Parallelization in biomedical time series analysis web platform: the MULTISAB project experience

Alan Jović[*], Krešimir Jozić[**], Davor Kukolja[*], Krešimir Friganović[*], Mario Cifrek[*]
[*] University of Zagreb Faculty of Electrical Engineering and Computing / Department of Electronics, Microelectronics, Computer and Intelligent Systems, Unska 3, 10 000 Zagreb, Croatia
[**] INA - industrija nafte, d.d., Avenija Većeslava Holjevca 10, p.p. 555, HR-10002 Zagreb, Croatia
Corresponding author: alan.jovic@fer.hr

Abstract - **Parallel execution of operations required for biomedical time series (BTS) analysis is an important issue in optimization of medical software efficiency. We investigate the applicability of several parallelization approaches to BTS analysis in the context of feature extraction from multiple heterogeneous BTS on a Java-based web platform called MULTISAB, designed for medical diagnostics. Considering only the calculation parallelization of many different BTS features, our research suggests that parallelization based on simple Java multithreading works the best. The threads are assigned based on the data and analysis parameters provided, where feature extraction parallelization is performed on the following levels: 1) multiple segments; 2) multiple signal trails; 3) multiple patient records. The synchronization mechanism should be simple: the analysis continues once all threads terminate their work and record the extracted feature vectors. A special case, when features from two signal trails (bivariate features) are extracted, includes multithreading on signal pairs for multiple signal trails level parallelization. We also provide an overview of the web platform architecture to put the parallelized parts into the overall perspective.**

Keywords - biomedical time series analysis, feature extraction, web platform, parallelization, multithreading

I. INTRODUCTION

Biomedical time series (BTS) analysis software usually possesses a significant amount of computational complexity, due to the fact that the implemented algorithms need to efficiently manage noisy and non-stationary properties of the considered time series [1]. The degree of complexity depends largely on the goal of the analysis and the features used in reaching that goal. For example, extracting linear statistical features such as mean and standard deviation from an already preprocessed BTS in order to determine if an analyzed segment significantly deviates from an expected behavior does not require intensive calculations. However, performing empirical mode decomposition based Hilbert-Huang transform and extracting many potentially relevant time-frequency features on a broader scale of learning a model for a disease, based on many patient records, is highly resource-demanding [2].

In our earlier investigations, we approached the problem of designing the architecture for a BTS analysis web platform called MULTISAB[1] [3]. The platform is intended to provide ubiquitous web access to interested researchers and medical personnel that want to analyze BTS. The platform is based on specification and execution of analysis scenarios. We have also previously shown the feature implementation details regarding heart rate variability, electrocardiogram, and electroencephalogram analysis, also including a definition of expert recommendation system for features that need to be extracted, depending on the analysis goal and data type [4]. Data mining methods, including feature selection and classification algorithms are also implemented in the platform, which is a topic of another, currently submitted conference paper [5]. At the moment, the platform is in integration and testing phase and is planned to be open for users soon.

In this work, we focus on the details regarding computational complexity amelioration that has been achieved in the platform. In particular, we address the complexity problem by examining the applicability of different parallelization approaches to the feature extraction step of the analysis. As feature extraction is the crucial part of BTS analysis, accelerating this step brings big difference to the web platform user experience.

The work is organized as follows. In Section 2, we provide a brief overview of the current state of MULTISAB platform implementation and highlight the spots where parallelization may be beneficiary. Section 3 first discusses some related work and then describes our attempt at directly parallelizing individual feature calculations. Section 4 provides the details of parallelization implemented in the platform, including some experimental validation. Section 5 concludes the paper and provides future guidelines.

II. PLATFORM STRUCTURE AND PARALLELIZATION LOCATION

A. MULTISAB Platform Structure

The MULTISAB web platform is organized in three separate subprojects: *frontend*, *backend* and *processing*.

[1] This work has been fully supported by the Croatian Science Foundation under the project number UIP-2014-09-6889.

The main technologies and intended tasks of each subproject are briefly explained here.

The *frontend* subproject is designed using several recognizable contemporary web frontend technologies, including Angular, HTML5, CSS3, and Node.js. The web site is organized as a single-page web application written using Angular components in TypeScript language, where a TypeScript compiler is used to produce JavaScript code.

The majority of web application's content is focused on conducting an analysis scenario. The user starts the analysis scenario by logging into the platform and opening a new session (or continuing the last existing one). BTS analysis process is divided into 8 steps, some of which may be skipped, depending on the user: 1) analysis type selection, 2) scenario selection, 3) input data selection, 4) records inspection, 5) records preprocessing, 6) feature extraction, 7) model construction, and 8) reporting [4].

The analysis on *frontend* proceeds from a starting step to an allowed set of next steps, which is continued until the final step, reporting, is reached. The possible transitions between steps are governed through a finite state machine (FSM), where a step may correspond to one or a few FSM states. Using this approach, we managed to provide a clearly defined BTS analysis workflow, which would not be possible using standard web site navigation, where any (or most) transitions between steps would be legal. An example of possible step transitions in the platform are shown in Fig. 1, where the user is currently in step "4. Data plot" (records inspection) and may move to the earlier step "3. Select input data" or to steps "5. Preprocessing" and "6. Feature extraction". OpenAPI [6] is used to create the documentation for the MULTISAB's RESTful API that is used for testing the communication between backend and frontend subprojects.

The *backend* subproject is designed using Java 9 and Spring Boot. Java Persistence API (JPA) is used for communication with the database, which is also a part of the backend project. H2 database management system was chosen primarily because it stores all data in a single file. As we store the uploaded BTS files into the *backend* server file system, the H2 database is used only to store registered users and sessions data. The *backend* subproject is built using Maven build tool in order to easily integrate all the required libraries. We use stateful session control by tokens. After logging in, the user is given a token for accessing protected resources on backend, which expires after a timeout if no action is made. For security insurance, we use HTTP/2 protocol. *Backend* currently includes the *processing* subproject as a library. *Backend* calls the methods of relevant classes in *processing* in order to enable the execution of analysis steps. For example, a call from `analysis.RecordsInspectionController` class on *backend* to `signalVisualization.ImageCreating` class on *processing* enables the construction of an image file for visualization of a patient record segment.

The *processing* subproject is designed in Java 9, and is intended to cover all the details regarding BTS analysis. It is implemented as a stand-alone library with potential to be distributed on several hosts. Currently, it is located on the same server as the *backend* subproject. The structure of the *processing* subproject is the most complex one, consisting of several frameworks for handling most of the analysis steps as well as additional platform functionalities. The frameworks with some of the most important packages (small first letter) and classes (capital first letter) are depicted in Table I.

TABLE I. MULTISAB PROCESSING SUBPROJECT FRAMEWORKS WITH IMPORTANT PACKAGES OR CLASSES

Framework	Package or class
Record input handling	`AnnFile`
	`CsvFile`
	`EdfFile`
	`InputData`
	`Metadata`
	`SignalParameterData`
	`TxtFile`
Preprocessing	`filtering`
	`iirj`
	`morphologicalOperations`
Signal visualization	`ImageCreating`
General time series features extraction	`frequencyDomain`
	`nonlinear`
	`timeDomain`
	`timeFrequencyDomain`
Specific (domain) time series features extraction	`eegAnalysis`
	`ecgAnalysis`
	`hrvAnalysis`
Feature extraction	`FeatureExtraction`
	`Parallelization`
Expert system recommendations	`DroolsExpertSystem`
	`ExpertSystem`
Data mining	`discretization`
	`featureSelection`
	`normalization`
	`Classification`
Reporting	`ReportingMain`

More details regarding *processing* functionalities are available in [5]. It is important to stress out that a large number of general BTS features (e.g. approximate entropy, mutual information) as well as specific BTS (i.e. HRV, ECG, EEG) features are implemented in the platform. Also, preprocessing steps, including filtering methods and ECG characteristic points detection using state-of-the-art algorithms such as the one from Elgendi et al. [7] are also supported. Data mining methods currently implemented include several filter-based feature selection algorithms (e.g. symmetrical uncertainty, Chi-square) and several classifiers: SVM [8], MLP, RBF, PNN and NEAT [9]. Aside from our own implementations, permissive licenses (Apache, MIT, BSD, or LGPL) only external libraries are included in MULTISAB.

B. Parallelization Candidate Locations

We carefully examined the potential spots in the platform that would benefit from parallelization. We concluded that efficient speed up of BTS analysis may be beneficiary in the preprocessing, feature extraction and data mining steps, as these are the computationally most demanding steps.

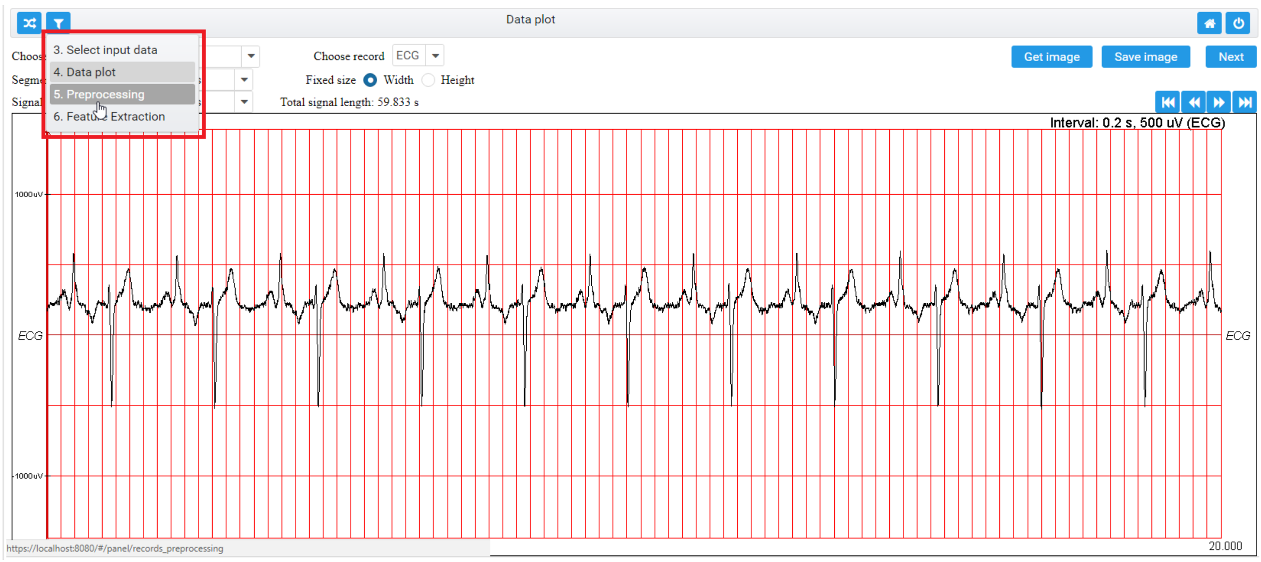

Figure 1. Moving through the analysis steps on frontend: an example where the user is currently in "4. Data plot" (records inspection) step, the relevant part is highlighted with the red rectangle

Our initial considerations indicated that input data preprocessing (such as applying various filters) parallelization may be achieved only for a subset of preprocessing methods. Also, heavy use of preprocessing methods is something that is not the primary focus of the platform. Therefore, currently, if any computationally demanding preprocessing is required, it is expected that the platform would take some extra time. Our decision was to focus first on parallelization of feature extraction, on which we report in detail in Section 3.

It is known that data mining algorithms, in particular, various ANN and SVM algorithms, greatly benefit from parallelization when training with sufficiently large datasets. Generally, parallelizing such algorithms is not trivial, as the internal steps are dependent on the previous calculations. Nevertheless, some efficient parallelization implementations are known, mostly based on OpenMP and GPU [10,11]. This is something that we may consider in the next phase of MULTISAB implementation, but probably not using OpenMP, as its support for Java is limited. We also plan to implement some naturally parallelizable data mining algorithms soon, such as random forest [12].

III. Parallelization Approaches

A. Related Work

The work of Wilson and Williams [13] reported achieving significant performance increase (up to 100x) using GPU parallelization in the case of many-channel (more than 100 channels) for EEG spatial filtering, because matrix multiplications, easily processed by GPU, are used for that purpose. The same work also reported success for parallelizing autoregressive Burg algorithm for power spectral density estimate. However, specific Burg algorithm code optimizations were needed, which prohibits a more general solution for various BTS preprocessing methods implemented in our platform. Chen et al. [14] managed to parallelize ensemble empirical mode decomposition algorithm for many-channel EEG on a CUDA GPU and thus enable faster calculation of Hilbert-Huang spectral entropy, which was used to detect epileptic seizures. Signal preprocessing parallelization may also be realized in a specialized hardware, as was shown by Ahn et al. [15].

The work of Sahoo et al. [16] focused on a web based cloud solution for ECG records preprocessing and visualization. The platform called Cloudware relies on Hadoop 'big data' infrastructure and enables fast preprocessing of ECGs. Parallelization is achieved on patient record and segment levels through the use of MapReduce jobs. By translating EDF format patient data to WFDB format and utilizing PhysioNet preprocessing algorithms (e.g. for R peak detection), which were not parallelized, resulted in faster preprocessing and inspection of large amounts of data available to the medical multicenter in which the system was installed.

B. Attempting BTS Feature Calculation Parallelization

Regarding feature extraction from BTS, we analyzed several possible approaches to parallelization. The first idea was to parallelize feature calculation algorithms themselves. Specifically, we considered GPU-based as well as CPU-based parallelization using Java OpenCL library called aparapi [17]. Aparapi is designed to translate native Java bytecode into OpenCL kernels dynamically at runtime, thus dispatching the tasks either to a multi-core CPU or to GPU, depending on the settings. OpenCL CPU uses processor drivers that need to be installed on the computer and the most advanced instruction set that we have available (e.g. AVX2 for an Intel processor). OpenCL GPU uses GPU's shader processors and, by default for large data sets, global GPU RAM. Initial experiments with aparapi GPU and CPU parallelization indicated more than an order of magnitude acceleration compared to Java Thread Pool. However, the test data included millions of signal samples and non-conditional simple calculations. When faced with a real-world feature calculation algorithm (e.g. approximate entropy), taking only several thousand samples, conditional next-step execution and non-trivial mathematical operations (e.g. square root), the results were far less impressive. More precisely, the overhead needed to transfer data to CPU or GPU, as well as inability to parallelize most of the feature calculation steps, resulted in parallelization execution time that was significantly longer than the traditional sequential algorithm. The most important problems seemed to be the nature of the feature calculation algorithm itself and dynamic translation of OpenCL kernel. Having multiple interdependent sequential steps, some conditional branching and non-trivial mathematical operations (which prevents parallel reduction technique [18]) resulted in significant performance drop. This was the conclusion for all the inspected feature calculation algorithms.

IV. Parallelization Implementation in MULTISAB

A. Parallelization Context

The *backend* subproject calls the methods of the *processing* subproject when a user's request from *frontend* arrives. Parallelization is handled as one of such requests, currently supported during feature extraction step. The assumptions for starting feature extraction are, as follows:

- There may be multiple patient records uploaded and preprocessed in the earlier steps of the analysis session in the platform.
- Patient records may contain heterogeneous signals (signal trails), e.g. 10 EEG trails, 2 ECG trails and 1 skin conductivity trail.
- The number of feature extraction iterations is equal to the number of different signal types (i.e. ECG, EEG...) in the records.
- All uploaded and preprocessed patient records contain the same types and numbers of signal trails.
- Each iteration of feature extraction is performed on one signal type in all records, because each signal type uses different features and feature parameters.
- All signals in all uploaded and preprocessed records are of equal duration, and if this condition is not satisfied, the analysis may be performed only until the end of the shortest signal trail in all records.

Prior to execution of parallelization for a feature extraction iteration, all the feature extraction parameters need to be set:

- The list of features that need to be extracted in the iteration.
- The list of feature parameters with values for each feature that needs to be extracted in the iteration (in the case where a feature has some parameters).
- The starting time in the record from which the feature extraction process starts (the same for all iterations).
- The analyzed segment width (the same for all iterations).
- The final time in the record until which the analysis is performed (the same for all iterations).

All of these feature extraction parameters are sent from the *backend* subproject after the user specified them via browser GUI on *frontend*. The user may choose not to use parallelization. In such a case, classical sequential feature extract would ensue.

B. Parallelization Progression

Parallelization starts by a call from *backend* to class `Parallelization` within the package called `analysis` in the *processing* subproject. The `Parallelization` class is designed to take as input the paths to patient records that were uploaded and which may have been preprocessed in the earlier steps of an analysis session. The file paths are stored in the database on `backend`. During object instantiation of the `Parallelization` class, all the selected data records are loaded into the working memory to enable faster calculations. Input record size as well as total upload size may be limited by the platform to prevent out-of-memory errors. The data structure in memory is somewhat dependent on the input file format, but it nevertheless allows the use of the same methods for parallelization of execution.

We achieved parallelization through the use of Java multithreading, without aparapi library. We limit the degree of parallelization to the number of available logical cores on the server computer at the moment of parallelization starting minus one (let the number of parallel threads in each such parallelization chunk be *k*). Parallelization is achieved on several levels: for multiple patient records, within a record for multiple signals of the same type, and for multiple segments within a signal trail. The procedure for parallelization of feature extraction is implemented, as follows:

1. If there is more than one segment present within a signal, then the features in these segments are extracted in parallel, where the parallelization proceeds until all the segments are analyzed.
2. Else, if there is only one segment per signal and there are multiple signals of the same type, then the signals are processed in parallel, where the parallelization proceeds until all the signals are analyzed; a special case for the implemented bivariate features is that all signal pairs (e.g. for calculation of the mutual information feature) are analyzed in parallel.
3. Else, if there is only one segment and one signal of the same type per record, and there is more than one record, then the parallelization is run so that multiple records are analyzed concurrently, which proceeds until all of them are analyzed.
4. Else, when there is only a single record with a single signal type and a single segment, then the parallelization is not performed and the record is analyzed in the original thread.

It is important to note that the extraction of the list of features in a single segment, signal, or record always proceeds sequentially. The reason for this is because the complexity of individual features calculation differs significantly from one feature to another. Therefore, due to easier synchronization, we decided not to create new threads for each feature calculation.

The parallelization synchronization is, as follows: all *k* threads need to finish with the current feature extraction before moving on to the next chunk of parallelization. This may slow down the possibly achievable runtime of parallelization, but the synchronization is simpler and thus less error-prone. After all *k* threads finish, the results of feature extraction (feature vectors) are appended to two output files, one in .csv format and the other one in a more informative, .arff file format [19]. The process is repeated for each parallelization chunk, until all the records are analyzed in the iteration.

After all the iterations are completed, the model construction step (data mining) may begin. Therein, complex models may consider features from different signal types stored in several output files, and also other patient metadata, while simple models may use feature vectors extracted from a single type of BTS.

In Table II, we state the most important classes in the *processing* subproject that directly or indirectly (through object references) take part in the feature extraction parallelization process. All of these classes may be found in the `analysis` package of the subproject.

C. *Parallelization Validation*

As demonstration, we conduct a validation experiment of the implemented feature extraction parallelization to establish its efficiency compared to sequential execution. For that purpose, we proceed with a rather complex feature extraction scenario for extracting multiple features (27 in total) from cardiac rhythm annotation records from PhysioNet MIT-BIH Arrhythmia database [20], which is the standard database for testing algorithms to detect various types of arrhythmias based on heart rate variability (HRV).

We include the following general time series features: four approximate entropies (for $m = 2$, $r = \{0.1, 0.15, 0.2, 0.25\}*\sigma$), four sample entropies (for m = 2, $r = \{0.1, 0.15, 0.2, 0.25\}*\sigma$), corrected conditional Shannon entropy, spatial filling index, Allan factor, and five recurrence plot features (rec. rate, Lmean, DET, rec. Shannon entropy, and rec. laminarity). We also include: 1) standard time domain HRV features – AVNN, SDNN, RMSSD, SDSD, pNN50, HRV triangular index, TINN; 2) standard frequency domain features calculated using Lomb-Scargle periodogram – low frequency PSD, high frequency PSD, and low/high frequency PSD ratio; and 3) standard deviation ratio of Poincaré plot feature. Some details about the features may be found in [21].

The experiments were conducted on an Intel Core i7-4790 CPU @3.6 GHz with 16 GB RAM and 8 logical cores, of which 7 were used for parallelization. We used three settings: 1) only a single record, "100.ann"; 2) first 12 records of the MIT-BIH Arrhythmia Database; and 3) all 48 records of the MIT-BIH Arrhythmia Database.

The results are shown in Table III. The experiments were run five times, with mean ± stddev reported. We can see that, with the exception of single record with the largest number of segments and shortest segment length, parallel execution outperforms sequential execution. We can observe that few longer segments take more time to analyze than many shorter segments. This is because most of the features calculations complexities are more than linearly dependent on the size of the segments. Based on the results for 7 segments, we can expect that the largest positive effect of parallelization takes place when the number of segments is the multiple of the number of used logical cores. Also, as we can see from the results of analyzing only 2 longest segments, better positive effect can be achieved by analyzing longer segments (the effect is almost the theoretical double for 48 records). The reason for this is because threads spend most of the time calculating the features, and less time preparing data for the next chunk or waiting on other threads to finish the current chunk. Based on these results, we may conclude that the maximum effect of multithreading for a general BTS analysis may be achieved for long segments and for the number of segments given as multiple of the number of used logical cores.

V. CONCLUSION

In this work, we have shown the organization of the MULTISAB platform and highlighted the steps that may

TABLE II. *PROCESSING* CLASSES THAT IMPLEMENT PARALLEL (AND SEQUENTIAL) FEATURE EXTRACTION

Class	Purpose
`Parallelization`	The main class intended for starting parallelization. It supports loading data from preprocessed files, adjusting parameters and job scheduling among threads
`ParallelExtraction Thread`	A single thread that deals with the analysis of a single data section (commonly, a file segment, rarely the whole file)
`Features`	The class intended for storing and transferring features that need to be analyzed
`FeatureParameters`	The class intended for storing and transferring parameter values for individual features that need to be analyzed
`FeatureExtraction`	The class intended for feature extraction from data. As input, it takes signal data needed to be processed. As output, extracted feature vectors are sent as results to the `ExtractedFeatures` class
`ExtractedFeatures`	The class intended to save the extraction results to two output files. Feature vectors are stored line by line in the output files, with the values that correspond to the extracted features and feature parameters specified by the user and available in the files' headers

TABLE III. PARALLELIZATION VALIDATION RESULTS ON MIT-BIH ARRHYTHMIA DATABASE RECORDS, RESULTS ARE IN MILLISECONDS

No. of analyzed record segments, with their length	**Parallel / sequential execution**	**Number of included records**		
		1	12	48
85 segments, each 20 s	parallel	116±30	357±14	1135±79
	sequential	103±14	393±15	1304±66
18 segments, each 90 s	parallel	145±13	447±9	1506±23
	sequential	178±8	962±29	3372±13
7 segments, each 240 s	parallel	200±17	890±43	2519±12
	sequential	266±16	1973±20	8002±42
3 segments, each 560 s	parallel	315±7	1966±23	7858±204
	sequential	531±16	4268±117	17672±115
2 segments, each 840 s	parallel	466±7	3556±49	14781±244
	sequential	766±29	6181±55	26395±223

benefit from parallelization. The specifics of the platform mandated a practical solution to accelerate feature extraction. We have learned that, due to the nature of feature calculations, it is impractical to perform OpenCL based CPU and GPU parallelization on the level of individual feature calculation algorithms. In our experience, the best solution was to use Java multithreading and parallelize feature extraction based on available data, by examining multiple segments, signals and records at the same time.

In the future, we plan to consider several venues for improvements. First, as mentioned, we would like to add the option to the *backend* subproject to communicate with multiple computers, each host containing an instance of the *processing* subproject, thus increasing the degree of parallelization. For some specific preprocessing algorithms for which parallelization was shown to be beneficial in literature [13], we also plan to add parallelization support using GPU. Another venue is to explore parallelization for some specific, yet unimplemented complex multivariate features, such as the one measuring synchronization between brain regions in an EEG [22].

REFERENCES

[1] S. A. Rahman, Y. Huang, J. Claassen, N. Heintzman, and S. Kleinberg, "Combining Fourier and lagged k-nearest neighbor imputation for biomedical time series data," *Journal of Biomedical Informatics*, vol 58, pp. 198–207, 2015; doi: 10.1016/j.jbi.2015.10.004

[2] R. J. Oweis and E. W. Abdulhay, "Seizure classification in EEG signals utilizing Hilbert-Huang transform," *BioMedical Engineering OnLine*, vol. 10, p. 38. 2011; doi:10.1186/1475-925X-10-38

[3] K. Friganović, A. Jović, K. Jozić, D. Kukolja, and M. Cifrek, "MULTISAB project: a web platform based on specialized frameworks for heterogeneous biomedical time series analysis - an architectural overview," *Proceedings of the International Conference on Medical and Biological Engineering (CMBEBiH 2017)*, Sarajevo, Bosnia and Herzegovina, Springer Nature Singapore, pp. 9–15, 2017.

[4] A. Jović, D. Kukolja, K. Friganović, K. Jozić, and S. Car, "Biomedical Time Series Preprocessing and Expert-System Based Feature Extraction in MULTISAB Platform," *Proceedings of MIPRO 2017 International Conference*, Opatija, Croatia, MIPRO, pp. 349–354, 2017.

[5] A. Jović, D. Kukolja, K. Friganović, K. Jozić, and M. Cifrek, "MULTISAB: A Web Platform for Analysis of Multivariate Heterogeneous Biomedical Time-series," *World Congress on Medical Physics & Biomedical Engineering (IUPESM 2018)*, Prague, *submitted work.*

[6] SmartBear Software, "Swagger Editor and Swagger UI," https://swagger.io/ (accessed: 2018-02-11)

[7] M. Elgendi, M. Meo, and D. Abbott, "A Proof-of-Concept Study: Simple and Effective Detection of P and T Waves in Arrhythmic ECG Signals," *Bioengineering*, vol. 3, p. 26, 2016; doi:10.3390/bioengineering3040026

[8] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 27, pp. 1–27, 2011.

[9] J. Heaton, "Encog: Library of Interchangeable Machine Learning Models for Java and C#," *Journal of Machine Learning Research,* vol. 16, pp. 1243–1247, 2015.

[10] P. Chang, Z. Bi, and Y. Feng, "Parallel SMO algorithm implementation based on OpenMP," *2014 IEEE International Conference on System Science and Engineering (ICSSE)*, Shanghai, pp. 236–240, 2014; doi: 10.1109/ICSSE.2014.6887941

[11] A. A. Huqqani, E. Schikuta, S. Ye, and P. Chen, "Multicore and GPU Parallelization of Neural Networks for Face Recognition," *Procedia Computer Science*, vol 18, pp. 349–358, 2013; doi:10.1016/j.procs.2013.05.198

[12] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001; doi: 10.1023/A:1010933404324.

[13] J. A. Wilson and J. C. Williams, Massively parallel signal processing using the graphics processing unit for real-time brain-computer interface feature extraction, *Frontiers in Neuroengineering,* vol. 2, pp. 1–11, 14 July 2009; doi: 10.3389/neuro.16.011.2009

[14] D. Chen, L. Wang, G. Ouyang, and X. Li, "Massively Parallel Neural Signal Processing on a Many-Core Platform," *Computing in Science & Engineering,* vol. 13, no. 6, pp. 42–51, 2011; doi: 10.1109/MCSE.2011.20

[15] J. Ahn, H. Chang, J. Cho, and W. Sung, "SIMD processor based implementation of recursive filtering equations," *2009 IEEE Workshop on Signal Processing Systems*, Tampere, Finland, pp. 87–92, 2009; doi: 10.1109/SIPS.2009.5336230

[16] S. S Sahoo, C. Jayapandian, G. Garg, F. Kaffashi, S. Chung, A. Bozorgi, C.-H. Chen, K. Loparo, S. D Lhatoo, and G.-Q. Zhang, "Heart beats in the cloud: distributed analysis of electrophysiological 'Big Data' using cloud computing for epilepsy clinical research," *Journal of the American Medical Informatics Association*, vol. 21, issue 2, pp. 263–271, 2014; doi:10.1136/amiajnl-2013-002156

[17] Syncleus, "Aparapi," http://aparapi.com, (accessed: 2018-02-11)

[18] NVIDIA, "NVIDIA CUDA Compute Unified Device Architecture Programming Guide," v. 2.0. Santa Clara, CA, NVIDIA, 2008.

[19] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explor. Newsl.* vol. 11, no. 1, pp. 10–18 2009; doi:10.1145/1656274.1656278

[20] G. B. Moody and R. G. Mark, "The impact of the MIT-BIH Arrhythmia Database," *IEEE Engineering in Medicine and Biology*, vol. 20, no. 3, pp. 45–50, 2001.

[21] A. Jović and N. Bogunović, "Evaluating and Comparing Performance of Feature Combinations of Heart Rate Variability Measures for Cardiac Rhythm Classification," *Biomedical Signal Processing and Control*, vol. 7, no. 3, pp. 245–255, 2012.

[22] D. Chen, X. Li, D. Cui, L. Wang, and D. Lu, "Global Synchronization Measurement of Multivariate Neural Signals with Massively Parallel Nonlinear Interdependence Analysis," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 22, no. 1, pp. 33–43, 2014; doi: 10.1109/TNSRE.2013.2258939