

Received January 19, 2021, accepted January 22, 2021, date of publication January 26, 2021, date of current version February 4, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3054750

Fast Algorithm Based on Parallel Computing for Sample Entropy Calculation

XINZHENG DONG^{1,2}, CHANG CHEN³, QINGSHAN GENG⁴, WENSHENG ZHANG⁵,
AND XIAOHUA DOUGLAS ZHANG^{1,2,3}, (Senior Member, IEEE)

¹School of Software Engineering, South China University of Technology, Guangzhou 510006, China

²Zhuhai Laboratory of Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Zhuhai College, Jilin University, Zhuhai 519041, China

³CRDA, Faculty of Health Sciences, University of Macau, Taipa, Macau

⁴Guangdong General Hospital, Guangdong Academy of Medical Science, Guangzhou 510080, China

⁵Research Center of Precision Sensing and Control, Institute of Automation, Chinese Academy of Sciences, Beijing 100864, China

Corresponding author: Xiaohua Douglas Zhang (douglaszhang@um.edu.mo)

This work was supported in part by the Science and Technology Development Fund, Macau, under Grant 0004/2019/AFJ and Grant 0011/2019/AKP, and in part by the University of Macau under Grant FHS-CRDA-029-002-2017, Grant EF005/FHS-ZXH/2018/GSTIC, and Grant MYRG2018-00071-FHS.


ABSTRACT Sample entropy is a widely used method for assessing the irregularity of physiological signals, but it has a high computational complexity, which prevents its application for time-sensitive scenes. To improve the computational performance of sample entropy analysis for the continuous monitoring of clinical data, a fast algorithm based on OpenCL was proposed in this paper. OpenCL is an open standard supported by a majority of graphics processing unit (GPU) and operating systems. Based on this protocol, a fast-parallel algorithm, OpenCLSampEn, was proposed for sample entropy calculation. A series of 24-hour heartbeat data were used to verify the robustness of the algorithm. Experimental results showed that OpenCLSampEn exhibits great accelerating performance. With common parameters, this algorithm can reduce the execution time to 1/75 of the base algorithm when the signal length is larger than 60,000. OpenCLSampEn also exhibits robustness for different embedding dimensions, tolerance thresholds, scales and operating systems. In addition, an R package of the algorithm is provided in GitHub. We proposed a sample entropy fast algorithm based on OpenCL that exhibits significant improvement for the computation performance of sample entropy. The algorithm has broad utility in sample entropy when facing the challenge of future rapid growth in the quantity of continuous clinical and physiological signals.

INDEX TERMS Algorithm, fast computation, graphics processing unit, parallel computing, sample entropy.

I. INTRODUCTION

Since entropy has been applied to the field of informatics [1], this measure of time series complexity has been continuously developed and improved. It is widely used in various fields such as astronomy [2], economics [3], and biology [4]. Over the past 20 years, the two most commonly used types of entropy, approximate entropy [5] and sample entropy [6], have been used for the measurement of nonlinear complexity in biological signals.

Since Kolmogorov-Sinai entropy [7], a theoretical metric used in nonlinear dynamic systems, is computationally difficult and not easily promulgated, Pincus [5] proposed

The associate editor coordinating the review of this manuscript and approving it for publication was Yilun Shang .

approximate entropy to measure the complexity and unpredictability of systems. Richman and Moorman [6] proposed sample entropy, which improved the shortcoming of sequence self-matching in approximate entropy. Multiscale entropy extends sample entropy to multiple time scales or signal resolutions to provide an additional perspective when the time scale of relevance is unknown. Furthermore, Costa *et al.* [8] applied multiscale entropy to physiological time series and clinical areas. His research showed that the entropy of healthy people is higher than that of cardiac autonomic neurosis patients. To date, as one of the nonlinear features used to evaluate sympathetic and parasympathetic cardiac functions, multiscale entropy has been used to study blood glucose data [9], electrocardiogram (ECG) data [4], airflow data [10], [11], electroencephalogram (EEG) data [12] and

so on. Researchers have also performed disease diagnosis and prognosis and assessed patient status based on the level of sample entropy.

Relative to Kolmogorov-Sinai entropy, the sample entropy algorithm has simplified the theoretical derivation and calculation steps, but it still includes the step of checking the similarity between vector pairs, which is the most time-consuming part of the entropy algorithm. Execution time increases quadratically with increasing series length N . To improve computational efficiency, many researchers have modified the sample entropy algorithm by combining existing methods. For example, Lee [13] combined the K -dimensional tree method and sample entropy, which can shorten the execution time. Manis [14] edged the bucket-assisted algorithm to sample entropy. In addition, they proposed a lightweight algorithm when the embedding dimension is small [15]. These methods sort the vector pairs, which can reduce the number of similarity checks and thus reduce the execution time. They advanced the running speed of sample entropy, but the effect on physiological data with a large data length is not obvious. We know that clinical continuous data are often characterized by large data volumes and many observational indicators. A Holter monitor, for example, is used to monitor 24 hours of continuous cardiac data in heart patients, and EEG signals in epileptic patients typically consist of 32 or 196 channels. How to quickly and easily perform sample entropy calculations on large amounts of clinical data is addressed in this paper.

Graphics processing unit (GPU) was originally designed for performing graphics-related tasks. Because of the emergence of general-purpose GPU (GPGPU) programming and its enormous computing power, GPU is increasingly used in various general-purpose applications [16]. In three areas of medical image processing—image reconstruction [17]–[23], image registration [24]–[26] and image segmentation [27]–[30]—many computing applications have exploited the capabilities of GPU to reduce execution times. Thus, the use of GPU to process biomedical signals has become a topic of interest. For example, Martínez-Zarzuela *et al.* [31] proposed a GPU-based implementation of cross-approximate entropy and achieved two orders of magnitude acceleration in analyzing magnetoencephalography (MEG) recordings. Konstantinidis *et al.* [32] used GPU parallel capabilities to offer a feasible solution for the real-time sensing of a user's affective state during emotion-aware computing. Eklund *et al.* [33] described how to perform statistical analysis for functional magnetic resonance imaging (fMRI) data on a GPU and showed that nonparametric tests of fMRI data based on a GPU are possible. Currently, personal computer (PC) is usually paired with a GPU device. This allows for running biological computations on a PC in a high-performance computing (HPC) environment. Shen *et al.* [34] introduced a parallel implementation of a computer simulation for ECG on a PC with a GPU and demonstrated that the setup not only meets calculation requirements but also avoids the use of

expensive supercomputers. Zhu and Wei [35] demonstrated the advantages of GPU for the simulation of action potential propagation in cardiac tissue.

There are three popular implementation frameworks for GPGPU programming: CUDA from Nvidia (<https://developer.nvidia.com/cuda>), DirectCompute from Microsoft (<https://developer.nvidia.com/directcompute>) and OpenCL [36] from the Khronos Group. CUDA is only available for NVIDIA's GPU. DirectCompute is specific to Microsoft Windows. Therefore, they are not portable among different operating systems. OpenCL is an open standard supported by major GPU device vendors and operating systems [37]. Taking advantage of the parallel capability of a GPU and OpenCL's portability across GPU and operating systems, we aim to develop an OpenCL-based parallel algorithm for calculating sample entropy.

Multi-core CPUs are typically composed of a small number of high-frequency processor cores, while GPUs are composed of hundreds of processing units running at a low-frequency. Such massively parallel architecture brings high computing performance. OpenCL provides hardware abstractions and programming interfaces to perform task-parallel or data-parallel computations in a heterogeneous computing environment consisting of the host CPU and any attached OpenCL devices. An OpenCL device is divided into one or more compute units which are further divided into one or more processing elements. Computations occur within the processing elements. The processing elements in one compute unit execute a single stream of instructions or each processing element maintains its own program counter [36].

The contribution of the paper can be summarized in the following points. First, we brought parallel computing based on GPU for the improvement of computational performance of sample entropy. Based on the principle of parallel computing, we proposed a new method to quickly calculate sample entropy based on the OpenCL framework. Second, we compared the speedup performances among our new method and several existing methods in different embedding dimensions, tolerances, scales and operating systems. Finally, we provide a C++ implementation and an R package for our method, which can help researchers quickly calculate sample entropy for clinical data with a large data size.

II. METHODS

A. SAMPLE ENTROPY

Let $X = \{x_1, x_2, \dots, x_N\}$ represent a time series of length N . The template vector $X_m(i)$ of length m is constructed from X :

$$X_m(i) = \{x_i, x_{i+1}, x_{i+2}, \dots, x_{i+m-1}\} \quad (1)$$

and the distance function between two such vectors is defined:

$$d(m, i, j) = d[X_m(i), X_m(j)] \\ = \max_k \{|x_{i+k-1} - x_{j+k-1}|\}, \text{ where } k = 1, \dots, m. \quad (2)$$

Then, the number of similar vectors within a distance threshold r from $X_m(i)$ can be defined as

$$B_i^m(r) = \sum_{j=1, j \neq i}^{N-m} H(m, i, j, r) \quad (3)$$

and the number of vectors similar to $X_{m+1}(i)$ can be defined as

$$A_i^m(r) = \sum_{j=1, j \neq i}^{N-m} H(m+1, i, j, r) \quad (4)$$

where $H(m, i, j, r)$ is defined as

$$H(m, i, j, r) = \begin{cases} 1, & d(m, i, j) \leq r \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

Finally, we define the sample entropy as:

$$\begin{aligned} & \text{SampEn}(m, r, N) \\ &= \begin{cases} -\ln \frac{A^m(r)}{B^m(r)}, & A^m(r) \neq 0 \text{ and } B^m(r) \neq 0 \\ -\ln \frac{1}{(N-m)(N-m-1)}, & \text{otherwise} \end{cases} \end{aligned} \quad (6)$$

where the probability that two vectors of length m are similar is defined as

$$B^m(r) = \frac{1}{N-m} \sum_{i=1}^{N-m} \frac{1}{N-m-1} B_i^m(r) \quad (7)$$

and the probability that two vectors of length $m+1$ are similar as

$$A^m(r) = \frac{1}{N-m} \sum_{i=1}^{N-m} \frac{1}{N-m-1} A_i^m(r) \quad (8)$$

To improve the calculation efficiency by eliminating repeated comparisons, we define the forward $B_i^m(r)$ and $A_i^m(r)$:

$$B_{i,forward}^m(r) = \sum_{j=i+1}^{N-m} H(m, i, j, r) \quad (9)$$

$$A_{i,forward}^m(r) = \sum_{j=i+1}^{N-m} H(m+1, i, j, r) \quad (10)$$

and the total number of forward similar vectors of length m and $m+1$:

$$B_{forward} = \sum_{i=1}^{N-m} B_{i,forward}^m(r) \quad (11)$$

$$A_{forward} = \sum_{i=1}^{N-m} A_{i,forward}^m(r) \quad (12)$$

Note that $B_{forward} = \frac{(N-m)(N-m-1)}{2} B^m(r)$ and $A_{forward} = \frac{(N-m)(N-m-1)}{2} A^m(r)$; therefore, the sample entropy can be calculated as follows when $B_{forward} \neq 0$ and $A_{forward} \neq 0$:

$$\text{SampEn}(m, r, N) = -\ln \frac{A^m(r)}{B^m(r)} = -\ln \frac{A_{forward}}{B_{forward}} \quad (13)$$

B. THE OPENCL-BASED PARALLEL ALGORITHM

OpenCL (Open Computing Language) is an open standard for general-purpose parallel programming across heterogeneous processing platforms such as Central Processing Unit (CPU), GPU and other processors [36]. An OpenCL program has two parts: kernels that execute on OpenCL devices and a host program that executes on the host. The host program defines the context for the kernels and manages their execution. A kernel is a function executed on an OpenCL device. Hundreds of processing elements within one GPU can simultaneously execute one kernel with different data. It means that we can use different data for the same function calculation at the same time. Our algorithm combines such a data parallel programming model and traditional sample entropy algorithm to improve computing performance. Because it uses the OpenCL standard to perform parallel computing, we call it OpenCLSampEn.

From the definition of sample entropy, we can know that the straightforward sample entropy algorithm can be divided into three steps: inputting data, checking the similarity between vector pairs, and calculating logarithm. The time complexity of these three steps is linear, quadratic, and constant respectively. Obviously, checking the similarity between vector pairs is the main time-consuming execution in sample entropy calculation. Therefore, the acceleration algorithm of sample entropy is reflected in improving the efficiency of checking similarity. In our method, we will make the similarity comparison as parallel as possible. Specifically, this process can be divided into the following four steps, which are shown in Fig. 1.

1. Construct an array of size $N-m$, which holds all the template vectors of length $m+1$ from the input time series with N points.

2. Sort the array by the first element of each template vector. The sorted array can reduce unnecessary comparisons between template vector pairs in the next step. After that, all elements of the sorted array are unfolded into a point array one by one. Thus, a new time series with $(N-m) \times (m+1)$ points is constructed.

3. Calculate $A_{i,forward}^m(r)$ and $B_{i,forward}^m(r)$, where i ranges from 1 to $N-m$. Different from other steps running on the CPU, this step runs on the GPU. $A_{i,forward}^m(r)$ and $B_{i,forward}^m(r)$ can be calculated together in one single kernel. A single kernel execution can run on all the processing elements of the GPU device in parallel, so $A_{i,forward}^m(r)$ and $B_{i,forward}^m(r)$ can be computed in parallel for different i , which will have a considerable acceleration effect. A vector array of length $N-m$ is used to store the result, and each vector consisting of two elements $A_{i,forward}^m(r)$ and $B_{i,forward}^m(r)$ is placed at the i -th element of the vector array.

4. Calculate $A_{forward}$ and $B_{forward}$, then obtain the result $\text{SampEn}(m, r, N)$. Both $A_{forward}$ and $B_{forward}$ can be obtained by accumulating each $A_{i,forward}^m(r)$ and $B_{i,forward}^m(r)$, respectively, in the vector array. Then, we can obtain the final result $\text{SampEn}(m, r, N)$ by calculating the negative logarithm of the quotient of $A_{forward}$ divided by $B_{forward}$.

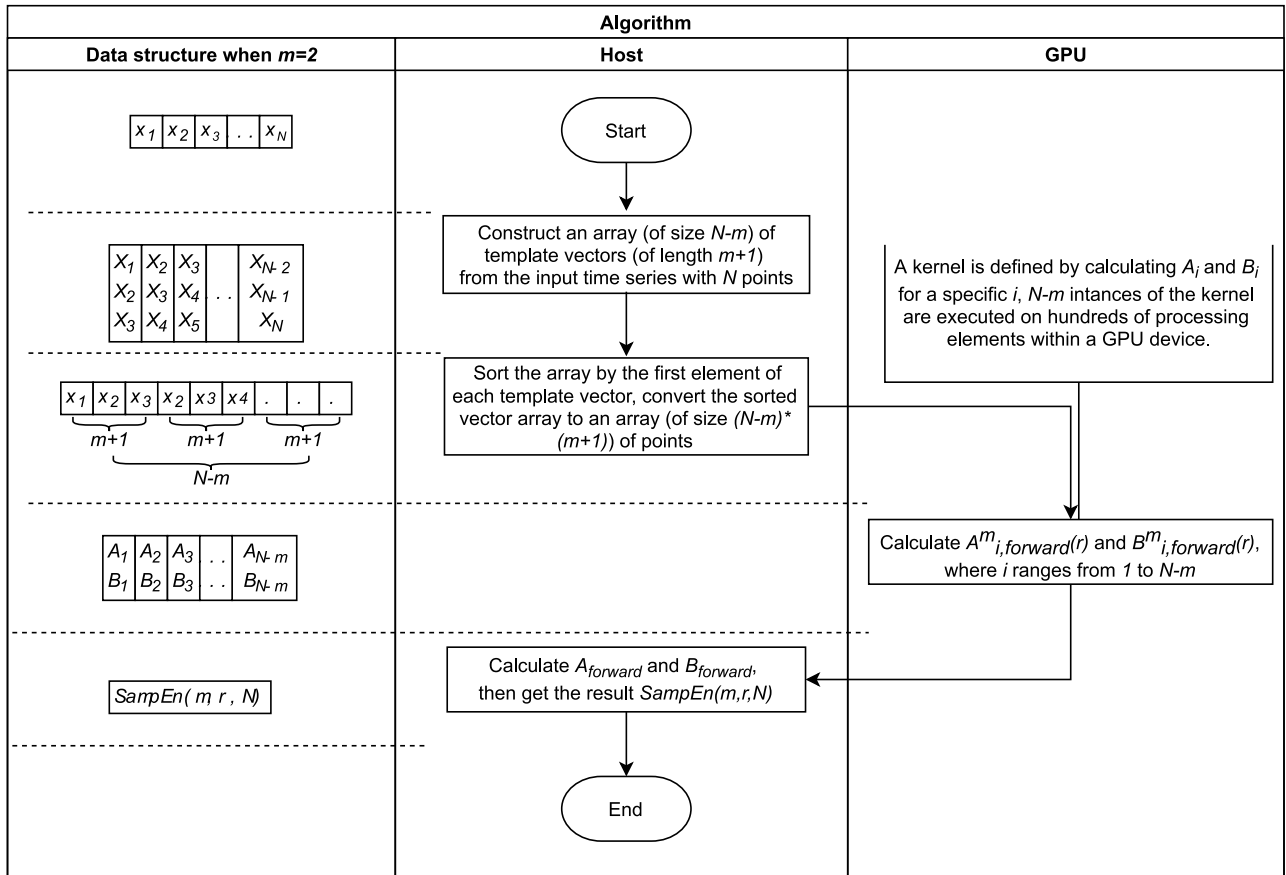


FIGURE 1. Flow chart of the main process of the OpenCL-based parallel algorithm.

In the first step, the array of template vectors is built at the storage cost of three times the volume of the original signal length. In the second step, the average time cost of sorting is $O(N \times \log_2 N)$. However, the combination of these two steps provides the basis for the third step so that we can not only eliminate unnecessary comparisons between vectors but also take advantage of the locality of concurrent accesses [38]. The execution time of the third step, which accounts for the majority of all steps, will be greatly reduced, thus improving the overall computing performance.

In OpenCL, when a kernel is submitted for execution by the host, an index space is defined. An instance of the kernel executes for each point in the index space. This kernel instance is called a work-item. Each work-item executes the same code, but the data can be different in each work-item. Several work-items compose a work-group. The work-items in a given work-group execute concurrently on the processing elements of a single compute unit. The work-items executing a kernel have the access to four distinct memory regions: global memory, constant memory, local memory and private memory. A local memory region is local to a work-group, and a private memory region is private to a work-item [36].

In our algorithm, the third step is defined as a kernel running on an OpenCL device, and other steps are contained

in a host program. The host program defines the context for the kernel and manages its execution. The input parameter of the kernel is stored in the constant global memory region, the output parameter is stored in the global memory region, and other internal variables of the kernel are private to work-items. The input data of the kernel is organized in a one-dimensional sequence. Besides this, the local work size of the kernel is set to 128, an empirical optimal value, which is selected by repeated experiments and comparisons. Single precision is a basic feature, while double precision is an optional extension feature in OpenCL devices, so we choose single precision for the portability of kernel across a variety of OpenCL device. We developed the algorithm in the C++ programming language and encapsulated it into an R package for convenience. The R package has been tested on different platforms: Linux, Windows, MacOS. It is available on GitHub: <https://github.com/dongxinzheng/ParallelSampEn>.

C. THE BASICSAMPEN, KDTREESAMPEN AND LIGHTWEIGHTSAMPEN ALGORITHMS

To verify the performance of the proposed algorithm, three algorithms are used for comparison with our algorithm: BasicSampEn, KDTreeSampEn and LightWeightSampEn. We use the C code of M. Costa, which is available

TABLE 1. Characteristics of the signals filtered from datasets.

ID	Mean	Standard deviation (SD)	Number of points	Sample entropy
1	0.734	0.221	116766	0.236
2	0.700	0.117	122819	0.357
3	0.614	0.103	136483	0.673
4	0.658	0.099	128744	1.025
5	0.688	0.162	116284	0.304
6	0.720	0.106	117709	0.539
7	0.720	0.119	117655	0.518
8	0.678	0.122	126848	0.502
9	0.689	0.151	118697	0.469
10	0.690	0.142	119859	0.655

TABLE 2. Configurations for test platforms.

Platform	Operating system version	GPU	CPU	Memory capacity
macOS	High Sierra	Intel Iris Plus	2.3 GHz Intel Core i5	16 GB
Windows	Windows 10 Home Edition	NVIDIA GeForce MX150	2.0 GHz Intel Core i7	8 GB
Linux	CentOS 7	NVIDIA Tesla V100 PCIe 16 GB	2.4GHz Intel® Xeon® Gold 6148 Processor	128 GB

in PhysioNet [8], [39], [40], as the basic algorithm: BasicSampEn is used to calculate multiscale sample entropy of clinical data. A fast algorithm proposed by Yu-Hsiang Lee [13] is called KDTreeSampEn. In the calculation of a matching part of the vector $X_m(i)$, this method constructs a high-dimensional binary tree model, the K-dimensional tree, and gives the tolerance r to determine the range of the nearest neighbor search. It sorts the original sequence in m -dimensional space and divides the space into several small parts, which can reduce the number of comparisons between vectors and quickly determine $A_i^m(r)$ and $B_i^m(r)$. A lightweight fast algorithm proposed by Manis, G. et al. in 2018 [15] is faster for a short time series and small m parameter. It is implemented to perform the comparison and called LightWeightSampEn in the following. The algorithm first sorts all vectors and finds vectors that meet the requirements and counts them by dichotomy. It also reduces the number of comparisons by sorting the vectors, thereby reducing the running time of the program.

D. EXPERIMENTAL DATASETS

The public clinical dataset *nsr2db* (Normal Sinus Rhythm RR Interval Database) from PhysioNet [39] was considered for evaluation of the four methods in our comparisons. It includes beat annotation files for 54 long-term ECG recordings of subjects in normal sinus rhythm (30 men, aged 28.5 to 76, and 24 women, aged 58 to 73). We obtained the RR interval files from the annotation files using the “ann2rr” command in the WFDB (WaveForm DataBase) package from PhysioNet and then removed the possible outliers that were not in this range [0.3, 1.5] in each file. Finally, ten files with more than 100,000 lines were randomly selected, as 100,000 is the

maximum number of points from one time series used in our comparisons. Table 1 shows the characteristics of the signals taken for evaluation of the four methods.

III. RESULTS

All four algorithms were tested on three different platforms: macOS, Windows, and Linux. Each platform represents a combination of an operating system and specific hardware resources. Table 2 shows the configurations of each test platforms. The macOS results was showed from part A to part D in this Section, while corresponding results from Windows and Linux were placed in supplementary materials. Then, the comparison of the three platforms under typical parameter values was shown in part E.

All algorithms were implemented in the C++ programming language within an executable file and run in the R language (version 3.6.1) environment for the following comparisons. Ten physiological signals were used in each comparison, and each signal was repeated 10 times. Thus, the results are shown as the average execution time and standard deviation of 100 runs. To exclude overheads from the running time, the data were read from the file into memory before timing. In addition, the creation time of the GPU context, which is a constant time connecting to the GPU, was also excluded. We designated the ‘*gettimeofday*’ function as the timing function for the macOS and Linux operating systems and designated the ‘*QueryPerformanceCounter*’ function as the timing function for the Windows operating system.

A. SPEEDUP PERFORMANCE FOR TYPICAL PARAMETER VALUES

We first checked the execution times of the four algorithms for the common condition: embedding dimension

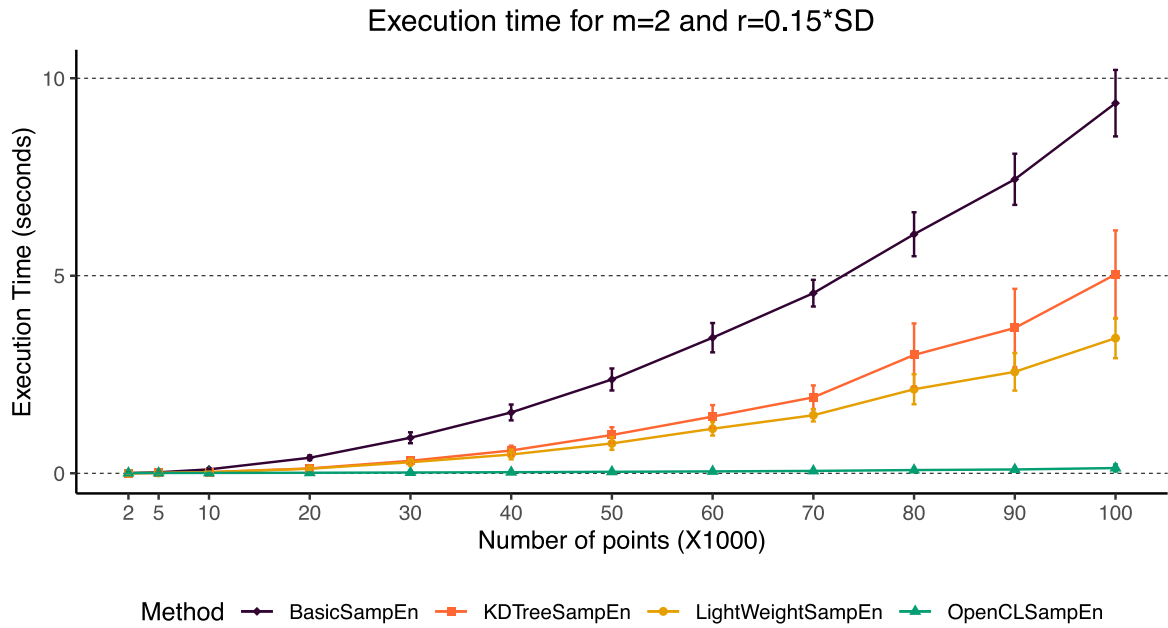


FIGURE 2. Execution time of the four algorithms on time series of different lengths for typical parameter values $m = 2$ and $r = 0.15 * SD$. Values are given as the means \pm standard deviation.

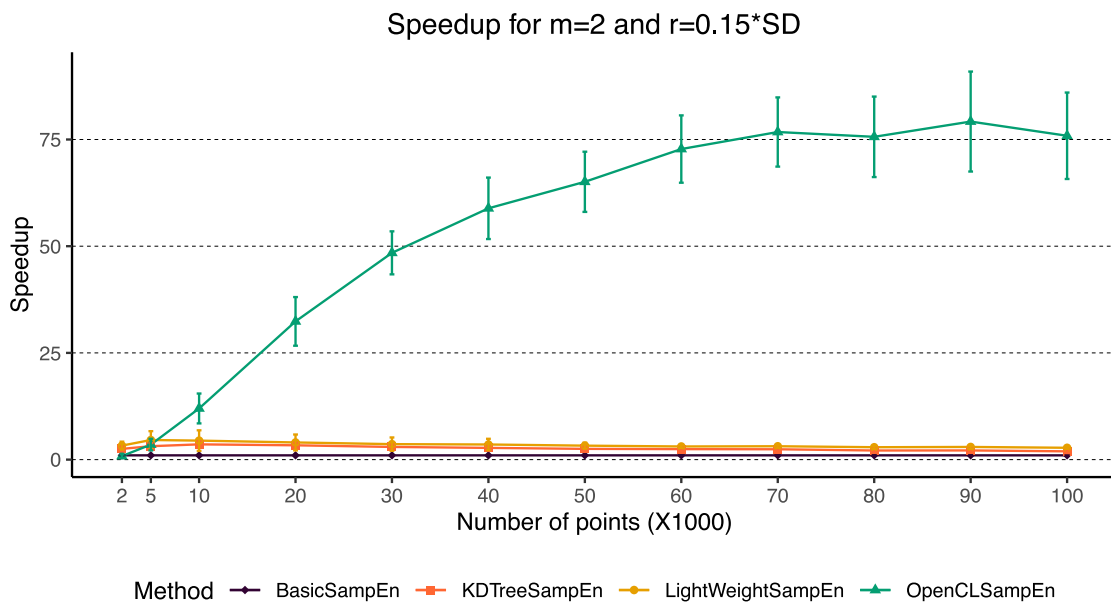


FIGURE 3. Speedup of the four algorithms gained from BasicSampEn on time series of different lengths for typical parameter values $m = 2$ and $r = 0.15 * SD$. Values are given as the means \pm standard deviation.

$m = 2$ and tolerance $r = 0.15 * SD$, where SD means the standard deviation of a physiological signal. Fig. 2 shows the execution time of all algorithms for different signal lengths. Compared with BasicSampEn, three acceleration algorithms all have a significant decrease in execution time. Compared to the other two acceleration algorithms, OpenCLSampEn algorithm has the best performance and the shortest execution time. Furthermore, OpenCLSampEn’s execution time does not raise with signal length

as rapidly as that of BasicSampEn, KDTreeSampEn and LeightWeightSampEn.

In order to clearly show the behavior of the four algorithms when the signal length is small, a metric named “speedup”, the quotient of execution time for each algorithm and basic algorithm, was used to evaluate the computing performance. In Fig. 3, it can be clearly seen that BasicSampEn is the most time-consuming of all algorithms. As a result, the speedup of BasicSampEn is a constant 1, and that of the

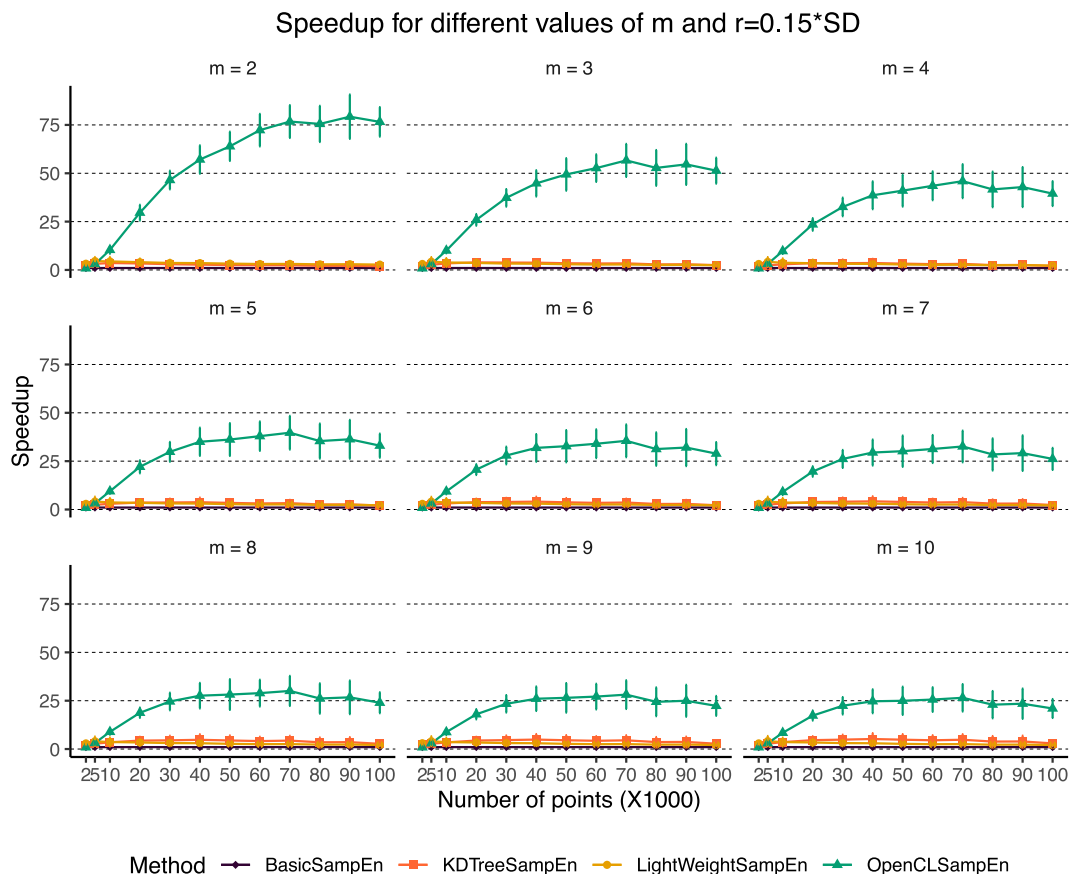


FIGURE 4. Speedup of the four algorithms gained from BasicSampEn on time series of different lengths for different values of m and $r = 0.15 * SD$. Values are given as the means \pm standard deviation.

other three algorithms is greater than 1, which means that they are all faster than BasicSampEn. When the length of the physiological signal is smaller than 5,000, these three accelerated algorithms have a similar growth trend that is slightly faster than the basic algorithm. With the increase of the data size, the speedup of OpenCLSampEn increases; its maximum is 75 compared with than BasicSampEn when the length is larger than 60,000. For a sample entropy analysis of clinical data, such as respiratory signals, 24-hour heartbeat signals and EEG signals, our method can quickly calculate the results. However, for all lengths, the speedups of KDTreeSampEn and LeightWeightSampEn are always less than 5. For physiological signals with a large amount of data, KDTreeSampEn and LeightWeightSampEn have limited computing capacity and cannot greatly improve the computing performance. It is worth noting that the large standard deviation of OpenCLSampEn’s speedup is derived from the basic algorithm, not the OpenCLSampEn algorithm itself.

B. SPEEDUP PERFORMANCE FOR DIFFERENT EMBEDDING DIMENSION M

From the last section, we know that when $m = 2$ and $r = 0.15 * SD$, OpenCLSampEn offers the highest speedup performance for a physiological signal with a data length

greater than 10,000. However, what happens when the values of parameters m or r change? We fixed the tolerance r to $0.15 * SD$ and changed the embedding dimension m from 2 to 10; the results are shown in Fig. 4. OpenCLSampEn still has a much higher speedup than the other two algorithms, but the speedup of OpenCLSampEn decreases with the increase of m , from a high of approximately 75 to a low of approximately 25. This is because when the embedding dimension increases, similarity calculations between vectors require more elements for vector comparison. The speedup of LightWeightSampEn has the same change trend as the change in embedding dimension m but is different from the trend of KDTreeSampEn, although this cannot be seen clearly in Fig. 4 because the change in value is too small. This is because the KDTreeSampEn algorithm has a greater advantage in high-dimensional range queries, although the improvement is negligible compared to OpenCLSampEn.

C. SPEEDUP PERFORMANCE FOR DIFFERENT TOLERANCE R

Fig. 5 shows the comparison of the acceleration performance of the four algorithms when tolerance r was changed from $0.05 * SD$ to $0.85 * SD$ and the embedding dimension m was fixed at 2. Similar to Fig. 4, OpenCLSampEn still has a much

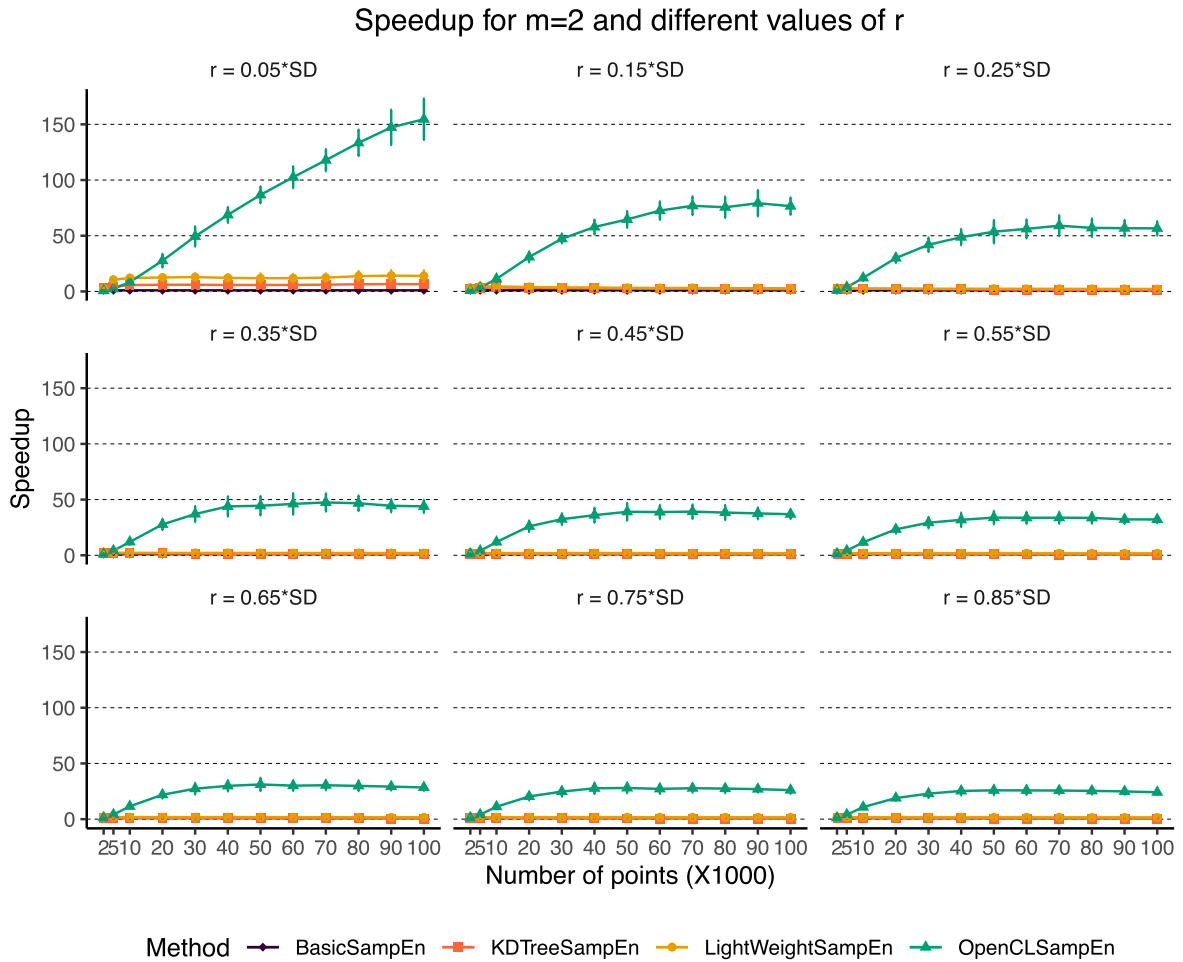


FIGURE 5. Speedup of the four algorithms gained from BasicSampEn on time series of different lengths for different values of r and $m = 2$. Values are given as the means \pm standard deviation.

higher speedup than the other two algorithms. The speedup of OpenCLSampEn decreases with the increase of r , from the high of approximately 150 to a low of approximately 30. The reason is that when tolerance r increases, the total number of compared vector pairs will increase, which requires more elements for vector comparison. The variation trend of the other two methods is consistent with that of openCLSampEn, although they cannot be seen clearly in Fig. 5 because the change in trend is too small.

D. SPEEDUP PERFORMANCE FOR MULTISCALE ENTROPY

Fig. 6 depicts the ensemble speedup of different algorithms for multiscale entropy on physiological signals with different lengths. Typical parameter embedding dimensions $m = 2$ and tolerance $r = 0.15 * SD$ were used. The accumulated execution times of multiple scales from 1 to 20 were used to calculate the ensemble speedup. Yentes et al. [41] found that at least 2,000 points could obtain stable sample entropy values. Thus, the length of the signal would start from 40,000, which still has 2,000 points after a coarse-graining process when

the scale factor increases to 20. From Fig. 6, the speedup of OpenCLSampEn increases from 10 to 40 with increasing data length, but the speedups of the other two algorithms decrease with increasing length, and the upper limit is always below 4. Therefore, when the OpenCLSampEn algorithm is applied to the analysis of multiscale entropy, it has a large computing performance advantage. Compared with Fig. 3, the highest speedup of the OpenCLSampEn algorithm is reduced from 76 to 40 when the data length reaches 100,000. The reason is that the accumulated time is the sum of execution times of 20 different time scales, while the speedup of the short signal is significantly smaller than that of the long signal.

E. SPEEDUP PERFORMANCE ON DIFFERENT PLATFORMS

In order to verify the portability of the proposed algorithm across GPU devices and operating systems, three platforms were used for testing (Table 2). The results for $m = 2$ and $r = 0.15 * SD$ are shown in Fig. 7. The top panels of Fig. 7 show that although the speedups of each algorithm on different operating systems are different, they have consistent

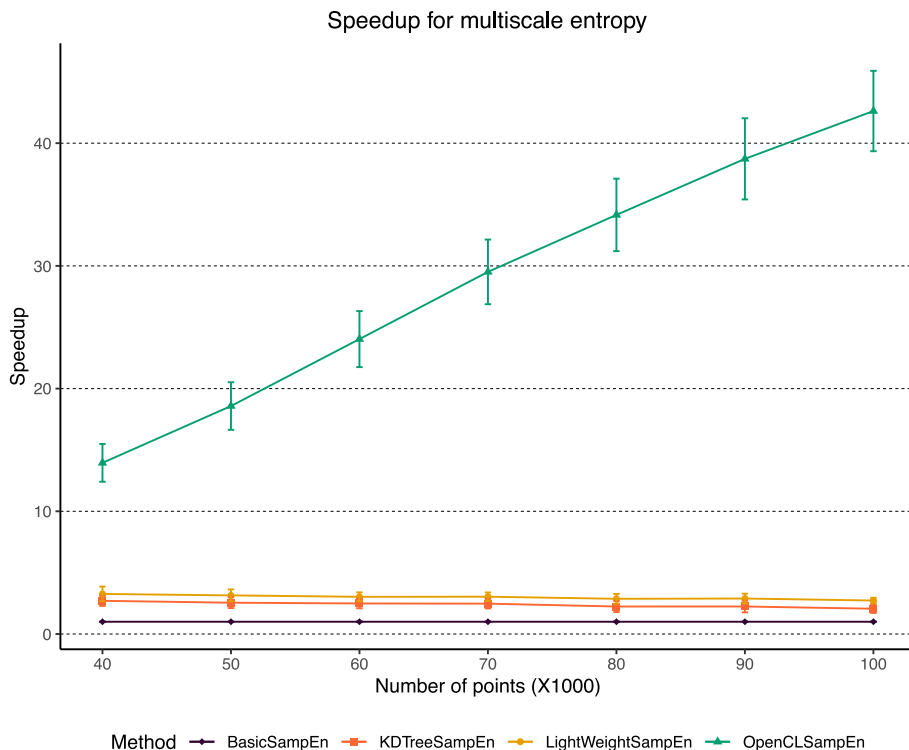


FIGURE 6. The ensemble speedup of the four algorithms for multiscale entropy on time series of different lengths. Typical parameter values $m = 2$ and $r = 0.15 * SD$ are used. The sum of multiple time scale execution times from 1 to 20 is used to calculate the ensemble speedup. Values are given as the means \pm standard deviation.

growth, trends with increasing signal length. The speedup of OpenCLSampEn exceeds the other two algorithms when the signal length is greater than a certain threshold. The threshold is 10,000 for the macOS and Windows operating systems and 20,000 for the Linux operating system, which can be seen in the bottom panels of Fig. 7. In addition, the speedup results of the Windows and Linux operating systems for different embedding dimensions m , tolerances r , and multiscale entropies are consistent with those of MacOS, which are available in the supplementary materials.

IV. DISCUSSION

Sample entropy is a widely used method for assessing the irregularity of physiological signals, but it has a high computational complexity, which prevents its application for time-sensitive scenes. To improve the computational performance of sample entropy analysis for the continuous monitoring of clinical data, a fast algorithm based on OpenCL was proposed in this paper. Considering the popularity of graphics cards in PCs and high-performance computers, a parallel computing model is employed in this algorithm. For implementation, the OpenCL GPGPU framework was adopted because it is an open industry standard that offers portability across GPU devices and operating systems.

Compared with the basic sample entropy algorithm and the other two algorithms reported in the previous literature, we found that our improved algorithm has the greatest

acceleration effect for clinical data, not only for commonly used parameter values $m = 2$ and $r = 0.15 * SD$ but also for different embedding dimensions m and different tolerances r . The effective design of the OpenCLSampEn algorithm and the adequate utilization of computing resources are the reasons for the results. Beyond that, the speedup of our proposed algorithm increases with increasing signal length, which is applicable to the analysis of physiological signals with large data sizes and multichannel signals. As a comparison, the speedup of other algorithms remains at a stable low level when the signal length increases. Our algorithm also has a good acceleration effect in different operating systems (Windows, Linux, macOS) and GPU devices. In addition, our algorithm can be used to improve the computational efficiency of the proposed entropies based on sample entropy, such as the multiscale sample entropy (MSE), refined MSE (RMSE) [42], composite multiscale entropy (CMSE) [43], refined CMSE (RCMSE) [44], etc. To make it easier for beginners to understand and use the algorithm, we provided an R package of this algorithm in GitHub as a convenience for researchers to calculate the sample entropy for large amounts of clinical data.

The proposed algorithm still has some small limitations. One limitation is that when the length is less than a certain threshold (5,000 for PCs), the speedup of our proposed algorithm does not provide significant improvement compared to the other two algorithms. This is because in this algorithm,

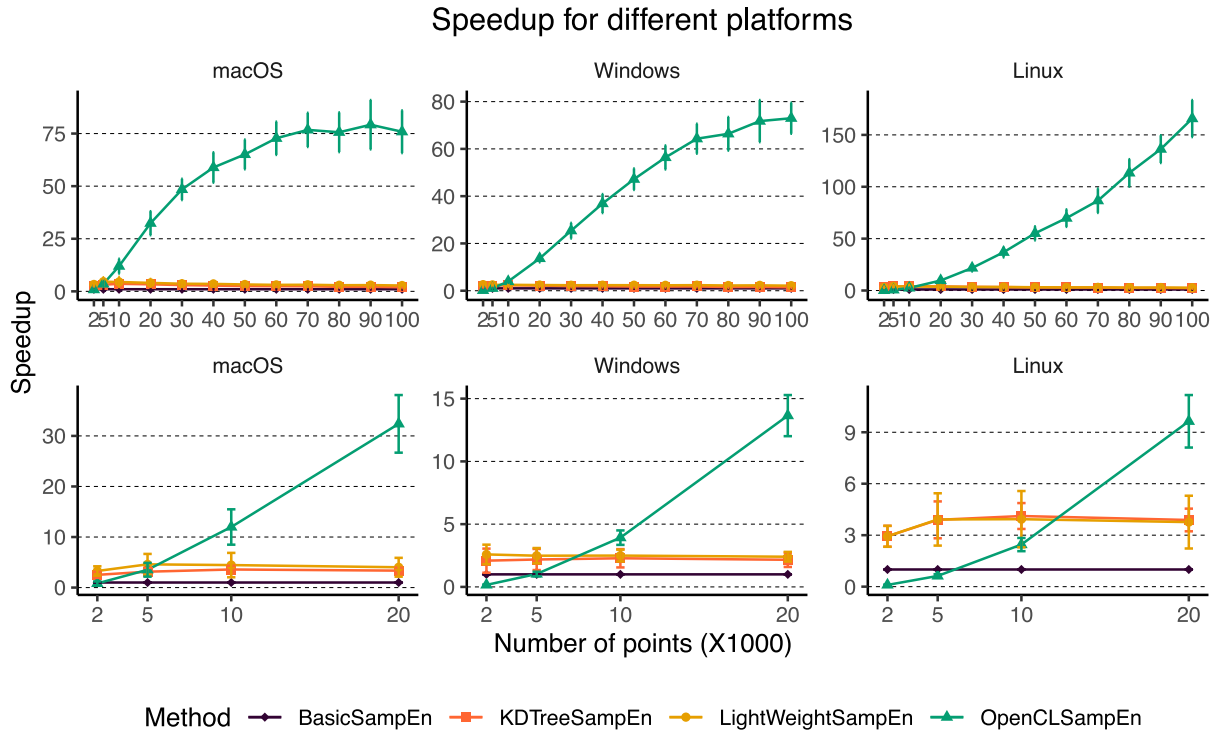


FIGURE 7. Speedup of the four algorithms on different operating systems (macOS, Windows, Linux) for typical parameter values $m = 2$ and $r = 0.15 * SD$. Values are given as the means \pm standard deviation. The top panels show the speedup of data length from 2,000 to 100,000. The bottom panels show the details of the corresponding top panels when the data length is between 2,000 and 20,000.

transferring data between the memory object and host memory takes up a small constant overhead. In addition, there is an extra overhead incurred when creating the GPU context before the calculation needs to be considered. Therefore, the method does not possess the advantage of computational performance in terms of sample entropy calculation for short time series. Another limitation is that a graphics card that supports OpenCL is required. This is not a problem for current mainstream PCs and HPCs. Therefore, we do not recommend analyzing long time series on a computer without a GPU.

V. CONCLUSION

We proposed a sample entropy fast algorithm based on OpenCL that exhibits a significant improvement in sample entropy computations for clinical data with a large data size. It combines entropy calculation, parallel computing, and OpenCL, uses the computing performance advantages of GPU to improve the sample entropy calculation, so as to obtain a stable, fast and wide-range sample entropy algorithm. These advantages provide the algorithm with broad utility in the analysis of sample entropy when facing the challenge of future rapid growth in the quantity of various continuous clinical and physiological signals.

REFERENCES

- [1] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul. 1948, doi: [10.1002/j.1538-7305.1948.tb01338.x](https://doi.org/10.1002/j.1538-7305.1948.tb01338.x).
- [2] R. Narayan and R. Nityananda, "Maximum entropy image restoration in astronomy," *Annu. Rev. Astron. Astrophys.*, vol. 24, no. 1, pp. 127–170, Sep. 1986.
- [3] A. Zellner, *Bayesian Methods and Entropy in Economics and Econometrics*. Amsterdam, The Netherlands: Springer, 1991.
- [4] C. Chen, Y. Jin, I. L. Lo, H. Zhao, B. Sun, Q. Zhao, J. Zheng, and X. D. Zhang, "Complexity change in cardiovascular disease," *Int. J. Biol. Sci.*, vol. 13, no. 10, pp. 1320–1328, 2017.
- [5] S. Pincus, "Approximate entropy as a measure of system complexity," *Proc. Nat. Acad. Sci. USA*, vol. 88, no. 6, pp. 2297–2301, 1991.
- [6] J. S. Richman and J. R. Moorman, "Physiological time-series analysis using approximate entropy and sample entropy," *Amer. J. Physiol.-Heart Circulatory Physiol.*, vol. 278, no. 6, pp. H2039–H2049, Jun. 2000, doi: [10.1152/ajpheart.2000.278.6.H2039](https://doi.org/10.1152/ajpheart.2000.278.6.H2039).
- [7] H. van Beijeren, J. R. Dorfman, H. A. Posch, and C. Dellago, "Kolmogorov-sinai entropy for dilute gases in equilibrium," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 56, no. 5, pp. 5272–5277, Nov. 1997.
- [8] M. Costa, A. L. Goldberger, and C.-K. Peng, "Multiscale entropy analysis of complex physiologic time series," *Phys. Rev. Lett.*, vol. 89, no. 6, Jul. 2002, Art. no. 068102, doi: [10.1103/PhysRevLett.89.068102](https://doi.org/10.1103/PhysRevLett.89.068102).
- [9] X. D. Zhang, Z. Zhang, and D. Wang, "CGManalyzer: An R package for analyzing continuous glucose monitoring studies," *Bioinformatics*, vol. 34, no. 13, pp. 1609–1611, Jul. 2018.
- [10] Y. Jin, C. Chen, Z. Cao, B. Sun, I. L. Lo, T.-M. Liu, J. Zheng, S. Sun, Y. Shi, and X. D. Zhang, "Entropy change of biological dynamics in COPD," *Int. J. Chronic Obstructive Pulmonary Disease*, vol. 12, pp. 2997–3005, Oct. 2017.
- [11] S. Sun, Y. Jin, C. Chen, B. Sun, Z. Cao, I. Lo, Q. Zhao, J. Zheng, Y. Shi, and X. Zhang, "Entropy change of biological dynamics in asthmatic patients and its diagnostic value in individualized treatment: A systematic review," *Entropy*, vol. 20, no. 6, p. 402, May 2018.
- [12] A. G. Hudetz, J. D. Wood, and J. P. Kampine, "Cholinergic reversal of isoflurane anesthesia in rats as measured by cross-approximate entropy of the electroencephalogram," *Anesthesiology*, vol. 99, no. 5, pp. 1125–1131, Nov. 2003.

- [13] Y.-H. Pan, Y.-H. Wang, S.-F. Liang, and K.-T. Lee, "Fast computation of sample entropy and approximate entropy in biomedicine," *Comput. Methods Programs Biomed.*, vol. 104, no. 3, pp. 382–396, Dec. 2011.
- [14] G. Manis, "Fast computation of approximate entropy," *Comput. Methods Programs Biomed.*, vol. 91, no. 1, pp. 48–54, Jul. 2008.
- [15] G. Manis, M. Aktaruzzaman, and R. Sassi, "Low computational cost for sample entropy," *Entropy*, vol. 20, no. 1, p. 61, Jan. 2018.
- [16] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.
- [17] C. Vinegoni, L. Fexon, P. F. Feruglio, M. Pivovarov, and R. Weissleder, "High throughput transmission optical projection tomography using low cost graphics processing unit," *Opt. Exp.*, vol. 17, no. 25, pp. 22320–22332, 2009.
- [18] Y. Watanabe and T. Itagaki, "Real-time display on Fourier domain optical coherence tomography system using a graphics processing unit," *J. Biomed. Opt.*, vol. 14, no. 6, 2009, Art. no. 060506.
- [19] L.-W. Chang, K.-H. Hsu, and P.-C. Li, "Graphics processing unit-based high-frame-rate color Doppler ultrasound processing," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 56, no. 9, pp. 1856–1860, Sep. 2009.
- [20] D. Liu and E. S. Ebbini, "Real-time two-dimensional temperature imaging using ultrasound," in *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, Sep. 2009, pp. 1971–1974.
- [21] P. Coupé, P. Hellier, X. Morandi, and C. Barillot, "Probe trajectory interpolation for 3D reconstruction of freehand ultrasound," *Med. Image Anal.*, vol. 11, no. 6, pp. 604–615, Dec. 2007.
- [22] I. Goddard, T. Wu, S. Thieret, A. Berman, and H. Bartsch, "Implementing an iterative reconstruction algorithm for digital breast tomosynthesis on graphics processing hardware," *Proc. SPIE*, vol. 6142, Mar. 2006, Art. no. 61424V.
- [23] G. Pratz and L. Xing, "GPU computing in medical physics: A review," *Med. Phys.*, vol. 38, no. 5, pp. 2685–2697, 2011.
- [24] P. Hastreiter and T. Ertl, "Integrated registration and visualization of medical image data," in *Proc. Comput. Graph. Int.*, Jun. 1999, pp. 78–85.
- [25] J. Antoine Maintz, "A survey of medical image registration," *Med. Image Anal.*, vol. 2, no. 1, pp. 1–36, Mar. 1998.
- [26] O. Fluck, C. Vetter, W. Wein, A. Kamen, B. Preim, and R. Westermann, "A survey of medical image registration on graphics hardware," *Comput. Methods Programs Biomed.*, vol. 104, no. 3, pp. e45–e57, Dec. 2011.
- [27] M. Rumpf and R. Strzodka, "Level set segmentation in graphics hardware," in *Proc. Int. Conf. Image Process.*, 2001.
- [28] J. Y. Hong and M. D. Wang, "High speed processing of biomedical images using programmable GPU," in *Proc. Int. Conf. Image Process. ICIP*, Oct. 2001, pp. 1103–1106.
- [29] O. Sharma, Q. Zhang, F. Anton, and C. Bajaj, "Multi-domain, higher order level set scheme for 3D image segmentation on the GPU," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2010, pp. 2211–2216.
- [30] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, "Medical image processing on the GPU—Past, present and future," *Med. Image Anal.*, vol. 17, no. 8, pp. 1073–1094, Dec. 2013.
- [31] M. Martínez-Zarzuola, C. Gómez, F. J. Díaz-Pernas, A. Fernández, and R. Hornero, "Cross-approximate entropy parallel computation on GPUs for biomedical signal analysis. Application to MEG recordings," *Comput. Methods Programs Biomed.*, vol. 112, no. 1, pp. 189–199, Oct. 2013.
- [32] E. I. Konstantinidis, C. A. Frantzidis, C. Pappas, and P. D. Bamidis, "Real time emotion aware applications: A case study employing emotion evocative pictures and neuro-physiological sensing enhanced by graphic processor units," *Comput. Methods Programs Biomed.*, vol. 107, no. 1, pp. 16–27, Jul. 2012.
- [33] A. Eklund, M. Andersson, and H. Knutsson, "fMRI analysis on the GPU—Possibilities and challenges," *Comput. Methods Programs Biomed.*, vol. 105, no. 2, pp. 145–161, Feb. 2012.
- [34] W. Shen, D. Wei, W. Xu, X. Zhu, and S. Yuan, "Parallelized computation for computer simulation of electrocardiograms using personal computers with multi-core CPU and general-purpose GPU," *Comput. Methods Programs Biomed.*, vol. 100, no. 1, pp. 87–96, Oct. 2010.
- [35] X. Zhu and D. Wei, "A computer simulation of clinical electrophysiological study," in *Proc. 30th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, Aug. 2008, pp. 585–588.
- [36] A. Munshi, "The OpenCL specification," in *Proc. Hot Chips Symp.*, Aug. 2011, pp. 1–314.
- [37] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra, "From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming," *Parallel Comput.*, vol. 38, no. 8, pp. 391–407, Aug. 2012.
- [38] E. S. John, G. David, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Comput. Sci. Eng.*, vol. 12, no. 3, pp. 66–72, May 2010.
- [39] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, Jun. 2000, doi: [10.1161/01.cir.101.23.e215](https://doi.org/10.1161/01.cir.101.23.e215).
- [40] M. Costa, A. L. Goldberger, and C.-K. Peng, "Multiscale entropy analysis of biological signals," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 71, no. 2, Feb. 2005, Art. no. 021906, doi: [10.1103/PhysRevE.71.021906](https://doi.org/10.1103/PhysRevE.71.021906).
- [41] J. M. Yentes, N. Hunt, K. K. Schmid, J. P. Kaipust, D. McGrath, and N. Stergiou, "The appropriate use of approximate entropy and sample entropy with short data sets," *Ann. Biomed. Eng.*, vol. 41, no. 2, pp. 349–365, Feb. 2013.
- [42] J. F. Valencia, A. Porta, M. Vallverdú, F. Claria, R. Baranowski, E. Orłowska-Baranowska, and P. Caminal, "Refined multiscale entropy: Application to 24-h holter recordings of heart period variability in healthy and aortic stenosis subjects," *IEEE Trans. Biomed. Eng.*, vol. 56, no. 9, pp. 2202–2213, Sep. 2009, doi: [10.1109/TBME.2009.2021986](https://doi.org/10.1109/TBME.2009.2021986).
- [43] S.-D. Wu, C.-W. Wu, S.-G. Lin, C.-C. Wang, and K.-Y. Lee, "Time series analysis using composite multiscale entropy," *Entropy*, vol. 15, no. 3, pp. 1069–1084, Mar. 2013.
- [44] S.-D. Wu, C.-W. Wu, S.-G. Lin, K.-Y. Lee, and C.-K. Peng, "Analysis of complex time series using refined composite multiscale entropy," *Phys. Lett. A*, vol. 378, no. 20, pp. 1369–1374, Apr. 2014.



XINZHENG DONG received the B.S. degree in computer science and technology from Information Engineering University, Zhengzhou, China, in 2006, and the M.S. degree in computer application technology from Beijing Information Science and Technology University, Beijing, China, in 2009. He is currently pursuing the Ph.D. degree in software engineering with the South China University of Technology, Guangzhou, China. His research interests include digital health, artificial intelligence, and high-performance computing.



CHANG CHEN received the B.S. degree in applied mathematics from Beijing Normal University, China, in 2016, and the Ph.D. degree in biomedical sciences from the University of Macau, in 2020. She is currently working with MSD Research and Development (China) Company Ltd. as a Statistician. Her research interests include exercise, heart rate variability, and consistency analysis.



QINGSHAN GENG received the Ph.D. degree in cardiovascular medicine from the Guangdong Cardiovascular Institute, Guangzhou, China, in 2000.

He is currently a Chief Physician of cardiology with Guangdong General Hospital and a Distinguished Professor with the South China University of Technology. His research interests include cardiovascular medicine, behavioral medicine, and psychosomatic medicine.



WENSHENG ZHANG is currently a Professor with the Research Center of Precision Sensing and Control, Institute of Automation, Chinese Academy of Sciences, China. He has published over 160 refereed articles in flag-ship journals, such as the IEEE, Elsevier, IFAC, and Springer, international conferences, and book chapters. He also has more than 40 authorized invention patents. As a principal or co-principal investigator, he participated over 20 China-funded research

projects. His broad research interests include artificial intelligence, machine learning, big data knowledge mining, probability graph model representation and reasoning, deep learning, precision perception and intelligent control, 3D digital physical simulation, and embedded video image processing. He received several awards, including the Second Prize of National Science and Technology Progress Award, and so on.



XIAOHUA DOUGLAS ZHANG (Senior Member, IEEE) is currently a Professor with the Faculty of Health Science, University of Macau, China. He is also a fellow of the American Statistical Association and an elected member of the International Statistical Institute. He has published over 100 refereed articles. He has worked in quantitative high-throughput genotyping and phenotyping for more than 20 years. The novel methods developed by him and his colleagues along with

their applications have a high impact on the high-throughput screening research field, especially in RNAi high-throughput screening (HTS) research. His lab focuses on research in digital health/medicine, artificial intelligence, and big data analysis. His current research in this area includes conducting studies in continuous monitoring of physiological signals (such as blood glucose, respiratory signals, and cardiovascular signals) and high-throughput genomics (such as CRISPR/CAS9 HTS and RNA-seq) to address medical questions in diabetes, respiratory diseases, cardiovascular diseases, neuro diseases, cancers, and so on. Critical research also includes the development of novel experimental designs, analytic methodology, and software (including R packages) for analyzing big data in digital health/medicine and precision medicine.

...