GPU-Accelerated Matrix Profile Computing for Streaming Time Series

A. A. Valiulin[1, *] **and M. L. Zymbler**[1, **]

(Submitted by A. M. Elizarov)

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

Received February 22, 2026; revised April 10, 2026; accepted April 16, 2026

Abstract—Currently, the mining of streaming time series has become increasingly important in a wide range of applications. The matrix profile (MP) is a simple yet powerful exploratory time series mining tool, which provides the discovery of numerous time series mining primitives without requiring prior domain knowledge. The research community has introduced a substantial number of MP computation algorithms. However, to the best of our knowledge, no existing method is both GPU-oriented and stream-oriented. In this article, we introduce the StreaMP (Streaming Matrix Profile) algorithm for GPUs, which overcomes the above limitation. Our algorithm processes incoming data in a segment-wise manner, accumulating the time series arriving from a sensor in RAM. StreaMP merges the MPs of the segment and the time series accumulated so far using a proposed multistep scheme. To compute MPs, StreaMP is able to apply any GPU-based algorithm, which encapsulates all details of parallel processing. The MP built by StreaMP makes it possible to discover repeated and anomalous patterns in the streaming time series in linear time. Experimental evaluation shows that StreaMP outperforms SCAMP, which is currently the fastest GPU-based MP computation algorithm for batch processing.

2010 Mathematical Subject Classification: `62M10, 65Y05`

Keywords and phrases: *streaming time series, matrix profile, parallel algorithm, SCAMP, GPU, CUDA*

INTRODUCTION

Currently, the mining of time series data, which arrive in a streaming manner from various edge devices, has become increasingly important in a wide range of applications: Internet of Things [1], smart cities [2] and buildings [3], personal healthcare [4], etc. In such scenarios, key tasks include motif and anomaly discovery, which require efficient and scalable processing techniques.

The Matrix Profile (MP) [5] is defined as a real-valued array, where the ith element is the distance from the ith subsequence of the original time series to its nearest neighbor (i.e., to the most similar non-overlapping subsequence). The MP has emerged as a flexible and versatile tool for extracting a multitude of time series mining primitives, including motifs (repeated patterns) [6], discords (anomalies) [7], snippets (behavioral patterns) [8], chains (evolving patterns) [9], etc. The MP allows for revealing underlying patterns and anomalies without requiring prior domain knowledge, which makes it well-suited for exploratory time series mining in many domains: digital industry [10], medicine [11], seismology [12], etc.

A considerable body of research has focused on developing efficient algorithms for the MP computing: STAMP [5], STOMP [5], STOMPI [5], SCRIMP++ [13], and SCAMP [14]. Some of the developments above have been parallelized for GPUs. However, to the best of our knowledge, STOMPI is the only existing algorithm designed for streaming operation, but it does not leverage GPU parallel processing capabilities. In this study, we address the limitation above. In summary, the main contributions of this article are as follows:

* E-mail: valiulinaa@susu.ru

** E-mail: mzym@susu.ru

- We introduce a novel algorithm called StreaMP (Streaming Matrix Profile), which enables GPU-accelerated MP computation for streaming time series. Our algorithm processes incoming data in a segment-wise manner and merges MPs of the segment and the time series read so far through our proposed multistep schema. To compute MPs, StreaMP employs a GPU-based algorithm, which we treat as a hyperparameter that encapsulates all parallel processing details. The MP built by StreaMP makes it possible to discover motifs and discords in the streaming time series in linear time.

- We carry out experiments to evaluate StreaMP against SCAMP [14], which is currently considered the fastest GPU-based MP computation algorithm for batch mode, employing SCAMP as an underlying tool to compute MPs on GPU. In the experiments, we measure performance as follows: for SCAMP, it is the running time to process the entire fixed-length time series; for StreaMP, it is the running time to process only the last segment. Experimental results showed that, for the time series longer than hundreds of thousands, StreaMP surpasses SCAMP in performance by several times: from one and a half to 27 for time series lengths from hundreds of thousands to tens of millions. Finally, we made our data and code publicly available for the research community to confirm and extend our work.

The remainder of the article is organized as follows. Section 1 provides a brief overview of related work and formal definitions. In Section 2, we introduce the StreaMP algorithm. In Section 3, we discuss the results of the experimental evaluation of StreaMP. Finally, in Conclusion, we summarize our findings and outline future research directions.

1. BACKGROUND

Prior to introducing our approach, we first briefly discuss related work and then give basic definitions and notation.

1.1. Related work

The matrix profile (MP) concept is proposed by Keogh *et al.* in [5], where also the following MP computation algorithms are introduced: STAMP, STOMP, and STOMPI.

STAMP (Scalable Time series Anytime Matrix Profile) [5] offers a faster MP computation by leveraging the MASS (Mueen's Algorithm for Similarity Search) algorithm [16] for efficient distance computation. STAMP employs an iterative refinement strategy: it approximates MP and then refines it by recomputing distance profiles for random subsequence subsets. As an anytime algorithm [17], STAMP progressively improves accuracy and can be terminated early at any point, offering a trade-off between computation time and accuracy. GPU-STAMP [5] further accelerates MP computation as a version of STAMP ported to graphics processors.

STOMP (Scalable Time series Ordered-search Matrix Profile) [5] builds upon STAMP to further improve MP computation speed. STOMP precomputes the dot products required by the MASS algorithm and reuses them across multiple subsequence comparisons, reducing redundant calculations. This optimization allows STOMP to achieve significantly faster performance than STAMP, particularly for longer time series. A GPU-accelerated version of STOMP, called GPU-STOMP [18], primarily targets batch processing of entire time series rather than streaming data.

SCRIMP++ [13] is a highly efficient, exact algorithm for computing MP. It uses a divide-and-conquer strategy, along with several optimizations to minimize redundant calculations. SCRIMP++ can process large datasets in a relatively short amount of time, making it a competitive alternative to other CPU-based MP algorithms. However, it is not inherently designed for parallel execution on GPUs or for processing streaming data.

SCAMP (Scalable Matrix Profile) [14] is a highly optimized algorithm that avoids the iterative refinement approach of STAMP and STOMP, and instead leverages fast Fourier transforms (FFTs) to compute all pairwise distances between subsequences efficiently. Importantly, SCAMP is designed for parallel execution on GPUs, making it well-suited for large datasets. However, SCAMP is primarily intended for processing entire time series in batch mode, rather than streaming data.

STOMPI (STOMP Incremental) [5] adapts STOMP for streaming data and assumes that there is an initial time series and its MP is pre-computed. The algorithm processes incoming data elements, updating MP incrementally. To the best of our knowledge, STOMPI is the only existing algorithm capable of computing MP in a true streaming fashion. However, STOMPI is not inherently parallelized to leverage the processing power of GPUs, which motivates the development of our StreaMP algorithm.

1.2. Formal Definitions and Notation

Time series and subsequence. Streaming data to be mined are a form of time series. A *time series* is a chronologically ordered sequence of real-valued numbers:

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

where n is the time series length and denoted by $|T|$.

When mining a time series, we primarily process relatively short segments, called subsequences. A *subsequence* $T_{i,m}$ of a time series T is a set of m consecutive elements starting at position i:

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

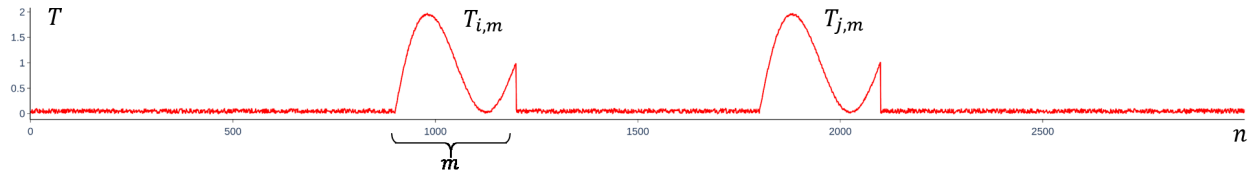
We denote the set of all m-length subsequences in T by S_T^m, where $|S_T^m| = n-m+1$.

Trivial matches. In time series mining, *trivial matches* refer to subsequences that are very close in time and therefore highly similar [6]. Trivial matches should be excluded from processing since they obscure more significant patterns and distort the results of mining algorithms. For the given subsequence $T_{i,m}$, we denote its non-trivial matches as $\mathcal{N}$ and avoid the trivial ones by imposing an exclusion zone of $m/2$ before and after the subsequence [6]:

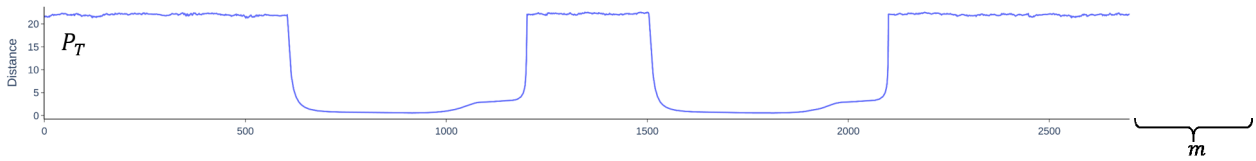
$$\mathcal{N}_{T_{i,m}} = \{T_{j,m} \in S_T^m \mid |i-j| \ge \lceil m/2 \rceil\}.$$

Nearest neighbor. To measure the similarity of time series subsequences, let us given a nonnegative and symmetric *distance function* $\mathrm{Dist} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$. For the given subsequence, its closest non-trivial match is called the (first) *nearest neighbor*. The fact that some subsequence N is the nearest neighbor of the given subsequence Q is indicated by the Boolean function $\theta_{1\mathrm{NN}} : S_T^m \times S_T^m \to \mathbb{B}$:

$$\theta_{1\mathrm{NN}}(Q, N) = \texttt{TRUE} \Leftrightarrow N = \arg\min_{C \in \mathcal{N}_Q} \mathrm{Dist}(Q, C).$$



(a) time series T



(b) matrix profile P_T

Figure 1. An example of the matrix profile

Matrix profile. At this point, we are ready to formally define the matrix profile [5] (see Fig. 1 for illustration). For the given time series T and subsequence length m, the *matrix profile (MP)* is an array $P_T \in \mathbb{R}^{n-m+1}$, where each element holds the distance between the respective subsequence and its nearest neighbor:

$$P_T = \{d_i\}_{i=1}^{n-m+1}, \quad d_i = \mathrm{Dist}(T_{i,m}, N),\ \theta_{1\mathrm{NN}}(T_{i,m}, N) = \texttt{TRUE}.$$

To locate the nearest neighbors, the matrix profile is accompanied by a supplemental index structure. The *matrix profile index (MPI)* is an array $I_T \in \mathbb{N}^{n-m+1}$, where each element holds the index of the nearest neighbor of the respective subsequence:

$$I_T = \{idx_i\}_{i=1}^{n-m+1}, \quad \theta_{1\mathrm{NN}}(T_{i,m}, T_{idx_i,m}) = \texttt{TRUE}.$$

Motifs and discords as the MP by-products. Motif points out repeated patterns in the time series and can informally be described as a pair of the most similar non-trivial match subsequences [5]. Formally, for the given time series T and subsequence length m, the *motif* is a pair of subsequences $\{T_{left,\,m},\, T_{right,\,m}\}$ for which the following holds:

$$\forall i,\, j\ \mathrm{Dist}(T_{left,\,m},\, T_{right,\,m}) \leq \mathrm{Dist}(T_{i,\,m},\, T_{j,\,m}),\ T_{right,\,m} \in \mathcal{N}_{T_{left,\,m}},\, T_{j,\,m} \in \mathcal{N}_{T_{i,\,m}}.$$

Discord reflects an anomalous pattern in the time series and can informally be described as a subsequence with the most dissimilar nearest neighbor [7]. Formally, for the given time series T and subsequence length m, the *discord* is a subsequence $T_{i,\,m}$ for which the following holds:

$$\forall j\ \mathrm{Dist}(T_{j,\,m},\, N) \leq \mathrm{Dist}(T_{i,\,m},\, N),\ T_{j,\,m} \in \mathcal{N}_{T_{i,\,m}}.$$

Obviously, for the given time series and subsequence length, top-1 motif and discord can be discovered through, respectively, the MP minimum and maximum:

$$\mathit{Top1Motif} = \{T_{left,\,m},\, T_{right,\,m}\},\ \mathit{left} = \arg\min_{1 \leq i \leq n-m+1} P_T(i),\ \mathit{right} = I_T(\mathit{left}),$$

$$\mathit{Top1Discord} = T_{i,\,m},\ i = \arg\max_{1 \leq i \leq n-m+1} P_T(i).$$

The rest motifs and discords are trivially generalized as the MP local minima and maxima, respectively.

Join matrix profile. The MP concept can be generalized to the case of two time series as follows [5]. Let us given two time series A and B (that are not necessarily of equal length) and the subsequence length m. Then it is possible to use the above MP and MPI definitions up to a change in the definition of the nearest neighbor. Avoiding the above requirement regarding the non-trivial matches, we demand that for any subsequence in A, its nearest neighbor should be some subsequence in B. Formally speaking, $\forall A_{i,\,m} \in S_A^m\, \mathcal{N}_{A_{i,\,m}} \equiv S_B^m$. In this case, we call P_{AB} and I_{AB} the *join matrix profile (JoinMP)* and *join matrix profile index (JoinMPI)*, respectively. From this point of view, MP and MP can be called *self-JoinMP* and *self-JoinMPI*, respectively. It is worth noting that, commonly, $P_{AB} \neq P_{BA}$ and $I_{AB} \neq I_{BA}$.

Distance function. To conclude formal definitions, we mention that all the MP computation algorithms above use the z-normalized Euclidean distance (ED), which is defined as follows. Let us have $X,\, Y \in S_T^m$, then

$$\mathrm{Dist}(X,\, Y) = \mathrm{ED}(\hat{X},\, \hat{Y}), \quad \mathrm{ED}(X,\, Y) = \left[\sum_{i=1}^{m} (x_i - y_i)^2\right]^{1/2},$$

where $\hat{X} = \{\hat{x}\}_{i=1}^m$, the z-normalized version of the subsequence X, is defined as follows:

$$\hat{x}_i = \frac{x_i - \mu_X}{\sigma_X}, \quad \mu_X = \frac{1}{m}\sum_{i=1}^{m} x_i, \quad \sigma_X = \left[\frac{1}{m}\sum_{i=1}^{m} x_i^2 - \mu_X^2\right]^{1/2}.$$

2. GPU-BASED STREAMING MATRIX PROFILE

Below, we introduce StreaMP, a novel algorithm for computing MP of streaming time series on GPU. The basic idea is to keep the time series arriving from a sensor in RAM and process incoming data segment-by-segment, where the segment length is the algorithm's parameter. During processing, we merge the MP of the time series accumulated so far with that of the current segment. To compute MPs for segments, we employ a GPU-based algorithm, which serves as a hyperparameter of StreamMP and encapsulates all parallel processing details, including data transfer to the GPU and retrieval of results back to the CPU. Next, the current segment is simply appended to the time series accumulated at the moment. However, if we simply concatenate the MP of the time series read so far with that of the segment, the resulting MP clearly does not satisfy the definition of MP. To make the above-mentioned concatenation possible and produce the correct streaming MP, we must perform several operations, including computing JoinMPs on the GPU that involve both the time series accumulated so far and the current segment.

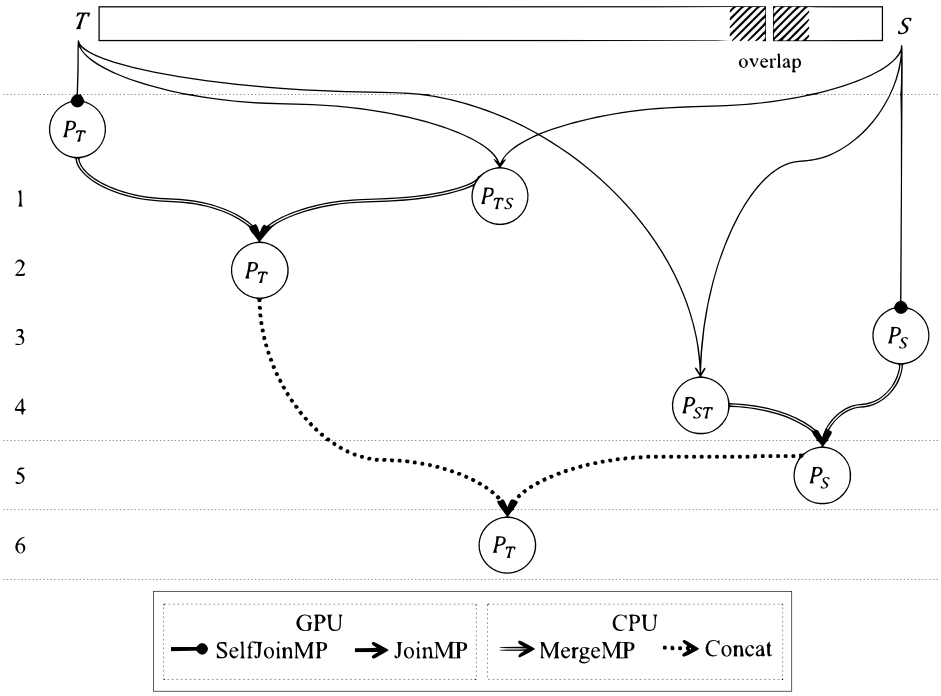


Figure 2. Calculation schema of StreaMP

The computing schema we propose is as follows. Figure 2 shows the moment when the MP of the streaming time series read so far has been computed, and the current segment has been read from the sensor and awaits processing. Let us denote the streaming time series read so far, its MP, and the current segment as T, P_T, and S, respectively. The schema consists of the following six steps. Firstly, we compute P_{TS}, the JoinMP of T and S. Notably, we overlap T and S to avoid losing results at the junction of the two operands, namely, $m-1$ last points of T, where m is the subsequence length, are copied into the beginning of S. In the second step, we merge MPs P_T and P_{TS}. This operation simply computes the element-wise minima of the operands and stores the results in the first operand. Further, we will use the resulting MP P_T for concatenation. In the third and fourth steps, we compute, respectively, self-JoinMP P_S and JoinMP P_{ST}, where the latter is correct thanks to the above-described overlapping. In the fifth step, we merge MPs P_S and P_{ST} placing the result into P_S. Finally, concatenation of MPs P_T and P_S results in the correct streaming MP P_T.

In Algs. 1 and 2, we show pseudocodes of StreaMP and the operation of merging MPs, respectively, which together implement the above-described ideas. Below, we clarify some details of StreaMP (Alg. 1), since the MergeMPs algorithm (Alg. 2) is self-explanatory. After initialization of data structures (see line 1), StreaMP processes the streaming time series segment-by-segment in an "eternal" loop until there is an interruption (see lines 2–24). In the loop body, firstly, the algorithm reads data from the sensor into the segment (see lines 3–10), overlapping each segment starting from the second with the time series read so far (see line 6). Then, we calculate the self-JoinMP and self-JoinMPI of the current segment (see line 11). Notably, both self-JoinMP and JoinMP computations can be implemented through any GPU-based algorithm. Indeed, we employ SCAMP [14], which currently is the fastest MP computing algorithm for GPU. Next, StreaMP starts to merge the segment and the time series read so far. In case the segment is the first one, it is used to initialize the time series, and the algorithm proceeds ahead of schedule to the beginning of the loop (see lines 12–15). The rest of the loop body basically follows the description we give above with Fig. 2. Notably, when MPs P_T and P_{TS} are merged, since the resulting MP is placed into P_T, we need to shift its MPI, I_T, where the offset is $|P_T|-m+1$ (see lines 17–19). Let us also mention that at the end, the current segment is appended to the time series read so far, excluding the overlapping points (see line 22). Finally, the loop body ends with top-1 motif and discord discovery, which is based on the built MP and MPI and thus costs $O(|P_T|)$.

Algorithm 1 StreaMP

Input data: segment length n, subsequence length m (both taken as a power of two and $n \gg m$)
Output data: not provided
Local data: streaming time series $T = \{T(k)\}_{k=1}^{\infty}$,
streaming MP and MPI $\{P_T, I_T\} = \{P_T(k), I_T(k)\}_{k=1}^{\infty}$,
current segment $S = \{S(k)\}_{k=1}^{n}$,
top-1 m-length motif and discord to be permanently found in stream $\{motif, discord\}$

$T \leftarrow \varnothing$; $P_T \leftarrow \varnothing$; $I_T \leftarrow \varnothing$ ▷ Initialize streaming data
▷ Process the streaming time series segment-by-segment until there is an interruption
while not *Interruption* **do**
  if $T = \varnothing$ **then**
    $S \leftarrow \varnothing$; $k \leftarrow 0$ ▷ Process the first segment
  else
    ▷ Process subsequent segments with overlap: a segment starts with the last subsequence of the time series read so far
    $S \leftarrow T_{n-m+2,\,m-1}$; $k \leftarrow m-1$
  end if
  while $k < n$ **do** ▷ Read sensor data into the segment
    $k \leftarrow k+1$; $S(k) \leftarrow$ ReadFromSensor()
  end while
  ▷ Merge the current segment and the time series read so far
  $\{P_S, I_S\} \leftarrow$ SelfJoinMP(S, m) ▷ Calculate self-JoinMP and self-JoinMPI of the current segment
  if $T = \varnothing$ **then** ▷ Merge the first segment with itself
    $T \leftarrow S$; $\{P_T, I_T\} \leftarrow \{P_S, I_S\}$
    continue
  end if
  ▷ Update MP and MPI of the time series read so far based on the current segment
  $\{P_{TS}, I_{TS}\} \leftarrow$ JoinMP$(T,\ S, m)$; $\{P_T, I_T\} \leftarrow$ MergeMPs$(P_T, I_T, P_{TS}, I_{TS})$
  ▷ Shift MPI values of the accumulated time series to prepare for concatenation with the current segmentвЂ™s MP
  for $k \leftarrow 1$ **to** $|P_T|$ **do**
    $I_T(k) \leftarrow I_T(k) + |P_T| - m + 1$
  end for
  ▷ Update MP and MPI of the current segment based on the time series read so far
  $\{P_{ST}, I_{ST}\} \leftarrow$ JoinMP(S, T, m); $\{P_S, I_S\} \leftarrow$ MergeMPs$(P_S, I_S, P_{ST}, I_{ST})$
  ▷ Concatenate the updated MP and MPI of the time series read so far with updated counterparts of the current segment
  $\{P_T, I_T\} \leftarrow \{P_T, I_T\} \cdot \{P_S, I_S\}$
  $T \leftarrow T \cdot S_{m-1,\,n-m+1}$ ▷ Concatenate the time series read so far with the current segment (excluding overlap)
  ▷ Discover top-1 motif and discord in the streaming time series based on the built MP and MPI
  $\{motif, discord\} \leftarrow$ DiscoverTop1MotifDiscord(T, P_T, I_T)
end while

Algorithm 2 MergeMPs

Input data: MP and MPI of the time series A and B $\{P_A, I_A\}$, $\{P_B, I_B\}$
Output data: P_A updated by the element-wise minima in A and B, and I_A updated accordingly

for $k \leftarrow 1$ **to** $|P_A|$ **do**
  if $P_B(k) < P_A(k)$ **then**
    $P_A(k) \leftarrow P_B(k)$; $I_A(k) \leftarrow I_B(k)$
  end if
end for
return $\{P_A, I_A\}$

3. EXPERIMENTAL EVALUATION

In the experiments, we evaluate our algorithm's efficiency in comparison with the state-of-the-art analog. We designed the experiments to be easily reproducible with our repository [15], which contains the algorithm's source code and supplemental data.

3.1. Experimental Setup

Baseline. Since our review of related work did not reveal any MP computation solution that is both GPU-oriented and stream-oriented, we compare StreaMP with SCAMP [14], the fastest state-of-the-art GPU-based MP algorithm, even though it is designed for batch rather than stream processing.

Table 1. Simulated sensors used in the experiments and their real-domain counterparts

Time series length, 2^x	Simulated sensor freq, KHz	Example of a real-domain sensor	
		freq, KHz	subject domain
16	240.9	240	Contactless measurement of respiration rate and respiratory signal characteristics [21]
17	228.5	230	Measurement of velocity, turbulence, and air flow profile; Investigation of sound generation and propagation in moving media [22]
18	209.2	210	Remote sensing of the seafloor; biological studies [23]
19	152.6	153	Acoustic assessment of biomass and distribution of key marine ecosystem components [24]
20	92.6	90	Detection of bearing defects and rotor imbalance at early stages; Monitoring of overloads and microstructural deformations in aircraft fuselages [25]
21	50.6	50	Attitude control of mini-satellites and UAVs under turbulent conditions; Tracking of surgical instrument micromovements [26]
22	26.6	25	High-frequency vibration monitoring; Recording and analysis of rapidly time-varying mechanical forces, pressures, vibrations, and deformations [27]
23	11.3	10	
24	2.5	2	Vibration monitoring; autonomous systems; resource-efficient solutions [28]
25	0.6	0.6	Monitoring of seismic wave propagation between wells in seismic exploration [29]

Simulation of stream processing. To simulate continuous MP computation on streaming data, we use a fixed-length time series, which can be entirely placed in RAM. We employ a set of synthetic time series generated through the random walk model [19]. In the set, each time series length is a power of two to provide the highest possible performance of the FFT that the MP computation algorithms rely on; the time series length gradually increases from 2^{16} (tens of thousands) to 2^{25} (tens of millions). We take the subsequence length 256, since in the experimental evaluation of time series mining algorithms, it is one of the most common values for such a parameter [20].

We simulate stream processing by implementing the ReadFromSensor subroutine (see line 9 in Alg. 1) with a deliberate delay, which plays the role of the sensor frequency. We calibrate the frequency based on the time series length used in the experiment, defining it as n/r_{last}, where n and r_{last} are, respectively, the segment length and the running time StreaMP processes the *last segment* of the input time series. This technique ensures minimal delay in streaming mode: the algorithm has enough time to process any segment before the next one arrives, even in the worst case, when computing JoinMPs is most time-consuming.

To confirm the relevance of the above-described simulation method, in Table 1, for each time series length involved in the experiments, we show the calibrating frequency calculated w.r.t. our hardware platform together with a real-domain example of streaming time series processing, which employs a sensor with the frequency close to calibrating one.

Hardware. In Table 2, we summarize the hardware platform of our experiments installed in the HPC center of the South Ural State University (Chelyabinsk, Russia) [30]. Notably, in the evaluation, we employ three configurations of the platform, where one, two, and four graphics processors are involved (referred below as 1×GPU, 2×GPU, and 4×GPU, respectively).

Table 2. Hardware platform of the experiments

Feature	GPU	CPU
Brand and product line	NVIDIA Tesla	Intel Xeon Gold
Model	V100 SXM2	6254
Number of cores	5120	18
Core frequency, GHz	1.455	3.1
RAM, Gb	32	64
Peak performance (double precision), TFLOPS	14.9	8

Metrics. In the experiments, we measure the performance of competitors, understanding it as follows. On one hand, despite the above-described approach to simulation imitating stream processing, we cannot employ metrics based on the running time over fixed-length time series, since the length of the real streaming time series is unlimited (infinite). Thus, the performance of StreaMP is defined as the running time to process the last segment of the fixed-length time series, which is the worst case when the algorithm merges the MPs of the current segment and the streaming time series accumulated at the moment. On the other hand, SCAMP is intended for batch, not stream processing. Thus, the performance of SCAMP is defined as the running time to process the entire fixed-length time series, which is the worst case when the algorithm computes the MP of the current segment appended to the streaming time series accumulated at the moment, performing this from scratch. To present the experimental results in a more convenient way, we additionally define the absolute speedup measure of our algorithm over its rival as a ratio of SCAMP and StreaMP performance.

3.2. Results and Discussion

Figure 3 depicts the experimental results for the 1×GPU configuration. StreaMP outperforms SCAMP except for time series of length 2^{16} (tens of thousands) and 2^{17} (hundreds of thousands). As expected, for short time series, our algorithm is slower than its competitor, since in such a case an overhead of CPU-GPU data transfer is greater than calculation time. As for the time series longer than hundreds of thousands, StreaMP surpasses SCAMP in performance by several times: from one and a half to 27 for time series lengths from hundreds of thousands to tens of millions.

The results demonstrate the fundamental difference between StreaMP and SCAMP algorithms when processing streaming time series. Thanks to its streaming nature, the performance of StreaMP depends only on the new segment length, not on the total length of the accumulated time series. SCAMP, in contrast, is designed for batch mode. In a streaming context, its processing time increases exponentially with increasing time series length, since when each new segment arrives, it must recompute the MP from scratch for the entire accumulated time series.

In Fig. 4, we show the absolute speedup of our algorithm on all the hardware configurations. As can be seen, StreaMP still outperforms SCAMP on two and four GPUs, although not as significantly as in the case of one GPU. Moreover, StreaMP scales poorly with respect to the number of GPUs. The most revealing cases are time series with tens of millions of lengths (i.e., 2^{24} and 2^{25}), where doubling the number of GPUs in the case of one and two GPUs allows for a performance gain of only about 5% and 3%, respectively. The reason is that SCAMP, which we use inside StreaMP to compute MPs, is not well suited for distributed processing of streaming time series. We leave the problem above for future work.

CONCLUSION

In this article, we addressed the topical problem of GPU-accelerated matrix profile (MP) computation for streaming time series. For the given time series and subsequence length, MP is

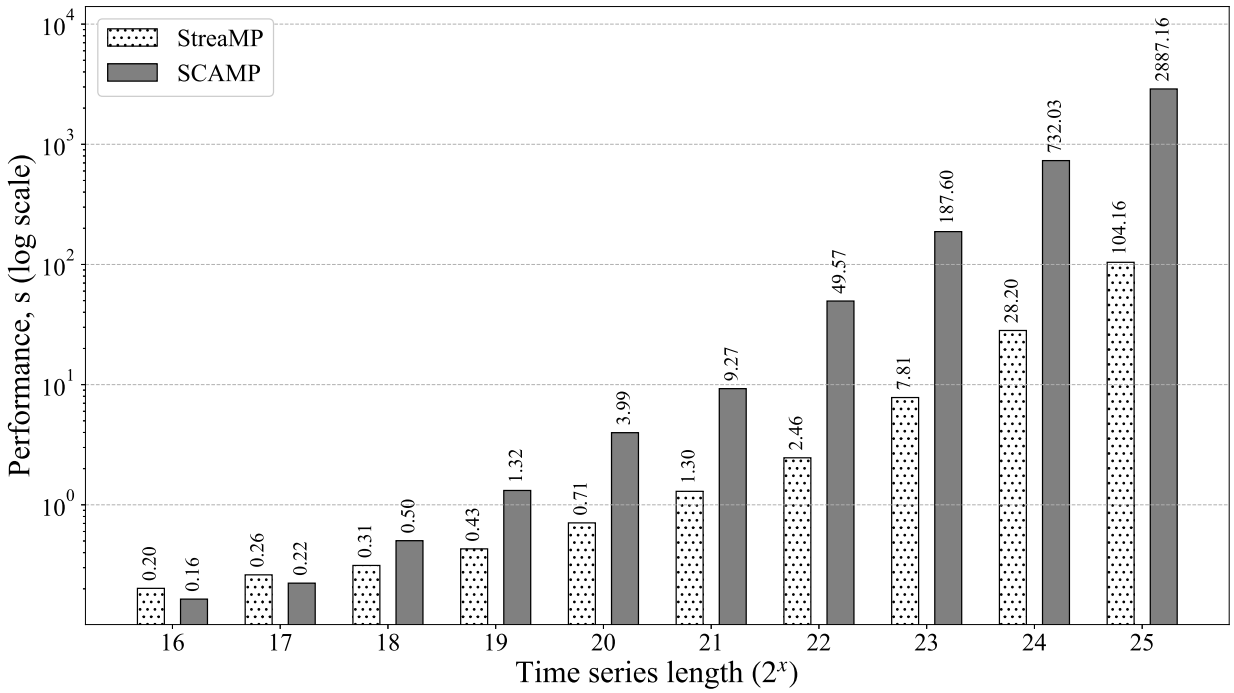


Figure 3. Performance of StreaMP and SCAMP on 1×GPU configuration

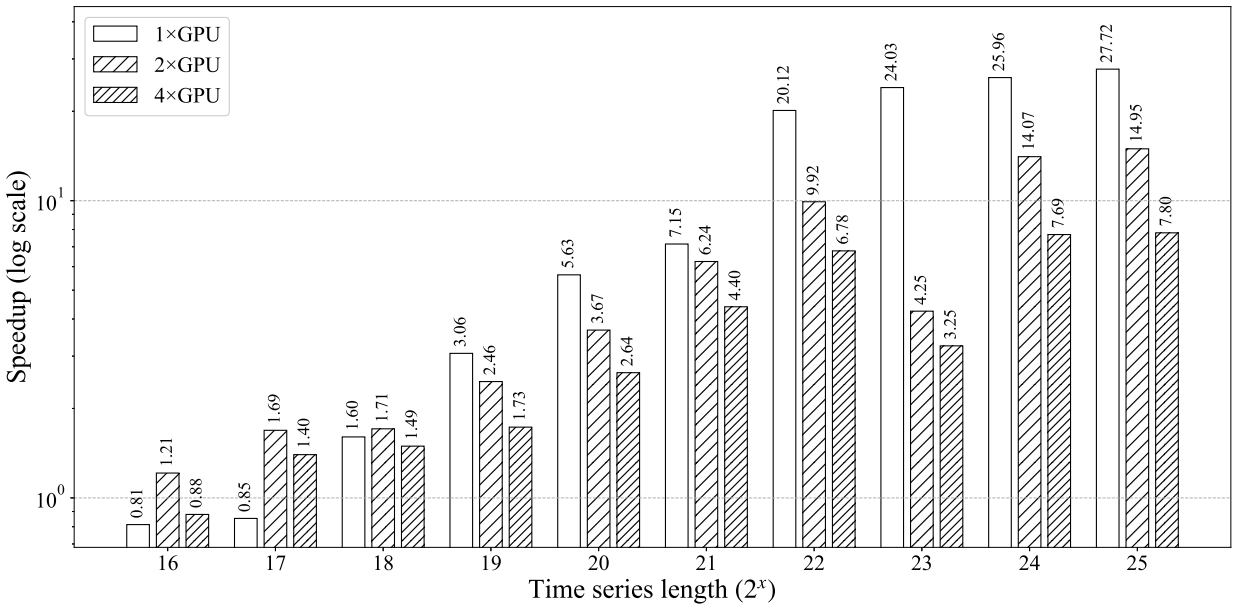


Figure 4. Speedup of StreaMP over SCAMP on 1×GPU, 2×GPU, and 4×GPU configurations

defined as a real-valued array, where the ith element is the distance from the ith subsequence to its nearest neighbor (i.e., to the most similar non-overlapping subsequence) [5]. The MP allows one to reveal numerous underlying patterns and anomalies: motifs [6], discords [7], snippets [8], chains [9], etc. However, among the wide range of MP computation algorithms (e.g., STAMP [5], GPU-STAMP [5], STOMP [5], GPU-STOMP [18], STOMPI [5], SCRIMP++ [13], and SCAMP [14]), none are both GPU-oriented and stream-oriented to the best of our knowledge.

In the article, we introduced a novel algorithm called StreaMP (Streaming Matrix Profile), which employs GPU to accelerate MP computation of streaming time series. StreaMP keeps in RAM the time series arriving from a sensor and processes incoming data in a segment-wise manner, where the segment length is the algorithm's parameter. While processing, we merge the MP of the time series accumulated at the moment with one of the current segment. To compute MPs, we employ a GPU-based algorithm, which we consider a hyperparameter of StreaMP. Next, the current segment is appended to the time series accumulated at the moment. In contrast, MP of the segment we computed is merged with MP of the time series read so far through our proposed multistep schema. StreaMP computes the JoinMP of the time series and the segment, and vice versa, and merges their resulting MPs with MPs of the time series and the segment, respectively. Notably, in all the steps above, avoid losing results at the junction of the time series and the segment, StreaMP overlaps them in a range that is one less than the subsequence length. In the end, the algorithm concatenates the modified MP of the time series and segment, placing the result into the former. The MP we built in this way makes it possible to discover motifs and discords in the streaming time series in linear time.

In the experimental evaluation, we employed SCAMP [14] as an underlying facility of StreaMP to compute MPs on GPU and compare our algorithm with SCAMP, since it is the fastest state-of-the-art GPU-based algorithm for batch mode. We simulated stream processing of a synthetic fixed-length time series through the deliberate delay, which imitates reading data from a sensor. The sensor frequency is calibrated to ensure the fact that StreaMP has enough time to process a segment before the next one arrives. In the experiments, we measure the performance of competitors, understanding it for SCAMP and StreaMP as the running time to process, respectively, the entire fixed-length time series and its last segment. Experimental results on a one-GPU configuration showed that, for time series longer than hundreds of thousands, StreaMP surpasses SCAMP in performance by several times: from one and a half to 27 for time series lengths from hundreds of thousands to tens of millions. However, for the shorter-length time series, our algorithm was predictably inferior to its opponent, since an overhead of CPU-GPU data transfer is greater than the calculation time. As for two- and four-GPU configurations, StreaMP still outperforms SCAMP (although not as significantly as in the case of one GPU) but scales poorly w.r.t. the number of GPUs since SCAMP we employ inside StreaMP to compute MPs is not well-suited for distributed processing of streaming time series.

We have made the StreaMP code and experimental data publicly available [15] so that the research community can confirm and extend our work. In further work, we plan to extend our approach, revisiting the problem of efficient distributed MP computing in streaming mode and considering the discovery of other MP-based data mining primitives in streaming time series.

ACKNOWLEDGMENTS

The research was carried out using the supercomputer resources of the South Ural State University (Chelyabinsk, Russia).

CONFLICT OF INTEREST

The authors declare that they have no conflict of interest.

REFERENCES

[1] 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

[2] S. Ivanov, K. Nikolskaya, G. Radchenko, L. Sokolinsky, and M. Zymbler, "Digital twin of city: Concept overview," in Proc. of 2020 Global Smart Industry Conf., GloSIC 2020, Chelyabinsk, Russia, November 17–19, 2020, pp. 178–186. https://doi.org/10.1109/GloSIC50886.2020.9267879

[3] M. Zymbler, Y. Kraeva, E. Latypova, S. Kumar, D. Shnayder, and A. Basalaev, "Cleaning sensor data in smart heating control system," in Proc. of 2020 Global Smart Industry Conf., GloSIC 2020, Chelyabinsk, Russia, November 17–19, 2020, pp. 375–381. https://doi.org/10.1109/GloSIC50886.2020.9267813

[4] I. Volkov, G. Radchenko, and A. Tchernykh "Digital twins, Internet of things and mobile medicine: A Review of current platforms to support smart healthcare," Program. Comput. Softw. **47** (8), 578–590 (2021). https://doi.org/10.1134/S0361768821080284

[5] 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

[6] A. Mueen, E. Keogh, Q. Zhu, S. Cash, and B. Westover, "Exact discovery of time series motifs," Proc. of the SIAM Int. Conf. on Data Mining, SDM 2009, April 30 – May 2, 2009, Sparks, Nevada, USA (SIAM), pp. 473–484. https://doi.org/10.1137/1.9781611972795.41

[7] E. J. Keogh, J. Lin, S. Lee, and H. V. Herle, "Finding the most unusual time series subsequence: algorithms and applications," Knowl. Inf. Syst. **11** (1), 1–27 (2007). https://doi.org/10.1007/s10115-006-0034-6

[8] S. Imani, F. Madrid, W. Ding, S. E. Crouter, and E. J. Keogh, "Introducing time series snippets: A new primitive for summarizing long time series," Data Min. Knowl. Discov. **34** (6), 1713–1743 (2020). https://doi.org/10.1007/s10618-020-00702-y

[9] Y. Zhu, M. Imamura, D. Nikovski, and E. J. Keogh, "Introducing time series chains: A new primitive for time series data mining," Knowl. Inf. Syst. **60** (2), 1135–1161 (2019). https://doi.org/10.1007/S10115-018-1224-8

[10] F. Nilsson, M. Bouguelia, and T.S. Rögnvaldsson, "Practical joint human-machine exploration of industrial time series using the matrix profile," Data Min. Knowl. Discov. **37** (1), 1–38 (2023). https://doi.org/10.1007/s10618-022-00871-y

[11] T. Uudeberg, J. Belikov, L. Päeske, H. Hinrikus, I. Liiv, and M. Bachmann, "In-phase matrix profile: A novel method for the detection of major depressive disorder," Biomed. Signal Process. Control. **88** (Part C), 105378 (2024). https://doi.org/10.1016/j.bspc.2023.105378

[12] N. S. Senobari, P. M. Shearer, G. J. Funning, Z. Zimmerman, Y. Zhu, P. Brisk, and E. Keogh, "The matrix profile in seismology: Template matching of everything with everything," JGR Solid Earth **129** (2), e2023JB027122 (2024). https://doi.org/10.1029/2023JB027122

[13] Y. Zhu, C. C. M. Yeh, Z. Zimmerman, K. Kamgar, and E. Keogh, "Matrix Profile XI: SCRIMP++: Time series motif discovery at interactive speeds," in Proc. of the IEEE Int. Conf. on Data Mining, ICDM 2018, Singapore, November 17–20, 2018, pp. 837–846. https://doi.org/10.1109/ICDM.2018.00099

[14] Z. Zimmerman, K. Kamgar, N. S. Senobari, et al., "Matrix profile XIV: Scaling time series motif discovery with GPUs to break a quintillion pairwise comparisons a day and beyond," in Proc. 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

[15] A. Valiulin and M. Zymbler, "StreaMP: GPU-accelerated algorithm to compute matrix profile of streaming time series," (2025). https://gitverse.ru/valiulinaa/StreaMP

[16] S. Zhong and A. Mueen, "MASS: distance profile of a query over a time series," Data Min. Knowl. Discov. **38** (3), 1466–1492 (2024). https://doi.org/10.1007/s10618-024-01005-2

[17] S. Zilberstein and S. Russell, "Approximate reasoning using anytime algorithms," in Imprecise and Approximate Computation, Ed. by S. Natarajan, pp. 43–62. Springer US (1995). https://doi.org/0.1007/978-0-585-26870-5_4

[18] Y. Zhu, Z. Zimmerman, N. Shakibay Senobari, et al., "Exploiting a novel algorithm and GPUs to break the ten quadrillion pairwise comparisons barrier for time series motifs and joins," Knowl. Inf. Syst. **54**, 203–236 (2018). https://doi.org/10.1007/s10115-017-1138-x

[19] K. Pearson, "The problem of the random walk," Nature **72**, 294 (1905). https://doi.org/10.1038/072294b0

[20] E. J. Keogh and C. A. Ratanamahatana, "Exact indexing of dynamic time warping," Knowl. Inf. Syst. **7** (3), 358–386 (2005). https://doi.org/10.1007/s10115-004-0154-9

[21] S. D. Min, J. K. Kim, H. S. Shin, Y. H. Yun, C. K. Lee, and M. Lee, "Noncontact respiration rate measurement system using an ultrasonic proximity sensor," IEEE Sensors J. **10** (11), 1732–1739 (2010). https://doi.org/10.1109/JSEN.2010.2044239

[22] X. Wang, D. Wang, and J. Yang, "Manufacture of piezoelectric sensor dedicated to airflow velocity detection," Sensors and Transducers **156** (9), 62 (2013).

[23] H. Dyvorne, C. Fermon, M. Pannetier-Lecoeur, H. Polovy, and A. L. Walliang, "NMR with superconducting-GMR mixed sensors," IEEE Trans. Appl. Supercond. **19** (3), 819–822 (2009). https://doi.org/10.1109/TASC.2009.2018767

[24] P. H. Ressler, "Acoustic backscatter measurements with a 153kHz ADCP in the northeastern Gulf of Mexico: determination of dominant zooplankton and micronekton scatterers," Deep Sea Res. Part I: Oceanogr. Res. Pap. **49** (11), 2035–2051 (2002). https://doi.org/10.1016/S0967-0637(02)00117-6

[25] C. Padovani, M. Bestetti, C. Valzasina, A. G. Bonfanti, and G. Langfelder, "Sub-10 μHz/$\sqrt{}$Hz Measurement instrumentation for 140-dB DR frequency-modulated MEMS Sensors," IEEE Trans. Instrum. Meas. **72**, 1–8 (2023). https://doi.org/10.1109/TIM.2022.3227996

[26] M. Gadola, F. Maspero, G. Langfelder, M. Sansa, T. Verdot, A. Berthelot, and P. Robert, "50-kHz MEMS gyroscopes based on NEMS sensing with 1.3 mdps/$\sqrt{}$Hz ARW and 0.5°/h stability," in Proc. of the 2020 IEEE Sensors Conf., Rotterdam, Netherlands, October 25-28, 2020, 1–4 (2020). https://doi.org/10.1109/SENSORS47125.2020.9278893

[27] Y. Zhang, X. Zhou, N. Zhang, et al., "Ultrafast piezocapacitive soft pressure sensors with over 10 kHz bandwidth via bonded microstructured interfaces," Nat. Commun. **15**, 3048 (2024). https://doi.org/10.1038/s41467-024-47408-z

[28] M. Bertocco, A. Flammini, D. Marioli, and A. Taroni, "Robust and accurate real-time estimation of sensors signal parameters by a DSP approach," IEEE Trans. Instrum. Meas. **49** (3), 685–689 (2000). https://doi.org/10.1109/19.850415

[29] Q. Liu, X. Qiao, Z. A. Jia, H. Fu, H. Gao, and D. Yu, "Large frequency range and high sensitivity fiber bragg grating accelerometer based on double diaphragms," IEEE Sensors J. **14** (5), 1499–1504 (2014). https://doi.org/10.1109/JSEN.2013.2296932

[30] N. Dolganina, E. Ivanova, R. Bilenko, and A. Rekachinsky, "HPC resources of South Ural State University," in Communications in Computer and Information Science, Proc. of the 16th Int. Conf. on Parallel Computational Technologies (PCT 2022), March 29–31, 2022, Dubna, Russia, Eds. L. Sokolinsky and M. Zymbler (Springer: Berlin/Heidelberg, Germany, 2022) **1618**, pp. 43–55. https://doi.org/10.1007/978-3-031-11623-0_4