J Med Syst (2015) 39: 18
DOI 10.1007/s10916-015-0197-3

TRANSACTIONAL PROCESSING SYSTEMS

Rapid Automated Classification of Anesthetic Depth Levels using GPU Based Parallelization of Neural Networks

Musa Peker · Baha Şen · Hüseyin Gürüler

Received: 29 June 2014 / Accepted: 13 January 2015 / Published online: 4 February 2015

Abstract The effect of anesthesia on the patient is referred to as depth of anesthesia. Rapid classification of appropriate depth level of anesthesia is a matter of great importance in surgical operations. Similarly, accelerating classification algorithms is important for the rapid solution of problems in the field of biomedical signal processing. However numerous, time-consuming mathematical operations are required when training and testing stages of the classification algorithms, especially in neural networks. In this study, to accelerate the process, parallel programming and computing platform (Nvidia CUDA) facilitates dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU) was utilized. The system was employed to detect anesthetic depth level on related electroencephalogram (EEG) data set. This dataset is rather complex and large. Moreover, the achieving more anesthetic levels with rapid response is critical in anesthesia. The proposed parallelization method yielded high accurate classification results in a faster time.

Keywords Anesthesia · Anesthetic depth level · Parallel programming · Parallel processing · Neural networks · EEG

This article is part of the Topical Collection on *Transactional Processing Systems*

M. Peker
Department of Information Technologies, Samandira Vocational and Technical School, Istanbul, Turkey

B. Şen
Department of Computer Engineering, Yildirim Beyazit University, Ulus Ankara, Turkey

H. Gürüler (✉)
Department of Information Systems Engineering, Faculty of Technology, Mugla Sitki Kocman University, Mugla, Turkey
e-mail: hguruler@mu.edu.tr

Introduction

General anesthesia means the suppression of activity in the central nervous system. It has three main foundations: unconsciousness, lack of movement (paralysis), and blunting of the stress response [1]. The aim of anesthesia is to reach these required endpoints with the minimum amount of risk possible to the patient. The optimal anesthetic drug would be provided hypnosis, amnesia, analgesia, and muscle relaxation without unwanted changes in blood pressure, heartbeat, breathing etc. [2]. General anesthetic drugs cause a depression in the central nervous system, starting from cortical and psychic centers and following basal ganglions, cerebellum, medulla spinalis and medullary centers, respectively. In surgical operations, it is important for the medical staff to have information regarding the depth of anesthesia (DOA) in a reliable and non-invasive manner thus, they can safely regulate the dose of the anesthetic [3]. Keeping the patient at the optimum anesthesia level during surgery is crucial and a challenging issue in clinical practice [4, 5]. Because both inadequate and excessive use of anesthetic drug are undesirable for anesthesia. Deep anesthesia can cause coma and death by depressing vital functions and causing depression in bulbar centers in more advanced stages. Whereas light anesthesia can be harmful because it cannot prevent painful and harmful stimulus, neuroendocrine and reflex responses given to them enough. For these reasons, research the methods for the determination of the DOA precisely is still in progress [2, 6].

Traditional methods are used to understand the DOA include measurements such as the patient's heart rate, oxygen saturation level (SpO_2), body movements, blood pressure, pupil size and level of perspiration. However, these physical symptoms vary depending on the type of surgery and from patient to patient. Additionally, long-term operations such as orthopedic surgery and operations on the brain, heart and spine, traditional indicators may not be sufficient and this can result in a patient gaining consciousness. Thus, the complexity of general anesthesia procedures requires the development of computer assisted support systems functioning in surgery [7].

Since anesthetic agents affect the brain's cortex, the monitoring of brain activity using EEG is a noninvazive approach to determine the DOA [8]. There are different EEG-based approaches to determine the DOA. Bispectral index (BIS) is the most commonly used brain monitor. BIS processes a single frontal electroencephalograph signal to calculate a dimensionless number for reflecting the patient's level of consciousness [9, 10]. The use of BIS monitoring helps in reducing anesthetic requirements, ensuring proper maintenance of the hypnotic state and it helps to avoid both extremely deep anesthesia and light anesthesia [11, 12].

Despite the widespread use; BIS index is also known to have some certain omissions. For instance, Gurkan et al. [8] reported that they observe different spectral features in EEGs of different patients in the same BIS values. This is caused by that BIS device updates 30-second EEG data with 10-second intervals. Since the short-term instantaneous changes in EEG that reflected in the spectral analysis are not reflected to the BIS value, momentarily DOA of the patient cannot be successfully estimated. In the same study it was observed that BIS value was still around the value of 70 when the patient became sober. This situations make reliability of the BIS device questionable for under some circumstances.

In fact, the disadvantages such as sensitivity and slow response rate on BIS index, can be resolved with faster systems [13, 14]. It can be predicted that this situation may be overcome by the systems with a lower response time, such as pipeline systems operating simultaneously or in a parallel configuration.

In this study, CUDA programming is applied to EEG data to facilitate faster classification of anesthetic depth level. To better display the effect of the approach, a multilayer perceptron (MLP) neural network which requires intensive mathematical operations was chosen as the classification algorithm. Mathematical operations are fundamentally involving; forward computation steps, the calculation of the activation functions, and weight updates to achieve optimal performance.

As a result of rapid development of the speed of graphics cards compared with CPUs, GPU parallel programming has become an important field of study. As shown in Fig. 1, there is a tremendous increase in number of cores in graphics cards compared to number of processors' cores. Therefore, the GPUs provide high processing performance this can be achieved by CUDA which is a GPU architecture that allows parallel processing of the codes, written in programming languages such as C and C++, on the graphics processor unit [15].

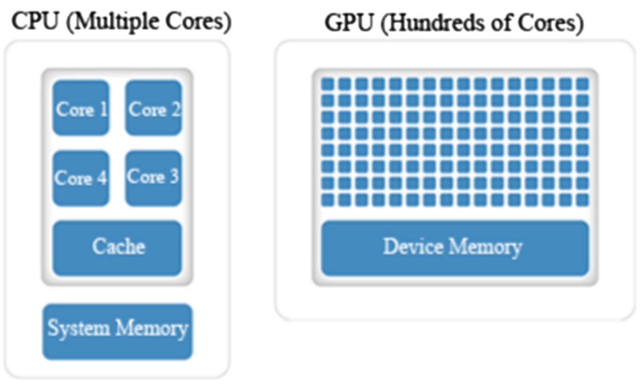

Fig. 1 Comparison of the number of cores on a CPU system and a GPU

Until the emergence of CUDA programming, parallel programming was carried out through the CPU. However, CPUs are originally serial processors thus complex software is required to use a combination of more than one CPU. With CUDA, parallel processing became easier with one of the most important reasons being that CUDA provides for parallelism manually. The program part written in CUDA is called a kernel. GPU runs thousands of copies of this core and makes it parallel. Since CUDA is an extension of the C language, transforming a program written through the CPU is easy.

There are some studies that investigated the reduction of the computational time with CUDA. Jang et al. [16] used CUDA and OpenMP to accelerate the neural networks. They worked on creating a text detection application and they described the challenges and obstacles that they faced in the process. They presented some sample codes related to the application. As a result, they presented the positive effects and benefits of GPU programming. Canto et al. [17] used parallel neural network training with CUDA-basic linear algebra subroutines (CUBLAS). The CPU and CUDA applications were compared in experiments performed with different numbers of hidden layer neurons. They presented a performance comparison of parallel and serial programming. He et al. [18] used standard GPU features to train parallel neural networks with CUDA They obtained good performances by applying matrix and vector operations on the GPU. Lahabar et al. [19] proposed a CUDA-based parallel neural network for pattern recognition. They compared the Matlab (CPU) version and their GPU application and reported the accelerations on the GPU.

In this paper we report on the application of CUDA and explored the features in relation to a parallelized simulation of a neural network based the automatic classification of anesthetic depth levels. The process was simplified by using the CUBLAS. Since CUBLAS does not have all the necessary libraries, the kernel function was also used. Two different graphics cards were applied to different configurations of neural networks in the experiments. Hence, CUDA programming had a positive effect. Several advantages of the proposed method are as follows:

- In this study, in contrast to the aforementioned studies, a parallel running neural network was implemented on EEG data.
- Since the access speed of general memory is slower than shared memory, active use of shared memory was targeted.
- The proposed system aims to produce faster results in the real-life implementation of biomedical signal processing applications to be used in real-time.
- Selecting various features from different categories is important in terms of obtaining a better representation of the incoming data thus achieving a successful performance.

Background

Multilayer perceptron neural network

The term feed forward back-propagation network has been coined because the error reduction process is carried out by spreading the error over the entire network. This is also called the back-propagation of error. This algorithm attempts to reduce the error value, (the difference between the actual obtained output and what the output should be) to a minimum level gradationally by reflecting it onto all weights. In the back-propagation algorithm, training starts with a random set of weights. The algorithm for Q-layer feed-forward network [20, 21] is;

$q=1,2,3,\ldots,Q$ layer number, the input of unit at the layer, the input of i unit at the layer, the weight value which connects unit at the layer to the unit at the layer. The processing steps are as follows:

Step 1: Real-valued small random numbers are assigned to as the initial value.

Step 2: A random (input-target) working model is selected and the forward output values are calculated for each unit at the layer. So, the output is obtained by the formula given in Eq. (1).

$$y_i^q = f\left(\sum_i y_i^{q-1} w_{ij}^q\right) \tag{1}$$

Step 3: The error terms are calculated for output units.

$$\delta_i^Q = (y_i^q - y_i^p) f'\left(X_i^Q\right) \tag{2}$$

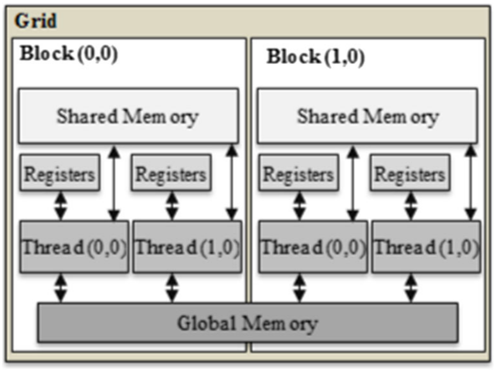

Fig. 2 CUDA programming model [15]

Step 4: For all units at the layer, the error terms are calculated for hidden layer units via the back-propagation method.

$$\delta_i^{q-1} = f'\left(X_i^{q-1}\right)\sum_i \delta_i^q w_i^q \tag{3}$$

Step 5: The Weight values are updated according to the following equations.

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}^q \tag{4}$$

Using the learning coefficient, the change made in weight during the learning phase should be according to the following equation.

$$\Delta w_{ij}^q = \eta \delta_i^q y_i^{q-1} \tag{5}$$

Step 6: Returning to step 2 the processes are repeated for each model, until the total error reaches an acceptable level.

CUDA programming

In software developed by utilizing CUDA programming, a portion of the written codes works on the CPU, and when parallelism is desired the other codes are processed in the GPU. The kernel function is the part of

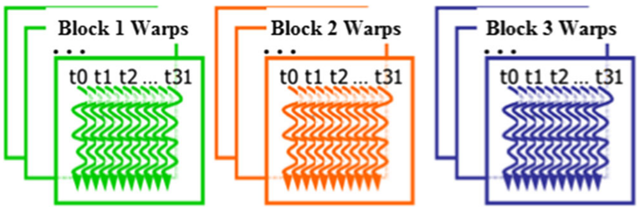

Fig. 3 Sample warp scheduler

Table 1 The age and weight of the patients

	Range	Mean	Standard deviation
Age	30–60	47.5	26.3
Weight	45–88	69.5	13.9

code which allows codes to work on the GPU. Before running this function, it is necessary to determine the values of certain parameters which are the number of blocks and threads. A thread is a basic execution unit paired with a single GPU core. A block structure is formed from a combination of a certain number of threads. A structure consisting of blocks and threads is called a grid structure. This grid structure has a great importance in the concurrent execution of threads on the GPU [15]. Grids are joined and create a graphics processor as shown in Fig. 2.

GPUs have different memory locations. The global memory is a basic communication area between the CPU and GPU; it is accessible from all the threads but has a long latency time. Each streaming processor includes a small shared memory shared by all the streaming processors (SP). This memory is very fast and all the variables defined in this memory can be accessed by all blocks. Apart from these two memories, there are also constant memory and texture memory which are specifically designed for different purposes [15].

The working principle of the threads is defined by warp schedulers in a grid structure. The threads in a streaming processor are sent to the streaming processor by warp scheduler in groups of 32. Figure 3 shows that the sample warp scheduler distributes threads to CUDA cores in a grid structure consisting of three blocks with 32 threads in each block.

A brief outline of the CUDA programming steps are; identification of the kernel function, identification and running of threads by GPU according to this kernel function, packaging of threads in groups of 32 called warp and the execution of these warps by the multi-thread processors called streaming multiprocessor.

Materials

Data set

The application was performed on a database that contains raw, long-term, and continuous records related to anesthetic data. The data was obtained from the hospital of the Yildirim Beyazit University. This study consisted of 20 cases in which the isoflurane anesthetic drug was administered. Different surgical operations were undertaken and the duration of the anesthesia varied. The age and weight of the patients are given in Table 1.

Data preparation

The EEG signal was recorded according to 10/20 electrode placement system from 15 channels (Fp1, Fp2, F7, F3, Fz, F4, F8, T7, C3, Cz, C4, T8, P3, Pz, P4). The EEG signal was filtered between 0.1 and 60 Hz and digitized with sampling rate of 256 Hz. In this study, applications were conducted in order to determine six levels of the depth of anesthesia. The BIS index was also recorded synchronously from the depth of anesthesia monitoring device (BIS^{TM}, Aspect Medical Systems) in order to compare with extracted attributes. This device consists of levels between 0 and 100. The anesthesia levels in this study and the corresponding levels of BIS values are presented in Table 2.

After gathering the data, filtering stage was taken place. First, the EEG signals was filtered with a 10 point noncausal moving average filter to smooth the EEG signals. A band pass filter (10th order IIR Butterworth band pass filter) was utilized to remove the effects of noise and artifacts from the signals. The frequency range of the filter was 0.1–60 Hz. In addition, a Notch filter was applied to eliminate the noise from the 50 Hz mains. In the next stage, EEG

Table 2 Anesthetic depth levels in relation to the BIS Index

Anesthetic depth levels	Durum	Aralık Değerleri	Seviye
Deep anesthesia	Cortical neuron suppression is increasing.	BIS Index: 0–25	6
		BIS Index: 25–50	5
Moderate anesthesia	Surgical anesthesia. Less likely to remember after surgery, visual processes and reflex movements available	BIS Index: 40–50	4
		BIS Index: 50–60	3
Light anesthesia	Increased sedation and memory failure situation, patient can be awakened with a stimulation.	BIS Index: 60–80	2
Awake	The patient is awake, aware, memory and conscious recall are complete.	BIS Index: 80–100	1

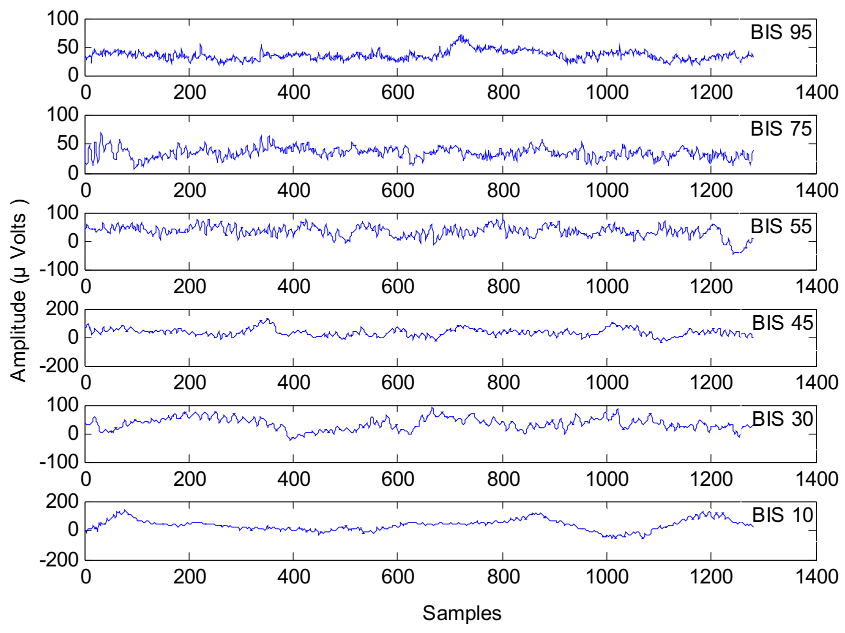

Fig. 4 EEG signals from different states of anesthesia

signals were divided into 10 sec epochs with each segment being passed through the windowing process using Hamming window. The EEG signals obtained from different levels of anesthesia are given in Fig. 4.

As it is shown from Fig. 4 that EEG waves show different characteristics, especially in frequency bands (alpha, beta, delta and theta), among depth of anesthesia levels. Low amplitude and mixed EEG frequency are apparent during the awake level. In the light anesthesia level, the highest amplitude with a frequency range of 2–7 Hz and the existence of alpha waves (7.5–12.5 Hz) are found. The beta waves (12.5–30 Hz) are observed during the moderate anesthesia. The deep anesthesia, may consist of low frequency waves which are lower than 7.5 Hz (delta and theta bands).

Feature extraction

This stage involves determining the significant information from the EEG signal. The features were given from four different categories (time, non-linear, frequency-based and entropy) in the feature extraction. Detailed information about these features and their descriptions can be found in [22, 23]. Forty four features were applied as inputs to the neural networks are given in Table 3.

Experimental setup

The experiments were carried out on the graphics cards Nvidia GeForce GT 525 M with GF108 architecture and Nvidia Quadro 2000 with GF106 architecture, and Intel Core i7-2670QM 2.20 GHz CPU with an 8GB memory. For comparisons to be equal, the software was coded in the same programming language. Also in the coding, a single precision representation was chosen. The developed software was created in Visual C++ 2010. Table 4 summarizes the hardware used in this study.

The GF108 architecture has 585 million transistors and 96 CUDA cores. The GF106 architecture has 1.17 billion transistors and 192 CUDA cores. Figure 5 shows

Table 3 Features selected as inputs to the NNs

Category name	Features
Time based features	Arithmetic mean, Maximum value, Minimum value, Standard deviation, Variance, Median, Zero crossings, Skewness, Kurtosis.
Entropy based features	Petrosian fractal dimension, Rényi entropy, Spectral entropy, Permutation entropy, Approximate entropy.
Frequency-based features	Wigner ville coefficients (4 Features), Wavelet transform based features (16 Features).
Non-linear based features	Hjorth Parameters (3 Features), Mean curve length, Hurst exponent, Mean energy, Mean teager energy.

Table 4 Summary of hardware features for the CPU and the GPU used

Processor	CPU (Intel)	GPU 1	GPU 2
Commercial model	Core i7-2670QM	Nvidia GeForce GT 525 M	Nvidia Quadro 2000
Number of cores @ speed	4 @ 2.2 GHz	96 @ 600 MHz	192 @ 625 MHz
Memory speed	1,333 MHz	900 MHz	1,300 MHz
Memory bus width	128 bit	128 bit	128 bit
Memory bandwidth	21.3 GB/s	28.8 GB/s	41.6 GB/s
Memory size (type)	8 GB (DDR3)	1GB (DDR3)	1GB (GDDR5)
Bus from/to CPU	Does not apply	PCI-e 2.0 x 16	PCI-e 2.0 x 16

the architecture of Nvidia Quadro 2000 with an enlarged panel showing the streaming processors.

The total CUDA cores (192) are clustered within 4 streaming processors, each one containing 48 cores. The graphics card gives memory support up to 1 GB of use. With 48 KB shared memory information can be shared between blocks. Streaming processor has eight special function units (SFU) which are used to carry out special mathematical operations. The SFU executes the commands that carry out operations such as sine, cosine, square root and interpolation. Each SFU executes one command per hour, for each thread. In each core, there is integer unit (INT) for integer operations and floating point unit (FP) for decimal operations.

Implementation of the CUDA to neural network

The mathematical calculations of neural networks, mostly consist of matrix multiplication operations. CUDA can process matrix multiplication efficiently using the shared memory per block. In applications, shared memory increases the efficiency significantly. For instance, in GPU general processes, 400–600 cycles are needed to access global memory, however, CUDA only needs 4 cycles to access the shared memory [25]. Therefore, CUDA′s shared memory is important in terms of the efficient execution of processing. Apart from matrix operations, the calculation of the activation function in each hidden neuron can be performed in parallel. Thus, threads can be created in equal numbers with the element number of the

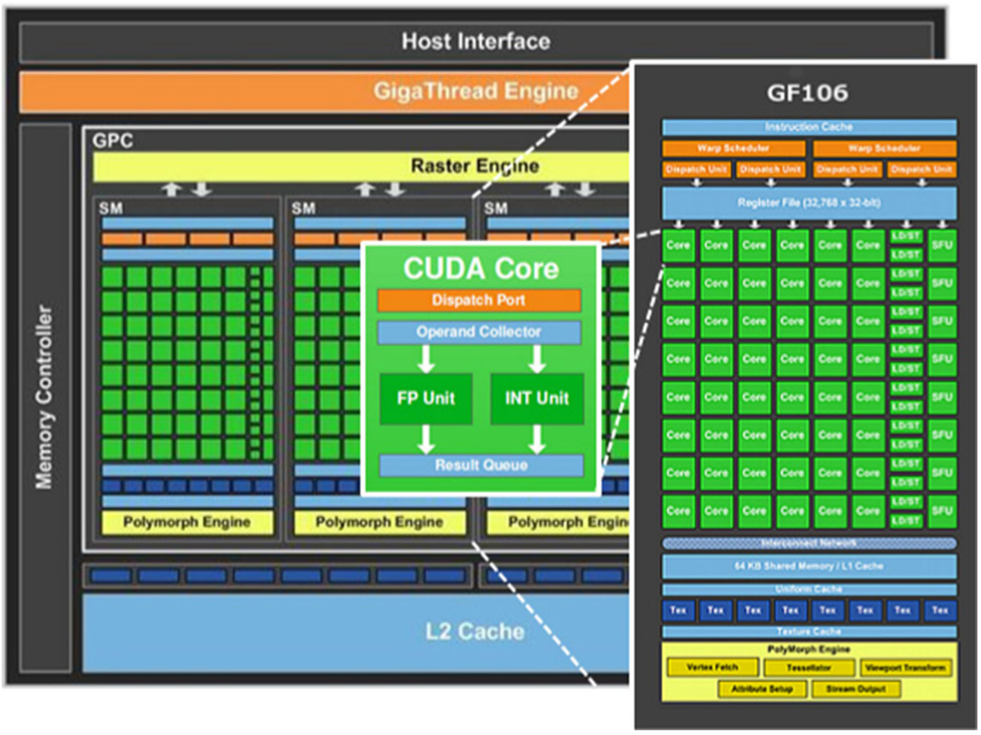

Fig. 5 Architecture of Nvidia Quadro 2000 [24]

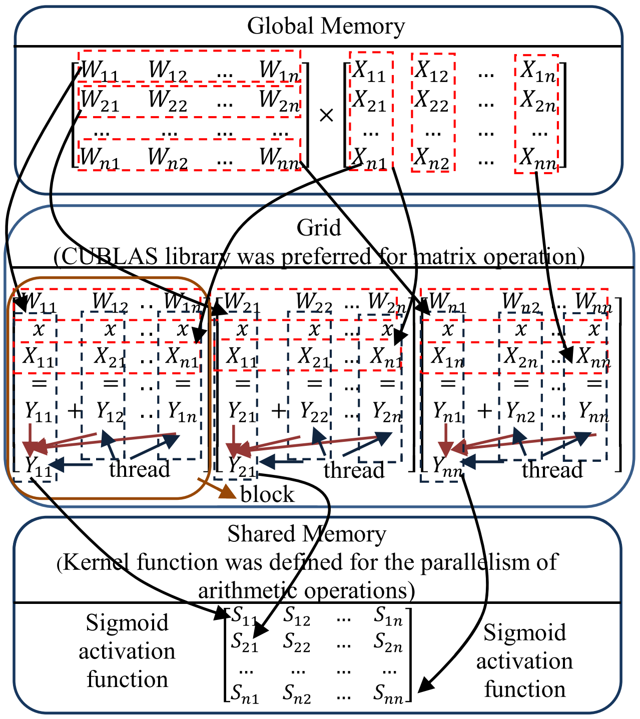
Global Memory
Grid
(CUBLAS library was preferred for matrix operation)
thread
thread
thread
block
Shared Memory
(Kernel function was defined for the parallelism of arithmetic operations)
Sigmoid activation function
Sigmoid activation function

Fig. 6 Matrix multiplication operations of neural network using CUDA (a section of forward computation steps)

matrix. Then the processes can be calculated independently in each thread [25].

The matrix operations of a portion of forward neural network calculations steps performed on CUDA are presented in Fig. 6. As shown in figure that the neural network has a suitable structure for parallel processing.

In this study, the parallel application of the back propagation neural network algorithm was carried out in 2 steps. In the first step, the matrix operations are parallelized and in the second step, the arithmetic operations are parallelized. The input values and multiplying weight values between the "input and hidden layers values" are examples of matrix operations. An example of the arithmetic operations is the calculation of the sigmoid function and weight updates. In the parallelization of the matrix operations (see Fig. 6), CUBLAS library was chosen since it offers a serial algorithm structured according to the optimal parallelism, the identification of the directories and blocks do not constitute a problem. For the arithmetic operations, parallelism was realized with the identification of the kernel which is the name given to the function running on the GPU. Examples of these two steps are given below.

1. The *cublasSgemm* function of the CUBLAS library was used for matrix multiplication operations. An example of its use is as follows.

$$'cublasSgemm\left(char\, transa,\, char\, transb,\, int\, n,\, int\, k,\, float\, alpha,\, const\, float\, *A,\, int\, lda,\, const\, float\, *B,\, int\, ldb,\, float\, beta,\, float\, *C,\, int\, ldc\right)'$$

where, and are the values of the matrix and are scalar values.

According to this function, matrix is in size, matrix is in size and matrix is in size. Function *cublasSgemm*, calculates the matrix operation. When taken as and, the process is then reduced to . The constant values are and . The expression of trans in functions is used for matrix transpose processes. Generally *'n'* structure is chosen. The usage is as follows.

$$if\ transa = 'N'\ or\ 'n'\ A = A,$$

$$if\ transa = 'T', 't', 'C'\ or\ 'c'\ then\ A = A^T$$

transb is used for matrix for the same purpose.

Apart from the *cublasSgemm* function, in this study the following functions are used; *cublasAlloc* for the separation of GPU memory, *cublasFree* to leave allocated memory space again, *CublasSetVector* for vector transferring from CPU to GPU, *CublasGetVector* for vector transferring from GPU to CPU, *CublasSetMatrix* for transferring matrix from CPU to GPU and *CublasGetMatrix* functions for transferring matrix from GPU to CPU.

Step 2: The kernel was defined for the parallelism of arithmetic operations and kernel functions have been developed for the activation function, the error function and weight updates. Figure 7 gives a sample of the kernel operations regarding the sigmoid activation function performed on the GPU.

Fig. 7 Kernel function written for the sigmoid function to run on the GPU

```
__global__ void sigmoid( float *out1, int maksimum) {
__shared__ float u_shared[32]; // Shared memory allocation
 int column = blockIdx.x * blockDim.x + threadIdx.x; // Address information that GPU-index (thread) will
                                                       process on.

if (column<maximum){
u_shared[column] = d_an[column]; // Writing to shared memory
u_shared[column]  = (1/(1+__expf(-1*u_shared[column]))); // Calculation of the sigmoid function value
__syncthreads();  } // The __syncthreads()  command is a block level synchronization barrier.
out1[column]=u_shared[column]; // Writing to global memory }
```

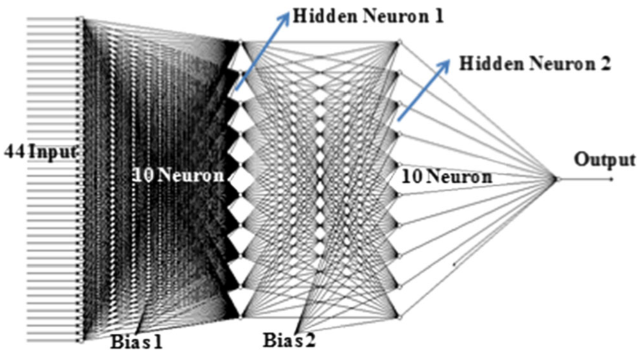

Fig. 8 ANN with 44-10-10-1 architecture

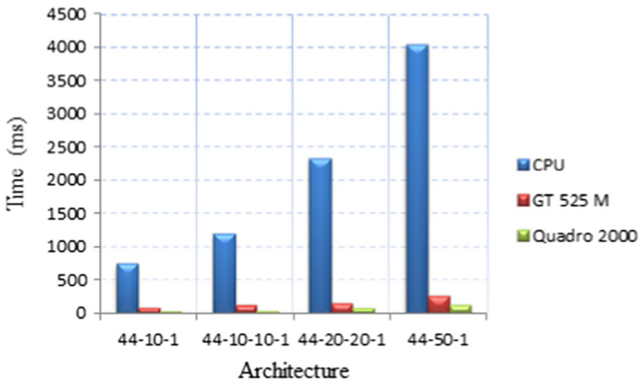

Fig. 9 Speed performances obtained with different architectures

In the scope of the study there were various optimization processes were performed on the neural network in order to resolve application to the GPU. The most important operations performed in software features for the CPU and the GPU are listed below:

- Code Partitions that can be made in parallel were run on CUDA, else were run on CPU.
- Shared memory usage was provided significant increase in performance during calculations used in functions.
- System bottlenecks are resolved using Nvidia Visual Profiler.
- To provide a better performance with CUDA Programming, each block contained at least 64 threads. Threads are executed in groups of 32 called a warp which executes a command at a time. Therefore, to achieve full efficiency, all of 32 threads in the warp must have the same execution path.
- A few CUDA-optimized libraries such as CUBLAS were utilized.

Detection of anesthetic depth level

The detection of anesthetic depth level experiments was performed in 4 different scenarios to better observe the speed of the operations. Each scenario testing with different ANN architectures involving different number of hidden layer (s) and neuron (s). An example of *ANN* with 44-10-10-1 architecture is presented in Fig. 8.

To achieve good results in all the experiments the learning coefficient and momentum coefficient were determined as 0.7. The iteration number was determined as 10,000. The scenarios were:

Scenario 1:	The neural network architecture was designed according to the following values; an input value of 44, the number of hidden layer neurons was 10 and the output value was 1.
Scenario 2:	The neural network architecture was designed according to the values; an input value of 44, the number of hidden layer-1 neurons was 10, the number of hidden layer-2 neurons was 10 and the output value was 1.
Scenario 3:	The neural network architecture was designed according to the values; an input value of 44, the number of hidden layer-1 neurons was 20, the number of hidden layer-2 neurons was 20 and output value was 1.
Scenario 4:	The neural network architecture was designed according to the values; an input value of 44, the number of hidden layer neurons was 50 and output value was 1.

Table 5 Speed performances obtained with different NN architectures

Architecture	Criteria	CPU	GT 525 M	Quadro 2000
44-10-1	Time	739 sn	90 sn	30 sn
	Speedup		8.2 x	24.6 x
44-10-10-1	Time	1,188 sn	120 sn	42 sn
	Speedup		9.9 x	28.2 x
44-20-20-1	Time	2,338 sn	160 sn	80 sn
	Speedup		14.6 x	29.2 x
44–50-1	Time	4,048 sn	265 sn	125 sn
	Speedup		15.2 x	32.3 sn

Table 6 The accuracy rate obtained with different ANN architectures

Architectures	Processors		
	CPU	GT 525 M	Quadro 2000
44-10-1	93.7 %	93.8 %	94.01 %
44-10-10-1	99.01 %	99.05 %	98.93 %
44-20-20-1	95.4 %	95.7 %	95. 5 %
44-50-1	88.75 %	87.09 %	88.08 %

Table 7 Scoring agreement between BIS scoring and the proposed method

	BIS score						
Proposed method	Anesthetic depth levels	Deep anesthesia (0–25)	Deep anesthesia (25–50)	Moderate anesthesia (40–50)	Moderate anesthesia (50–60)	Light anesthesia (60–80)	Awake (80–100)
	Deep anesthesia (0–25)	729	3	0	0	0	0
	Deep anesthesia (25–50)	4	771	4	0	0	0
	Moderate anesthesia (40–50)	2	3	638	2	0	0
	Moderate anesthesia (50–60)	0	0	3	582	6	3
	Light anesthesia (60–80)	0	0	0	1	604	5
	Awake (80–100)	0	0	0	0	2	642
Accuracy (%)		99.18	99.22	98.91	99.48	98.69	98.76
Overall accuracy (%)							99.05

Table 8 Quantitative comparison of the current study with the reported methods

Authors	Anesthetic levels	Method	Accuracy rate
Srinivasan et al. [27]	Low, Medium, High	Feature extraction: • Normalized spectral entropy Classification Algorithms: • Recurrent artificial neural network	99.6 %
Zhang et al. [28]	Awake, Asleep	Complexity based on Lempel-Ziv (CBLZ) Approximate Entropy (ApEN) Spectral Entropy (SE) Median Frequency (MF)	CBLZ: 93 % ApEN: 89 % SE: 76 % MF: 64 %
Nicolaou et al. [29]	Awake, Anesthetized	Feature extraction: • Granger Causality Classification Algorithms: • SVM	98 %
Esmaeili et al. [30]	awake, moderate anesthesia, surgical anesthesia, isoelectric	Feature extraction: • Spectral features • Burst suppression ratio Classification Algorithms: • Fuzzy rule-base index (FRI) • An adaptive network-based fuzzy inference system (ANFIS) • Linear discriminant analysis (LDA)	FRI: 96.75 % ANFIS: 94.33 % LDA: 91.42 %
Lalitha et al. [31]	Low, Medium, High	Feature extraction • Correlation dimension (CD) • Lyapunov Exponent (LE) • Hurst Exponent (HE) Classification Algorithms: • Elman neural network (EN) • Multilayer perceptron (MLP)	CD+EN: 97.8 % CD+LE+EN:97.5 % LE+EN: 99 % LE+MLP: 95 %
Rabbani et al. [32]	$90<BIS<100$, $80<BIS<90$ $70<BIS<80$, $60<BIS<70$ $50<BIS<60$, $40<BIS<50$ $30<BIS<40$, $20<BIS<30$ $10<BIS<20$, $0<BIS<10$	Feature extraction • Complex wavelet transform Classification algorithms • ANFIS	94.07 %
Our work	Deep Anesthesia (0–25) Deep Anesthesia (25–50) Moderate Anesthesia (40–50) Moderate Anesthesia (50–60) Light Anesthesia (60–80) Awake (80–100)	Feature extraction • Attributes given in Table 3 Classification Algorithms: • Feed Forward Neural Network (with Backpropagation algorithm)	99.05 %

Experimental results about speed performances for these architectures are presented in Table 5.

Input matrix was given in Eq. (6) showing that the data set consists of 44 features and 3,113 examples.

$$input_{matrix}([44x3113]) = \begin{bmatrix} X_{1,1} & \cdots & X_{1,3113} \\ \vdots & \ddots & \vdots \\ X_{44,1} & \cdots & X_{44,3113} \end{bmatrix} \quad (6)$$

Speedup calculation was calculated as a fraction of CPU time (T_{CPU}) and the GPU time (T_{GPU}). Speedup is calculated as shown in Eq. (7). Time calculation T_{CPU} and T_{GPU} were calculated as the statistical median of ten sequential running calculations, hence eliminating the random error [26].

$$Speedup = T_{CPU}/T_{GPU} \quad (7)$$

The experiments were carried out by changing the number of hidden layers and number of neurons in the hidden layers in order to test the performance of the GPU. Better results were obtained with Nvidia Quadro 2000 graphics card which has more cores. In Fig. 9, the results obtained in 4 different scenarios are presented. As can be seen from the chart, the same problem was solved faster with GPU programming.

When examining the results, it can be seen that proposed method quickly classifies the EEG data with high accuracy values. Furthermore, better results are obtained with the increase in the number of CUDA cores in graphics cards. A success performance analysis of the proposed methods was performed in the final stage. Better results were obtained with 44-10-10-1 ANN architecture. Accordingly, the accuracy rates that were obtained are presented in Table 6.

It is expected that proposed method will provide support to anesthetists in the classification of the depth of anesthesia. The errors in each stage can be determined by exploring the confusion matrix, as shown in Table 7. This indicates that there is an agreement between the proposed method and the BIS values.

The performance of the proposed method was compared with the recent studies available from the literature listed in Table 8. This table shows that the performance of the automatic classification of anesthetic depth level implementations, including 5 or above anesthesia level, was generally in the range of 64–97 %. Many of the studies, which obtained 95 % or higher compatibility on 3 or 4 anesthetic depth levels. It is believed that this study provides substantially contribution to the field, since it gave a 99.05 % accuracy rate on 6 anesthetic depth levels.

In this study, the BIS value was used as outcome for performance test of the system. The results of the present study demonstrate that the classification of anesthetic depth levels were obtained very quickly. Nevertheless, the fact that the BIS itself alone may not be representative of anesthetic depth should be considered.

Results

This study presents an automated method to classify the depth of anesthesia in a short time with high accuracy rate by analyzing EEG signals. This classification allows the anesthetists to make decisions concerning the level of anesthesia required.

The effect of CPU and GPU on processing times of neural networks and success rates were investigated with the proposed method. The use of CUDA technology on the neural network algorithm accelerated the processing time allowing the EEG signals to be classified in a shorter time. As a result of different experiments, an average of 20x speed-up was obtained. The success performance of the method was also high. Using a combination of many attributes in different categories leads to a better representation of data using a smaller number of attributes. These attributes are thought to affect the high rate of success.

Providing high accurate and quick results encourage for the future studies will be continue for not only anesthesia but also for other biomedical signal processing problems in real time or need to be run quickly.

Acknowledgments Our thanks to Abdulkadir but from the Department of Anesthesiology and Reanimation in Yildirim Beyazit University, Turkey who provided the anesthesia dataset.

References

1. Miller, R. D., Eriksson, L. I., Fleisher, L. A., Wiener-Kronish, J. P., and Young, W. L., *Miller's Anesthesia 2 volume set*, 7th edition. Churchill, Livingstone, 2010.
2. Shulman, M., Braverman, B., and Ivankovich, A. D., Sevoflurane triggers malignant hypertermia in swine. *Anesthesiology.* 54(3): 259–260, 1981.
3. Shalbaf, R., Behnam, H., Sleigh, J. W., Steyn-Ross, A., and Voss, L. J., Monitoring the depth of anesthesia using entropy features and an artificial neural network. *Journal of Neuroscience Methods.* 218(1): 17–24, 2013.
4. Saraoglu, H. M., and Edin, B., E-Nose system for anesthetic dose level detection using artificial neural network. *Journal of Medical Systems.* 31(6):475–482, 2007.
5. Saraoglu, H. M., and Sanli, S., A fuzzy logic-based decision support system on anesthetic depth control for helping anesthetists in surgeries. *Journal of Medical Systems.* 31(6):511–519, 2007.
6. Gunturkun, R., Estimation of medicine amount used anesthesia by an artificial neural network. *Journal of Medical Systems.* 34(5):941–946, 2010.
7. Krol, M., and Reich, D. L., Development of a decision support system to assist anesthesiologists in operating room. *Journal of Medical Systems.* 24(3):141–146, 2000.

8. Gürkan, G., Determining the depth of anesthesia by the analysis of EEG signals, Istanbul University, *PhD Thesis*. 2011.
9. Monk, T. G., Saini, V., Weldon, B. C., and Sigl, J. C., Anesthetic management and one-year mortality after noncardiac surgery. *Anesth Analg.* 100(1):4–102, 2005.
10. Myles, P. S., Leslie, K., McNeil, J., Forbes, A., and Chan, M. T. V., Bispectral index monitoring to prevent awareness during anaesthesia: the B-Aware randomised controlled trial. *The Lancet.* 363(9423): 1757–1763, 2004.
11. Punjasawadwong, Y., Boonjeungmonkol, N., and Phongchiewboon, A., Bispectral index for improving anesthetic delivery and postoperative recovery. *Anesthesia & Analgesia* 106(4):1326, 2008.
12. Liu, J., Singh, H., and White, P. F., Electroencephalographic bispectral index correlates with intraoperative recall and depth of propofol-induced sedation. *Anesth Analg* 84(1):185–9, 1997.
13. Kertai, M. D., Whitlock, E. L., and Avidan, M. S., Brain monitoring with electroencephalography and the electroencephalogram-derived bispectral index during cardiac surgery. *Anesthesia and Analgesia.* 114(3):533–546, 2012.
14. Zoughi, T., Boostani, R., and Deypir, M., A wavelet-based estimating depth of anesthesia. *Engineering Applications of Artificial Intelligence.* 25(8):1710–1722, 2012.
15. CUDA Programming Guide 4.2, http://developer.nvidia.com/cuda (Accessed: 12.01.2014).
16. Jang, H., Park, A., and Jung, K., Neural network implementation using CUDA and OpenMP. *Digital Image Computing: Techniques and Applications (DICTA)*, pp. 155-161, 2008.
17. Canto, X. S., Ramirez, F. M., and Cetina, V. U., Parallel training of a back-propagation neural network using CUDA. *ICMLA*, pp. 307-312, 2010.
18. He, T., Dong, Z., Meng, K., Wang, H., and Oh, Y., Accelerating multi-layer perceptron based short term demand forecasting using graphics processing units, In: *Transmission & Distribution Conference & Exposition: Asia and Pacific*, pp. 1-4, 2009.
19. Lahabar, S., Agrawal, P., and Narayanan, P. J., High performance pattern recognition on GPU. In: *National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG'08)*, pp. 154-159, 2008.
20. Kelesoglu, O., and Akarsu, E. E., *Determination of annual heat loss and requirement of energy in a reinforced concrete building by artificial neural networks, e-Journal of New World Sciences Academy Natural and Applied Sciences, 3(2):381-390*, 2008.
21. Ahmad, F., Mat Isa, N. A., Hussain, Z., and Osman, M. K., Intelligent medical disease diagnosis using improved hybrid genetic algorithm - multilayer perceptron network. *Journal of Medical Systems.* 37(2):1–8, 2013.
22. Sen, B., and Peker, M., Novel approaches for automated epileptic diagnosis using FCBF feature selection and classification algorithms. *Turkish Journal of Electrical Engineering & Computer Sciences.* 21: 2092–2109, 2013.
23. Sen, B., Peker, M., Celebi, F.V., and Cavusoglu, A., A comparative study on classification of sleep stage based on EEG signals using feature selection and classification algorithms. *Journal of Medical Systems*. 38(18), doi: 10.1007/s10916-014-0018-0, 2014.
24. Nvidia GeForce GF106, http://www.tomshardware.com/reviews/geforce-gts-450-gf106-radeon-hd-5750,2734.html (Accessed: 24.10.2014)
25. Lin, J., and Lin, J., Accelerating BP neural network-based image compression by CPU and GPU cooperation. *International Conference on Multimedia Technology. pp.* 40–44, 2010.
26. Kajan, S., and Slačka, J., Computing of neural network on graphics card. In: *International Conference Technical Computing*. ISBN 978-80-970519-0-7, 2010.
27. Srinivasan, V., Eswaran, C., and Sriraam, N., EEG based automated detection of anesthetic levels using a recurrent artificial neural network. *International Journal of Bioelectromagnetism.* 7:267–270, 2005.
28. Zhang, X. S., Roy, R. J., and Jensen, E. W., EEG complexity as a measure of depth of anesthesia for patients. *IEEE Transactions on Biomedical Engineering.* 48(12):1424–33, 2001.
29. Nicolaou, N., Hourris, S., Alexandrou, P., and Georgiou, J., EEG-based automatic classification of 'awake' versus 'anesthetized' state in general anesthesia using granger causality. *PLoS ONE.* 7(3): e33869, 2012. doi:10.1371/journal.pone.0033869.
30. Esmaeili, V., Assareh, A., Shamsollahi, M. B., Moradi, M. H., and Arefian, N. M., Estimating the depth of anesthesia using fuzzy soft computation applied to EEG features. *Intelligent Data Analysis.* 12(4):393–407, 2008.
31. Lalitha, V., and Eswaran, C., Automated detection of anesthetic depth levels using chaotic features with artificial neural networks. *Journal of Medical Systems.* 31(6):445–452, 2007.
32. Rabbani, H., Dehnavi, A. M., and Ghanatbari, M., *Estimation the depth of anesthesia by the use of artificial neural network. Artificial Neural Networks - Methodological Advances and Biomedical Applications*, 2011. doi:10.5772/15139.