap_{j=1}^{q} D_j^{(i)}$.

As before, the resulting set of the second phase needs to be refined, since it consists only of discords discovered in one fragment. In the third phase, PADDi performs global refinement as follows. First, each fragment discord set $D^{(i)}$ is sent by the respective CPU among the cluster nodes in an "all-to-all" manner (using the `MPI_Allgatherv` function), resulting in the whole time series candidate set, $C^{(i)} = \cup_{i=1}^{p} D^{(i)}$. Next, as in the previous phase, we send the candidate set to the GPU of the cluster node and refine it against the respective segment, updating the set $D_j^{(i)}$. Then, like in the previous phase, we transfer the partial results from GPU to CPU and obtain their intersection: $C^{(i)} = \cap_{j=1}^{q} D_j^{(i)}$. Finally, we send the updated discord set $C^{(i)}$ to the master node (using the `MPI_Gatherv` function), which obtains the m-length discord set of the whole time series intersecting the received partial results: $D_m = \cap_{i=1}^{p} C^{(i)}$. In the end, the master node obtains the all-length discords discovered set as $D = \cup_{minL}^{maxL} D_m$.

While in the local selection and local refinement phases, we employ our earlier developed algorithm PD3, we implemented the global refinement from scratch. In this phase, we calculate the distance between each m-length candidate from the set $C^{(i)}$ and each m-length subsequence in the fragment $T^{(i)}$ through the advanced formula given in Section 2.1. In the memory of the GPU installed on the cluster node, we represent the above-mentioned sets as matrices $C \in \mathbb{R}^{|C^{(i)}| \times m}$ and $S \in \mathbb{R}^{(n-m+1) \times m}$, respectively. Multiplication of the matrices S and C results in the matrix $P \in \mathbb{R}^{(n-m+1) \times |C^{(i)}|}$, where each element represents the scalar product of the corresponding rows of the matrix S (subsequences) and columns of the transposed matrix $C^{\intercal}$ (candidates). To calculate the above multiplication on the GPU, we employ the parallel block-wise algorithm [23]. Further, we use the elements of the matrix P to calculate the distances between the candidates and subsequences through the above-mentioned formula. Finally, we take as a result each candidate, where the distance to its nearest neighbor exceeds the threshold.

4. EXPERIMENTAL EVALUATION

In the experiments, to confirm the efficiency of PADDi, we evaluate the performance and speedup our algorithm. We designed the experiments to be easily reproducible with our repository [14], which contains the algorithm's source code and supplemental data. Below, Section 4.1 describes the metrics, hardware, and time series employed in the experiments, and Section 4.2 discusses the experimental results.

4.1. Experimental Setup

Metrics. The algorithm's performance is interpreted as its running time. For each experiment, we ran PADDi ten times and took the average value as the final running time. The speedup of our algorithm employing k graphics processors is calculated as $S(k) = {}^{t_1}/_{t_k}$, where t_1 and t_k are the running times of PALMAD on one GPU and PADDi on k GPUs, respectively. We omit the comparison of our algorithm's scalability with parallel rivals, since our review of related work (see Section 1) does not reveal any other discord discovery solution oriented to high-performance clusters with GPU nodes. We also skip the comparison of PADDi and MERLIN performance results since,

Table 1. Time series employed in the experiments

Time series	Length, n	Discord length range, $minL..maxL$	Domain
ECG	$2 \cdot 10^6$	64..128	ECG of an adult patient
GAP			Power demand of a household in France
MGAB			Synthetic time series generated by the Mackey–Glass equation

obviously, the original serial algorithm is significantly (orders of magnitude) inferior to its parallel descendant.

Datasets. In our study, we employed the time series listed in Table 1. Each time series consists of two million points. For all time series, we discover discords with length starting from 64 and ending by 128. The ECG [24] time series contains the measurements of an adult patient's electrocardiogram. The GAP [25] time series represents minute-by-minute indicators of the total energy consumption of a private house in France in 2006–2010. The MGAB [26] time series is synthetically generated based on the Mackey—Glass equation (nonlinear differential equation with time delay) [27].

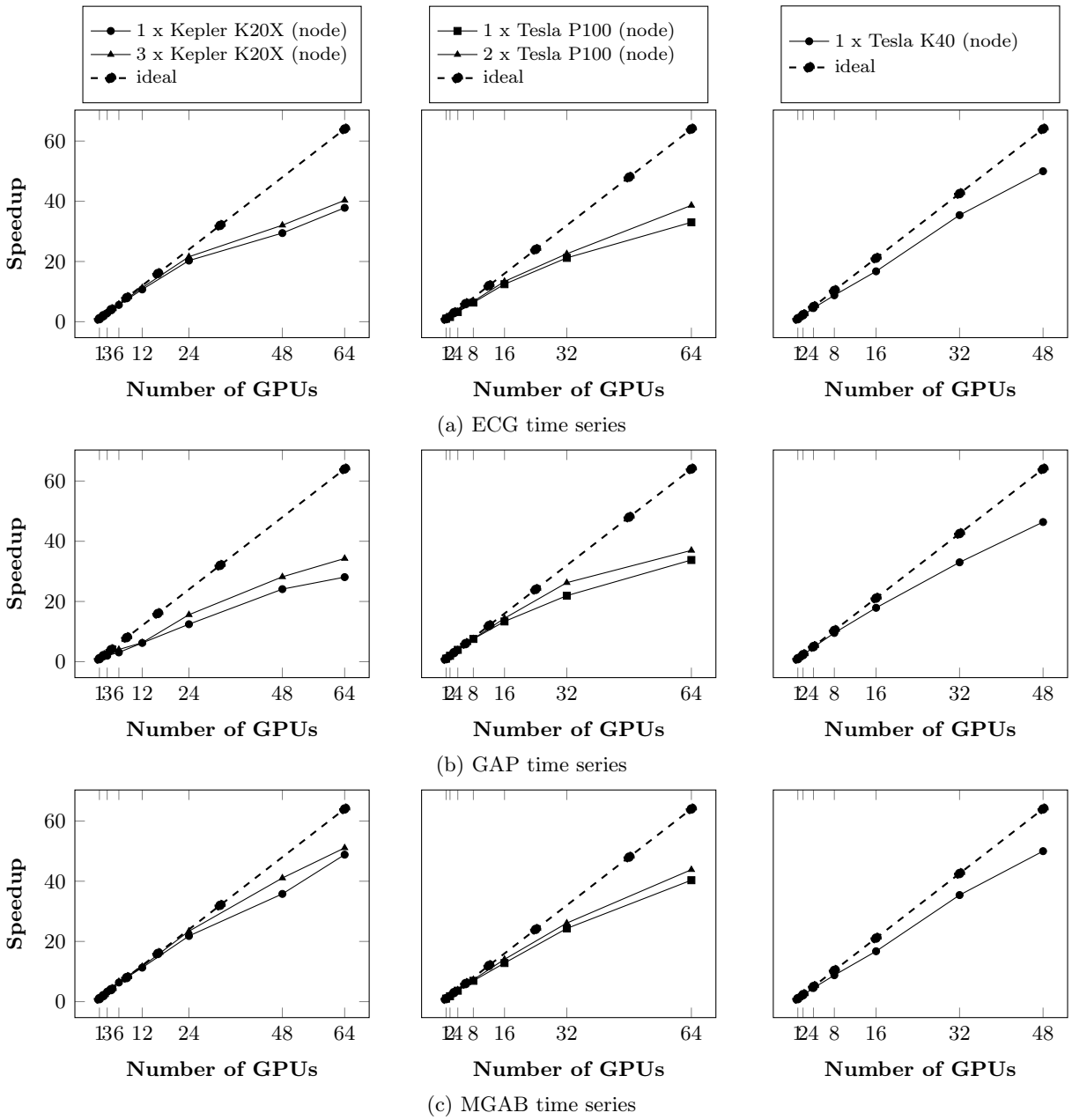
Table 2. Hardware platform of the experiments

	Feature	GPU-MSU		GPU-UNN
Node	Brand and family	NVIDIA Tesla		NVIDIA Kepler
	Model	K40	P100	K20X
	Number of CUDA cores	2 880	3 584	2 688
	Core frequency, GHz	0.745	1.19	0.732
	RAM, Gb	11.56	16	6
	Peak performance (double precision), TFLOPS	1.682	4	1.31
Cluster	Number of nodes	48	64	64
	GPUs per node	1	2	3

Hardware. Table 2 summarizes the hardware platform we use in the experiments. We employ two high-performance clusters with multi-GPU nodes, namely Lomonosov-2 [5] and Lobachevsky [28]. As can be seen, we run our algorithm on the following three configurations of the above clusters: 48×K40, 64×2×P100, and 64×3×K20X.

4.2. Results and Discussion

Speedup and performance. Figures 3 and 4 depict the scalability of our algorithm over million-length time series from different subject domains, where the results show a picture common to all time series and all hardware configurations involved in the experiments. As can be seen, speedup and performance decrease compared to ideal when the algorithm runs on a configuration



(a) ECG time series

(b) GAP time series

(c) MGAB time series

Figure 3. Speedup of PADDi

with a larger number of cluster nodes. Nevertheless, the algorithm's scalability remains linear, and there is no stagnation or degradation.

Impact of data exchanges on scalability. Fig. 5, depicts the proportion of the algorithm's running time between the local and global stages, where the former includes the local selection and local refinement, and the latter corresponds to the global refinement. In addition, for the corresponding configurations and time series, we show the size of candidates involved in data exchanges between cluster nodes. It can be seen that the longer running time of the global stage corresponds to a larger size of candidates involved in data exchanges between cluster nodes. This, in turn, results in a lower speedup and performance of the algorithm (cf. Figs. 3, 4 and Fig. 5).

Next, let us discuss the algorithm's scalability when it is run on a high-performance cluster configured with different numbers of nodes, where for all the configurations it has the same number of GPUs in total (for the configurations 64×2×P100 and 64×3×K20X). Fig. 5 shows that when

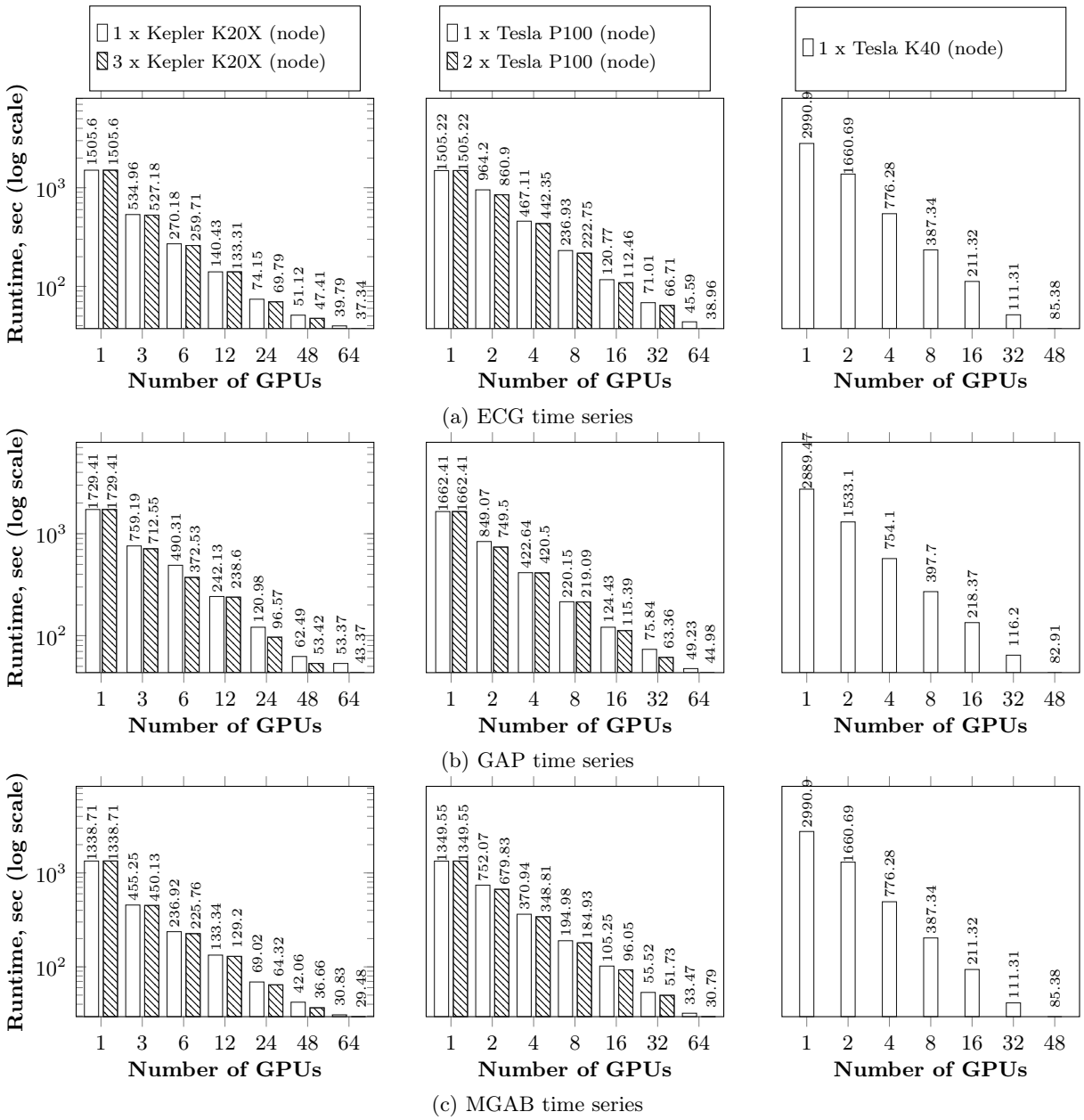
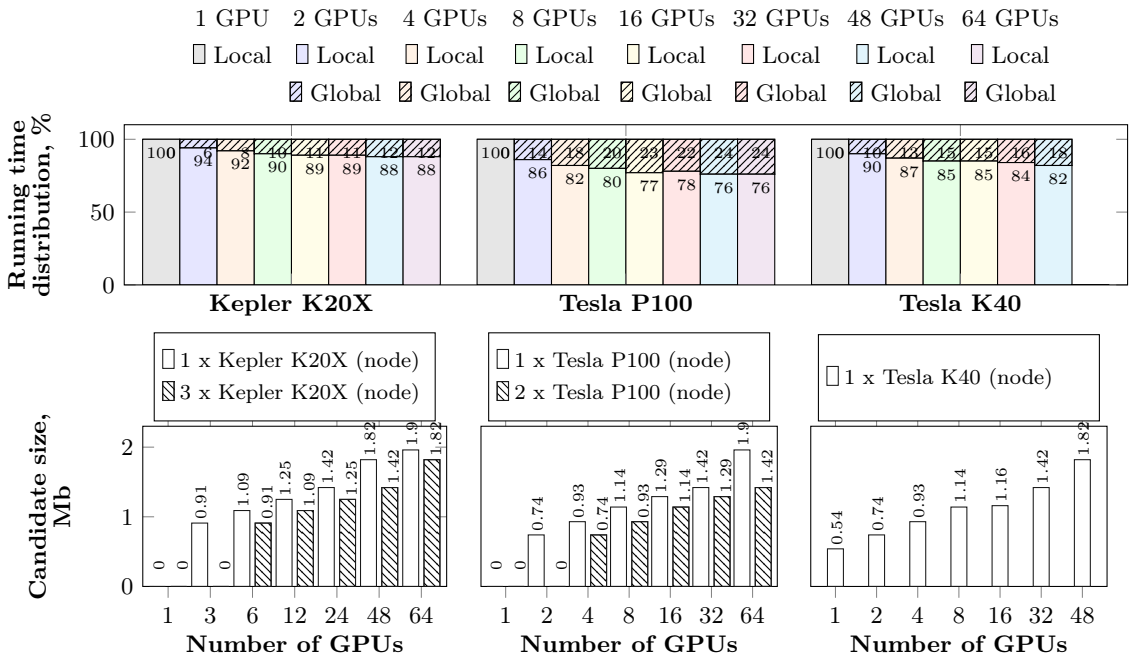


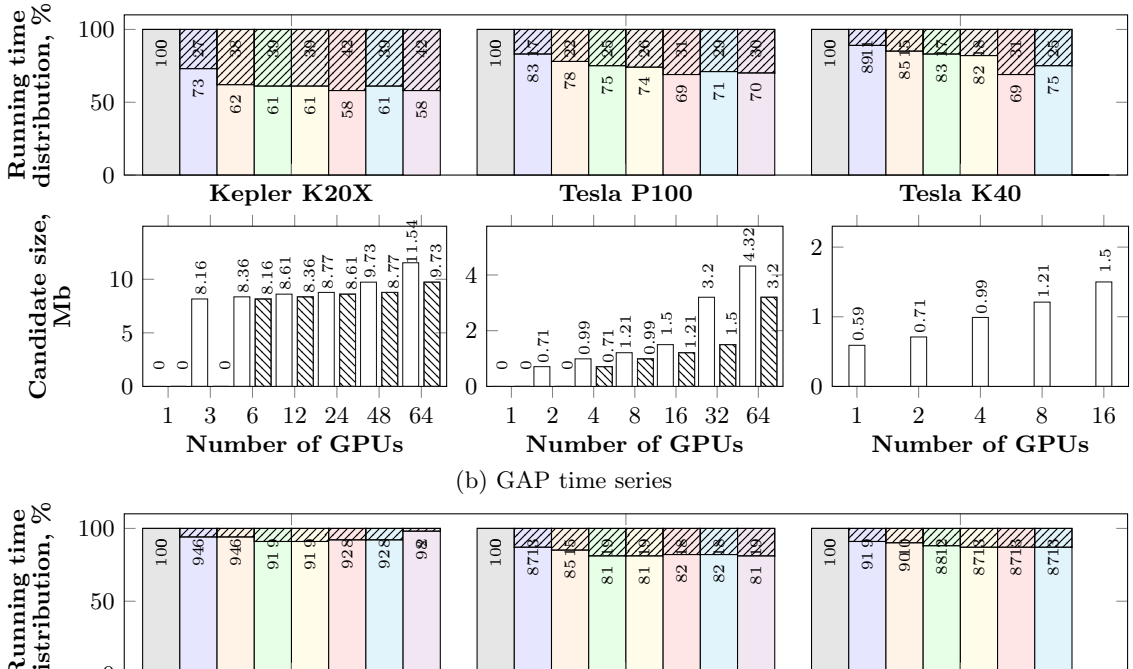
Figure 4. Performance of PADDi

PADDi runs on cluster nodes with the highest possible number of GPUs, it is more scalable compared to the case of a larger number of cluster nodes with one GPU on each node. The reason is that one GPU discards fewer candidates in its fragment. This, in turn, increases the size of the data to be sent, and the time spent on the refinement phase due to the larger amount of calculations.

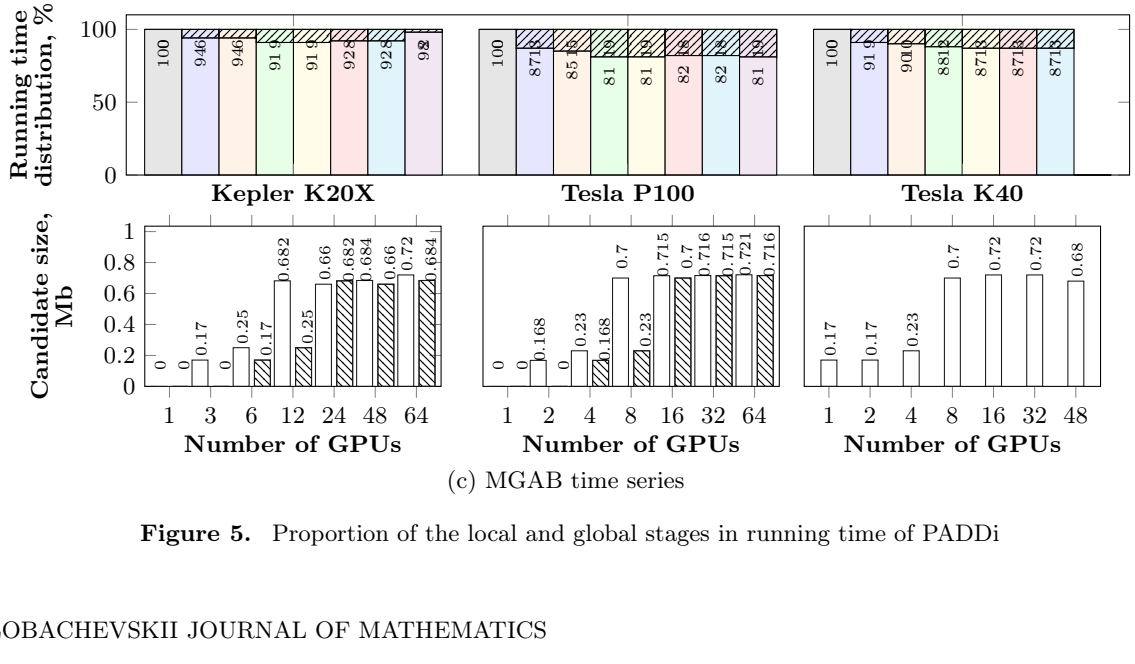
It is worth noting that discord discovery on a hardware architecture with distributed memory cannot avoid exchanges of candidates. Candidates selected in some fragment of a time series, but refined within that fragment without involving the candidates of other fragments of the time series, obviously do not have to be true positive discords w.r.t. the definition in Section 2 [6].



(a) ECG time series



(b) GAP time series



(c) MGAB time series

Figure 5. Proportion of the local and global stages in running time of PADDi

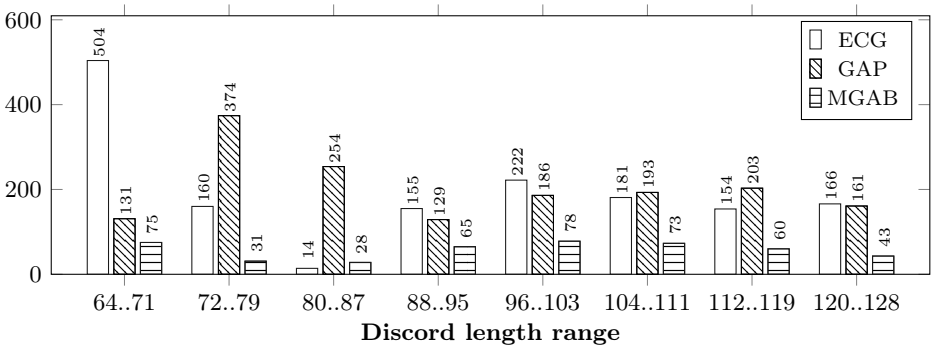


Figure 6. Total number of discovered discords

Correctness of discord discovery. In the experiments, we do not evaluate our algorithm's accuracy of anomaly discovery, since it requires a time series with anomalies manually marked by an expert in the subject domain. However, such a markup is hardly ever possible for the million-length time series involved in our experiments. Anyway, in the experiments, we confirmed that PADDi produces the correct results as follows. We employ SCAMP [29], which is currently the fastest parallel algorithm to calculate the matrix profile (MP) [30] of a time series. MP is defined as an array, where the ith element is the distance from the ith subsequence of the original time series to its non-overlapping nearest neighbor. Since discords can be found as local maxima of MP, we verified that discords discovered by PADDi are the same as those obtained as a by-product of the MP calculation. In addition, Fig. 6 depicts the statistical results and shows that the discords found have approximately the same distribution by length. For any length, the number of discords found is less than 0.08% of the total number of subsequences in the time series, confirming the intuitive idea that anomalies are very rare.

CONCLUSIONS

In this article, we address the subsequence anomaly discovery in a very long time series, which cannot be entirely placed in RAM. Such a problem remains one of the most topical issues for researchers and practitioners in a wide spectrum of subject domains. Our study is based on the discord concept by Keogh *et al.* [6], which is currently considered one of the best approaches to formalize and discover subsequence anomalies. Discord is defined as a given-length subsequence that is maximally far away from its non-overlapping nearest neighbor. We also employ the algorithms by Keogh *et al.*, DRAG [10] and MERLIN [11], which implements, respectively, fixed- and arbitrary-length discord discovery in a time series stored on disk.

In the article, we propose a novel parallel algorithm called PADDi (PALMAD-based Anomaly Discovery on Distributed GPUs), which discovers arbitrary-length discords in a very long time series on a high-performance cluster with nodes, each of which is equipped with multiple GPUs. PADDi employs our earlier developed parallel algorithms PD3 [12] and PALMAD [13], which accelerate, respectively, DRAG and MERLIN on a GPU.

Our algorithm exploits two-level data parallelism. At the first level, the time series is divided into equal-length fragments distributed over disks associated with the nodes of the high-performance cluster. The second level of parallelism is implemented by splitting each fragment into equal-length segments, which are processed by GPUs installed on the cluster node. Both fragments and segments are formed with overlapping to prevent loss of results at junctions of fragments and segments, respectively. We employ MPI and CUDA to implement data exchanges between cluster nodes and calculations on GPUs within a node, respectively. PADDi iterates the specified range of discord lengths, where at each iteration, it performs parallel preprocessing and discord discovery for the respective length. The discord discovery is performed in two following phases by each cluster node. In the first phase, through our PD3 parallel algorithm, in each segment processed by a separate GPU installed on the node, we select potential discords and then discard false positive of them, resulting in the local candidate set. In the second phase, local candidate sets are sent among cluster nodes in an "all-to-all" discipline, resulting in a global candidate set. Next, each cluster node refines the global candidates within its fragment based on the parallel block multiplication of the matrix of

candidates and the matrix of segment subsequences, obtaining the local resulting set of true positive discords. Finally, each cluster node sends the local resulting set to a master node, which outputs the end result as the intersection of the received local resulting sets.

We carry out extensive experiments to evaluate PADDi over real-world and synthetic million-length time series on two high-performance clusters, Lomonosov-2 (Moscow State University, Russia) and Lobachevsky (State University of Nizhny Novgorod, Russia), which have up to 64 nodes with different models of GPU. The results are common to all time series and all hardware configurations involved in the experiments: our algorithm's performance and speedup remain linear without stagnation or degradation; the more GPUs are installed on one node of the cluster, the greater speedup we obtain. To facilitate the reproducibility of our study, we establish a repository [14], which contains the algorithm's source code and supplemental data.

In our further research, we plan to implement the above-described approach for the case of high-performance cluster based on nodes with multiple Intel many-core CPU.

FUNDING

This work was financially supported by the Russian Science Foundation (grant no. 23-21-00465). The study was carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University and at Lobachevsky State University of Nizhny Novgorod.

CONFLICT OF INTEREST

The authors declare that they have no conflict of interest.

REFERENCES

1. P. Boniol, J. Paparrizos, and T. Palpanas, "New Trends in Time Series Anomaly Detection," in *Proceedings of the 26th International Conference on Extending Database Technology, EDBT 2023, March 28–31, 2023, Ioannina, Greece* (OpenProceedings.org, 2023), pp. 847–850. https://doi.org/10.48786/EDBT.2023.80
2. S. Kumar, P. Tiwari, and M. L. Zymbler, "Internet of Things is a revolutionary approach for future technology enhancement: A review," J. Big Data, **6**, 111 (2019). https://doi.org/10.1186/s40537-019-0268-2
3. S. Ivanov, K. Nikolskaya, G. Radchenko, L. Sokolinsky, and M. Zymbler, "Digital twin of city: Concept overview," Proc. of 2020 Global Smart Industry Conf., GloSIC 2020, Chelyabinsk, Russia, November 17-19, 2020, 178–186. https://doi.org/10.1109/GloSIC50886.2020.9267879
4. M. Zymbler, Y. Kraeva, E. Latypova, S. Kumar, D. Shnayder, and A. Basalaev, "Cleaning sensor data in smart heating control system," Proc. of 2020 Global Smart Industry Conf., GloSIC 2020, Chelyabinsk, Russia, November 17-19, 2020, 375–381. https://doi.org/10.1109/GloSIC50886.2020.9267813
5. V. V. Voevodin, A. S. Antonov, D. A. Nikitenko, P. A. Shvets, S. I. Sobolev, I. Y. Sidorov, K. S. Stefanov, Vad. V. Voevodin, and S. A. Zhumatiy, "Supercomputer Lomonosov-2: Large scale, deep monitoring and fine analytics for the user community," Supercomput. Front. Innov. **6** (2), 4–11 (2019). https://doi.org/10.14529/jsfi190201
6. E. J. Keogh, J. Lin, and A. W. Fu, "HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence," in *Proceedings of the 5th IEEE International Conference on Data Mining, ICDM 2005, November 27–30, 2005, Houston, Texas, USA* (IEEE Comput. Soc., 2005), pp. 226–233. https://doi.org/10.1109/ICDM.2005.79
7. V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM Comput. Surv. **41** (3), 15:1–15:58 (2009). https://doi.org/10.1145/1541880.1541882
8. A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A Review on Outlier/Anomaly Detection in Time Series Data," ACM Comput. Surv. **54** (3), 56:1–56:33 (2022). https://doi.org/10.1145/3444690
9. J. Lin, E. J. Keogh, S. Lonardi, and B. Y. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, DMKD 2003, June 13, 2003, San Diego, California, USA* (ACM, 2003), pp. 2–11. https://doi.org/10.1145/882082.882086
10. D. Yankov, E. J. Keogh, and U. Rebbapragada, "Disk aware discord discovery: Finding unusual time series in terabyte sized datasets," in *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28–31, 2007, Omaha, Nebraska, USA*. (IEEE, 2007), pp. 381–390. https://doi.org/10.1109/ICDM.2007.61

11. T. Nakamura, M. Imamura, R. Mercer, and E. J. Keogh, "MERLIN: Parameter-free discovery of arbitrary length anomalies in massive time series archives," in *Proceedings of the 20th IEEE International Conference on Data Mining, ICDM 2020, November 17–20, 2020, Sorrento, Italy* (IEEE, 2020), pp. 1190–1195. https://doi.org/10.1109/ICDM50108.2020.00147
12. Y. Kraeva and M. Zymbler, "A parallel discord discovery algorithm for a graphics processor," Pattern Recognition and Image Analysis **33** (2), 101–112 (2023). https://doi.org/10.1134/S1054661823020062
13. M. Zymbler and Ya. Kraeva, "High-Performance Time Series Anomaly Discovery on Graphics Processors," Mathematics **11** (14), 3193 (2023). https://doi.org/10.3390/math11143193
14. Y. Kraeva, PADDi: PALMAD-based Anomaly Discovery on Distributed GPUs. https://github.com/kraevaya/PADDi. Accessed Dec 10, 2023.
15. M. Zymbler, A. Polyakov, and M. Kipnis, "Time series discord discovery on Intel many-core systems," in *13th International Conference, PCT 2019, Kaliningrad, Russia, April 2-4, 2019, Revised Selected Papers*. Communications in Computer and Information Science **1063**, 168–182. https://doi.org/10.1007/978-3-030-28163-2_12
16. M. Zymbler, "A parallel discord discovery algorithm for time series on many-core accelerators," Numerical Methods and Programming **20** (3), 211–223 (2019). https://doi.org/10.26089/NumMet.v20r320
17. M. Zymbler, A. Grents, Y. Kraeva, and S. Kumar, "A parallel approach to discords discovery in massive time series data," Computers, Materials & Continua **66** (2), 1867–1878 (2021). https://doi.org/10.32604/cmc.2020.014232
18. Y. Wu, Y. Zhu, T. Huang, and et al., "Distributed discord discovery: Spark based anomaly detection in time series," in *Proceedings of the 17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, Proceedings of the 7th IEEE International Symposium on Cyberspace Safety and Security, CSS 2015, and Proceedings of the 12th IEEE International Conference on Embedded Software and Systems, ICESS 2015, August 24–26, 2015, New York, NY, USA* (IEEE, 2015), pp. 154–159. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.228
19. T. Huang, Y. Zhu, Y. Mao, and et al., "Parallel discord discovery," in *Advances in Knowledge Discovery and Data Mining: 20th Pacific-Asia Conference, PAKDD 2016, April 19–22, 2016, Auckland, New Zealand, Proceedings, Part II* (Springer, 2016), pp. 233–244. https://doi.org/10.1007/978-3-319-31750-2_19
20. B. Zhu, Y. Jiang, M. Gu, and Y. Deng, "A GPU acceleration framework for motif and discord based pattern mining," IEEE Trans. Parallel Distributed Syst. **32** (8), 1987–2004 (2021). https://doi.org/10.1109/TPDS.2021.3055765
21. T. Huang, Y. Zhu, Y. Wu, and W. Shi, "J-distance discord: An improved time series discord definition and discovery method," in *Proc. of 2015 IEEE International Conference on Data Mining Workshop, ICDMW* (IEEE, 2015), pp. 303–310. https://doi.org/10.1109/ICDMW.2015.120
22. A. Mueen, S. Nath, and J. Liu, "Fast approximate correlation for massive time-series data," in *Proc. of the ACM SIGMOD Int. Conf. on Management of Data, SIGMOD 2010, June 6–10, 2010, Indianapolis, Indiana, USA* (ACM Press, 2010), pp. 171–182. https://doi.org/10.1145/1807167.1807188
23. NVIDIA. CUDA C++ Best Practices Guide. Release 12.3. 2023. https://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf. Accessed Apr 18, 2023.
24. A. L. Goldberger, L. A. N. Amaral, L. Glass, and et al., "PhysioBank, PhysioToolkit, and PhysioNet components of a new research resource for complex physiologic signals," Circulation **101** (23), 215–220 (2000). https://doi.org/10.1161/01.CIR.101.23.e215
25. G. Hebrail and A. Berard, Individual household electric power consumption. https://doi.org/10.24432/C58K54. Accessed Apr 18, 2023.
26. M. Thill, W. Konen, and T. Bäck, Markus-Thill/MGAB: The Mackey–Glass anomaly benchmark. Version v1.0.1. https://doi.org/10.5281/ZENODO.3762385. Accessed Apr 18, 2023.
27. M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," Science **197** (4300), 287–289 (1977). https://doi.org/10.1126/science.267326
28. Lobachevsky supercomputer. https://hpc-education.unn.ru/en/resources. Accessed Dec 01, 2023.
29. Z. Zimmerman, K. Kamgar, N. S. Senobari, and et al., "Matrix profile XIV: Scaling time series motif discovery with GPUs to break a quintillion pairwise comparisons a day and beyond," in *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2019, November 20–23, 2019, Santa Cruz, CA, USA* (ACM, 2019), pp. 74–86. https://doi.org/10.1145/3357223.3362721
30. C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, Z. Zimmerman, D. F. Silva, A. Mueen, and E. Keogh, "Time series joins, motifs, discords and shapelets: A unifying view that exploits the matrix profile," Data Min. Knowl. Discov. **32** (1), 83–123 (2018). https://doi.org/10.1007/s10618-017-0519-9