CMST 21(4) 191-200 (2015) DOI:10.12921/cmst.2015.21.04.003

Family of Parallel *LMS*-based Adaptive Algorithms of Echo Cancellation

A. Dobrucki[1], **M. Walczyński**[2], **W. Bożejko**[3]

[1]*Wrocław University of Technology*
Faculty of Electronics, Department of Acoustics and Multimedia
Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland

[2]*General Tadeusz Kościuszko Military Academy of Land Forces*
Faculty of Management, Department of System Engineering
ul. Czajkowskiego 109, 51-150 Wrocław, Poland

[3]*Wrocław University of Technology*
Institute of Computer Engineering, Control and Robotics
Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland

E-mails: andrzej.dobrucki@pwr.wroc.pl maciej.walczynski@pwr.wroc.pl wojciech.bozejko@pwr.wroc.pl

Received: 22 September 2014; revised: 29 June 2015; accepted: 08 July 2015; published online: 01 December 2015

Abstract: In this paper we propose a number of new, genuine parallel *LMS*-based adaptive algorithms in the context of their use in the acoustic echo cancellation. The most complex parts of the *LMS*-based algorithms were determined and parallelized. A number of genuine parallelization methods were proposed taking into consideration varied types of architectures of modern concurrent computing environment, such as GPUs, clusters of workstations and cloud computing.

Key words: GPU, DSP, adaptive algorithms, echo cancellation, parallel algorithms

I. INTRODUCTION

The problem of acoustic echo cancellation has been widely discussed in literature (see [1-5]), but mainly in the context of calculation with the use of one processor (sequential). While considering multiprocessor approaches one can enumerate, among others, subband echo cancellation which is a method by its nature predisposed to effective implementation in parallel calculation systems [6].

This paper develops a description of an approach applied in use of parallel and distributed calculation environment both in acoustic echo cancellation [7], and noise elimination [8, 9], pattern search [10, 11] and issues connected with parallel processors work optimization [12]. Herein we present a case study of a parallel single-run algorithm of acoustic echo cancellation. Results can be implemented in the GPU (Graphics Processing Unit) computing environment as well as in VLSI or FPGA architectures.

The application and development of modern technologies results in creating new, better and quicker communication methods. The number of phone calls realised through headsets has recently risen rapidly. The number of conference calls and video conferences is also growing quickly. In view of acoustic specificity of these types of communication that results from weak acoustic isolation between the speaker (speakers) and microphone (microphones) there appears a serious problem of acoustic echo occurrence. As long as for most users of tele- and videoconference systems the occurrence of their own, slightly delayed voice in the "receiver" is a neutral or even desirable phenomenon, delays of tens of milliseconds [13] make the conversation arduous, and in case of 250 ms delays simply impossible [14]. For this reason it seems really important to use an effective method of echo damping or cancellation. Figure 1 presents a diagram of acoustic echo cancellation system using the adaptive filtering algorithm.

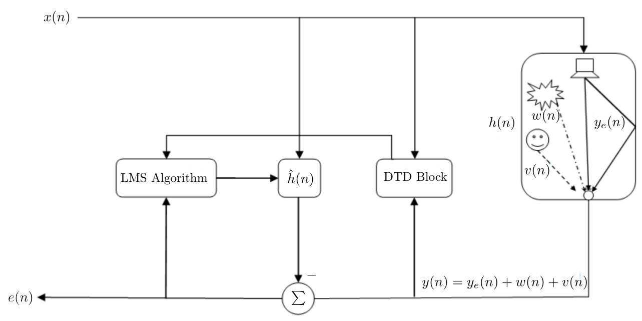

Fig. 1. Diagram of acoustic echo cancellation system using *LMS* algorithm

In this paper we shall propose parallel schemes of the *LMS* algorithm execution in the *GPGPU* environment. The closest theoretical model of such calculation environment assuming processor access to shared memory in short time is the parallel random access computer (*PRAM*). This model shall be discussed in the next chapter and then used to carry out theoretical analysis of parameters (speedup S_p, efficiency E_p, for p processors) of the proposed methods.

II. LMS FAMILY ALGORITHMS

LMS Algorithm. The *LMS* (*Least Mean Square*) related filters are used extremely frequently in the process of echo cancellation. These filters constitute a group of gradient adaptive filters, realizing the strategy of temporal minimization, but not expected error value. The function of cost takes the form:

$$J = J(n) = e^2(n), \tag{1}$$

instead of

$$J = E[e^2(n)], \tag{2}$$

where for some discrete random variable Z that can take value z_i with probability p_i, E denotes the expected value which takes the form:

$$E[Z] = \sum_{i=1}^{n} z_i p_i, \tag{3}$$

and $e(n)$ is the error function that can be defined as:

$$e(n) = y(n) - \hat{y}(n). \tag{4}$$

In case of echo cancellation vector $y(n)$ will be simply the signal that comes back to the first of the speakers. The signal can be presented as superposition of three constituent signals according to the following dependence:

$$y(n) = y_e(n) + v(n) + w(n), \tag{5}$$

where $y_e(n)$ is the echo of the first speaker, $v(n)$ contains the speech signal of the second speaker, whereas $w(n)$ is the signal containing noise. The component $v(n)$ occurs only in the case when both speakers talk at the same time. Such a situation is defined as *DT* (*Double Talk*). When the speakers talk concurrently, the correlation between signals $x(n)$ and $y(n)$ is smaller than in case when only the first speaker talks. The detection of concurrent speech of both speakers is a task of *DTD* algorithms (*Double Talk Detection*). When such a situation is detected, the adaptive filter coefficients are "frozen". The adaptation process is then continued after the completion of second speaker's speech.

It is assumed that the echo signal of the first speaker $y_e(n)$ may be presented as follows:

$$y_e(n) = \sum_{i=0}^{N-1} h_i x_i(n) = \boldsymbol{h}^T \boldsymbol{x}(n), \tag{6}$$

where *N* indicates the filter order, whereas h_i is the *i*-th filter coefficient.

The vector of $\boldsymbol{h}$ coefficients is, however, unknown. Therefore, we can define the estimate set by the following equation:

$$\hat{y}(n) = \widehat{\boldsymbol{h}}^T \boldsymbol{x}(n) + v(n) + w(n), \tag{7}$$

where $\hat{y}(n)$ makes the estimation of combined echo signal from the first speaker, noise signal and the second speaker.

Adaptive filtering assumes continuous $\widehat{\boldsymbol{h}}$ vector matching to $\boldsymbol{h}$ vector, so that with each iteration $\widehat{\boldsymbol{h}}$ vector much better approximates $\boldsymbol{h}$ value. However, since $\boldsymbol{h}$ vector is unknown, therefore $\widehat{\boldsymbol{h}}$ and $\boldsymbol{h}$ matching must occur not directly. One of the matching methods is minimizing the error function described in (4).

In this case the $\widehat{\boldsymbol{h}}$ coefficients adaptation consists in such a correction that would make the allowance proportional to the cost function gradient, but with an opposite sign. Hence the equation determining the filter coefficient update takes the following form:

$$\widehat{\boldsymbol{h}}(n+1) = \widehat{\boldsymbol{h}}(n) + \triangle\widehat{\boldsymbol{h}}(n) = \widehat{\boldsymbol{h}}(n) - \frac{\partial J(\widehat{\boldsymbol{h}}(n))}{\partial \widehat{\boldsymbol{h}}(n)}. \tag{8}$$

In many cases the equation (8) is expanded to the form of:

$$\widehat{\boldsymbol{h}}(n+1) = \widehat{\boldsymbol{h}}(n) - \frac{1}{2}\mu(n)\boldsymbol{W}(n)\frac{\partial J(\widehat{\boldsymbol{h}}(n))}{\partial \widehat{\boldsymbol{h}}(n)}, \tag{9}$$

where time variable scale factor $\mu(n)$ determines the rate of $\triangle\widehat{\boldsymbol{h}}(n)$ vector change, whereas $\boldsymbol{W}(n)$ matrix is responsible for the adaptation process rate.

In case of classical *LMS* filter the weight matrix $\boldsymbol{W}(n)$ is replaced by a diagonal unit matrix $\boldsymbol{I}$:

$$\boldsymbol{W}(n) = \boldsymbol{I}, \tag{10}$$

whereas the scalar scale coefficient remains time-constant:

$$\mu(n) = \mu = \text{const.} \tag{11}$$

In case of algorithms based on temporal value of error square value, where the cost function is defined with the equation (1), derivative of this function in relation to filter coefficient changes is given with the formula:

$$\frac{\partial J(\widehat{\boldsymbol{h}}\,(n))}{\partial \widehat{\boldsymbol{h}}\,(n)} = \left[\frac{\partial e^2}{\partial \hat{h}_0(n)}, \frac{\partial e^2}{\partial \hat{h}_1(n)}, \ldots, \frac{\partial e^2}{\partial \hat{h}_{N-1}(n)}\right]^{\mathrm{T}}, \tag{12}$$

where N designates filter order.

Moreover, the value of partial derivative in relation to k-th filter coefficient can be presented as:

$$\begin{aligned}&\frac{\partial e^2\,(n)}{\partial \hat{h}_k\,(n)} = 2e\,(n)\,\frac{\partial e\,(n)}{\partial \hat{h}_k\,(n)} = \\ &= 2e\,(n)\,\frac{\partial (y\,(n) - \sum_{k=0}^{N-1}\,(\hat{h}_k x(n-k))}{\partial \hat{h}_k\,(n)} = \\ &= -\,2e\,(n)\,x\,(n-k)\,,\end{aligned} \tag{13}$$

where the expression $\sum_{k=0}^{N-1} \hat{h}_k x(n-k)$ is the estimator of $y(n)$ signal.

Eventually the equation determining the filter coefficient value in the next iteration in case of *LMS* algorithm takes the following form:

$$\widehat{\boldsymbol{h}}\,(n+1) = \widehat{\boldsymbol{h}}\,(n) + \mu \boldsymbol{x}\,(n)\,e(n). \tag{14}$$

The advantage of the classical *LMS* algorithm is the simplicity of its implementation and stability, while its biggest disadvantage is slow convergence. In order to improve this feature numerous modifications have been created to improve the convergence rate through insignificant increase of the number of operations.

NLMS Algorithm. The *NLMS* (*Normalized Least Mean Square*) is a variant of *LMS*. The main difference between those two algorithms is that the *NLMS* contain a time varying scale coefficient which is proportionate to the magnitude of $\boldsymbol{x}$ signal. The value of scale coefficient is given $\mu\,(n)$ by following formula:

$$\mu\,(n) = \frac{\mu_0}{\boldsymbol{x}^T(n)\boldsymbol{x}(n)}. \tag{15}$$

For the *NLMS* algorithm the equation determining filter coefficient value in the next iteration takes the following form:

$$\widehat{\boldsymbol{h}}\,(n+1) = \widehat{\boldsymbol{h}}\,(n) + \frac{\mu_0}{\boldsymbol{x}^T(n)\boldsymbol{x}(n)}\boldsymbol{x}\,(n)\,e(n). \tag{16}$$

PNLMS Algorithm. The *PNLMS* (*Proportionate Normalized Least Mean Square*) is a variant of *NLMS*. This algorithm is a modification of the *NLMS* algorithm that includes a set of individual scale coefficients which are calculated individually for each filter coefficient. For the *PNLMS* algorithm the equation determining filter coefficient value in the next iteration takes the following form:

$$\begin{aligned}\widehat{\boldsymbol{h}}\,(n) &= \widehat{\boldsymbol{h}}\,(n-1) \\ &+ \frac{\mu}{\gamma + \boldsymbol{x}^T\,(n)\,\boldsymbol{G}\,(n)\,\boldsymbol{x}\,(n)}\boldsymbol{G}(n)\boldsymbol{x}\,(n)\,e(n),\end{aligned} \tag{17}$$

where $\boldsymbol{G}(n)$ is a diagonal matrix that adjust contains individual scale coefficients for taps of filters, and γ is a scalar parameter that prevents division by zero and stabilizes the solution.

III. PARALLEL COMPUTER MODEL

The theoretical model of parallel computer being expansion of a sequential *RAM* model is called *PRAM* (*Parallel Random Access Machine*) and is extraordinarily close to practical realizations of *GPU* (Graphics Processing Unit) cards. The model includes a set of n-identical *RAM* machines. Each of them (they are numbered with consecutive natural numbers) has its unique index and its own, local register set. Moreover, as a part of *PRAM* computer functions a set of global registers enabling storing any integer number. Operation of the *PRAM* computer begins with positioning of ian nput set in the initial cells of global register. Originally only one processor is active. Each stage of performance of momentarily active processors may consist of performing in parallel and synchronically one operation from the standard set of instructions or activating successive not active processor that in the next step shall perform the instructions as any activated processor.

We should also notice that since in the primary *PRAM* model none of the processors is privileged in relation to others there are possible conflicts in parallel access to common memory. When two or more processors tried to read or write to one and the same memory cell at the same time, the systems could become unstable. Due to this reason in the primary *PRAM* model it was forbidden to refer to the same storage area simultaneously by more than one processor. Later, however, the situation of parallel access to the same area of common memory has been solved. In view of the organization method for simultaneous access to memory to write and read, the *PRAM* computers can be divided into 4 categories: *EREW* (*Exclusive Read Exclusive Write*), *CREW* (*Concurrent Read Exclusive Write*), *ERCW* (*Exclusive Read Concurrent Write*) and *CRCW* (*Concurrent Read Concurrent Write*). Although the *EREW PRAM* computer model has the biggest limitations among the four basic types of *PRAM* machines, in view of practical difficulties that may occur during the trial of simultaneous write and read from the same memory cells, the actual realizations are close to that *PRAM* machine model. Nevertheless, there exist a series of solutions enabling simulation of *CREW*, *ERCW* and *CRCW PRAM* machines with the use of *EREW PRAM* model.

While testing the efficiency of parallel algorithms implemented for the *PRAM* model basic assumptions is made, that the time measure of their performance may be the number of parallel references to memory during program execution [15]. For simplicity it is adopted, that access time of one processor to memory is unit time. This assumption results directly from the fact, that for the *RAM* machine the time of algorithm execution measured as the number of references to memory is in limit (asymptotic) the same, as the time measured with any other measure.

Additionally, we will need following definitions of speedup and efficiency:

Definition 1. Speedup. Let us consider a problem P a parallel algorithm A_p and a parallel machine M with q identical processors. Let us define by $T_{Ap,M}(p)$ the time of calculations the algorithm A_p needs to solve the problem P on the machine M making use of $p \leq q$ processors. Let T_{As} be the time of calculations needed by the best (the fastest) known sequential algorithm A_s which solves the same problem P on the sequential machine with the processor identical to processors of the parallel machine M. We define the *speedup* as

$$S_p = \frac{T_{A_s}}{T_{A_p,M}(p)}.$$

Definition 2. Efficiency. The *efficiency* E_p of the parallel algorithm A_p executed on the parallel machine M is defined as

$$E_p = \frac{S_p}{p}.$$

and describes an average fraction of time used by each processor effectively. The value of efficiency belongs to the range $[0, 1]$.

IV. PARALLEL LMS ALGORITHM

A single-run algorithm of acoustic echo cancellation shall be called here such an algorithm, in which the filtering process is done one-time on a given set of data.

In this paper the following designations have been accepted:

$x(n)$ – the input of adaptive filter, containing the value of far-end speaker's signal in n-th time moment,

$y(n)$ – the signal containing the echo of far-end speaker's speech and possibly near-end speaker's speech and noises in n-th time moment.

$\hat{y}\,(n)$ – output of adaptive filter containing adaptively filtered signal x, being the estimate of echo signal in n-th time moment,

$e\,(n) = y\,(n) - \hat{y}(n)$ – the value of the error function in n-th time moment,

N – the number of weight coefficients of adaptive filter,

μ – the coefficient of adaptation rate ($0 < \mu < 1$),

$\widehat{\boldsymbol{h}}(n)$ – weight coefficients of adaptive filter in n-th time moment.

IV. 1. Single-run LMS algorithm

The first of single-run algorithms of echo cancellation that shall be discussed here is the classical sequential *LMS* algorithm. This algorithm has a strict sequential character and can be executed on a single-processor *RAM* machine.

Theorem 1. The *LMS* algorithm may be executed in $O(N)$ time on a single-processor *RAM* machine.

Proof. Step 0 of Algorithm 1 is executed in constant time $O(1)$. Step 1 is also executed in constant time $O(1)$. Summation in step two is executed in time $O(N)$. Step 3 is executed in constant time $O(1)$. Step 4 requires computational expenditure $O(N)$, whereas step 5 is executed again in constant time $O(1)$. The whole determines the time of step two and four execution determining the time of algorithm performance time to $O(N)$.■

The diagram of sequential *LMS* algorithm for *RAM* machine has been presented in Figure 2.

Algorithm 1. LMS algorithm scheme

```
Step 0 $n := 0$;
Step 1 Read input $x(n)$ and $y(n)$,
Step 2 Calculate:
  temp:= 0;
  for $i := 0$ to $N - 1$ do
  temp:=temp+$\hat{h}i(n)x(n-i)$;
    $\hat{y}(n) =$ temp;
Step 3 Calculate:
  $e(n) = y(n) - \hat{y}(n)$;
Step 4 // Updating of filter coefficients:
  for $i := 0$ to $N - 1$ do
  $\hat{h}i(n+1) = \hat{h}i(n) + 2\mu \cdot e(n) \cdot x(n-i)$;
Step 5
  $n := n + 1$;
if $n < L$ then go to Step 1;
```

Fig. 2. The diagram of sequential *LMS* algorithm

IV. 2. Single-run ParLMS algorithm

The next of the single-run echo cancellation algorithms is a parallel *ParLMS* algorithm. It constitutes a parallel version of the single-run classical LMS algorithm. First, we shall discuss the algorithm version working on N – processor CREW PRAM machine. Next we shall prove that the same algorithm can be executed on $O(\frac{N}{\log N})$ processor *CREW PRAM* machine preserving the time of computation at the level of $O(\log N)$.

The *ParLMS* algorithm has been enhanced in relation to the *LMS* algorithm taking the following designations:

$p = N$	– number of processors,
id	– identification (number) of the processor,
μ_{id}	– coefficient of adaptation rate ($0 < \mu_{id} < 1$), local for each processor.

Theorem 2. The *ParLMS* algorithm can be executed in $O(\log N)$ time on *CREW PRAM* machine with the use of N processors.

Proof. Step 0 is executed in constant time $O(1)$. Step 1 is executed only by zero processor also in constant time $O(1)$. The summation instep two is executed in time $O(\log N)$ with the use of computation scheme, in which the processors are organized in a logical structure of a binary tree. Steps 3, 4 and 5 demand constant time $O(1)$. The whole determines the time of step two execution determining the time of algorithm performance time to $O(\log N)$.■

Conclusion 2. For the method based on Theorem 2 we obtain speedup $S_p = O\left(N/\log N\right)$ and efficiency $E_p = O\left(1/\log N\right)$.□

The scheme of the parallel single-run *ParLMS* algorithm, being the transposition of *LMS* algorithm to the parallel *CREW PRAM* machine has been presented in Figure 3. This algorithm can be executed both on $O(N)$ and $O(N/\log N)$ – processor *CREW PRAM* machine preserving the computation time at the level of $O(\log N)$, which shall be demonstrated in proves of Theorem 2 and 3.

Algorithm 2. The diagram of parallel ParLMS algorithm.

```
1  Step 0 $n := 0$;
2  Step 1    if $id = 0$ then
3    Read input $x(n)$ and $y(n)$,
4  Step 2    Calculate:
5  $\hat{y}(id) := \hat{h}_{id}(n)x(n-id)$;
6    for $i := 0$ to $\lceil\log(N-1)\rceil$ do
7    $\hat{y}(id) := \hat{y}(id) + \hat{y}(id+2^i)$;
8  Step 3    if $id = 0$ then
9    $e(n) = y(n) - \hat{y}(0)$;
10 Step 4    // Updating of filter coefficients
11 $\hat{h}_{id}(n+1) = \hat{h}_{id}(n) + 2\mu_{id} \cdot e(n) \cdot x(n-id)$;
12 Step 5
13   $n := n+1$;
14   If $n < L$ then go to Step 1;
```

Fig. 3. The diagram of parallel *ParLMS* algorithm.

Theorem 3. The *ParLMS* algorithm can be executed in $O(\log N)$ time on *CREW PRAM* machine with the use of $O(N/\log N)$ processors.

Proof. While considering Step 2 of the *ParLMS* algorithm, an assumption has been made that the vector of input is stored in global memory. In this memory the intermediate results are also stored. Access to each vector element, as well as execution by each of the processors on the data elementary function may be realized in constant time $O\left(1\right)$. The computation has a tree character, hence we can demonstrate in what way one must assign tasks to $O(\frac{N}{\log N})$ processors to satisfy the assumption of computational complexity defined as $O(\log N)$. The following designations of were made: g – depth (level number) of tree computation scheme, N – the number of *ParLMS* coefficient filters, p – number of processors.

It should be observed that performance at the i-th level cannot be executed before all computation at level $i+1$ has been done. This limitation results directly from the fact that the computation of a given tree node cannot be executed before computing all its slave nodes in the tree. The input of all elements at level g are initial input of all *ParLMS* algorithm, therefore it is impossible to execute the computation at level $g-1$ without previous executing all computation for level g. Generally, it is impossible to execute all computation at level i without previous execution of all computation at levels from g to $i+1$.

Let n be the total number of elements in the circuit. For g-level circuit we can define n as follows:

$$n = \sum_{i=1}^{g} n_i, \tag{18}$$

where n_i denotes the number of elements at i-th level. The elements at i-level can be assigned to $\left\lceil \frac{n_i}{p} \right\rceil$ processor groups. Where $\lceil . \rceil$ denotes ceil function and $\lfloor . \rfloor$ floor function. In this case $\left\lfloor \frac{n_i}{p} \right\rfloor$ processor groups shall contain p elements each, whereas the last of the groups in case when $\left\lceil \frac{n_i}{p} \right\rceil \neq \left\lfloor \frac{n_i}{p} \right\rfloor$ shall contain the rest of not assigned elements. In view of the above the computation complexity at i-th level is $O(\left\lceil \frac{n_i}{p} \right\rceil)$. Hence the total computational time for all levels (thus the time of executing all the step 2 of the algorithm) is of the order of $O(\sum_{i=1}^{g} \left\lceil \frac{n_i}{p} \right\rceil)$. Using the ceiling function we get: $\sum_{i=1}^{g} \left\lceil \frac{n_i}{p} \right\rceil \leq \sum_{i=1}^{g} (\frac{n_i}{p} + 1) = \frac{n}{p} + g$. Moreover, observing that $g = \lceil \log(n_g) \rceil = \lceil \log(N) \rceil$ we obtain finally the computational complexity of step two given as $O(\log N)$. The substitution operations in lines 5 and 11 are executed by N processors in time $O\left(1\right)$. If $O(\frac{N}{\log N})$, i.e. $\frac{N}{N/\log N}$ times less processors were used, then the computation time would be $\frac{N}{N/\log N} = \log N$ times longer, in accordance to Brent theorem [16]. Hence the computational complexity of lines 5 and 11 shall be in the method from Theorem 3. $O(\frac{N}{\log N})$, preserving (towards the complexity of $O\left(1\right)$ of lines 1,2,3,9,13) the total algorithm execution time $O\left(\frac{N}{\log N}\right)$.■

Conclusion 3. For the method based on Theorem 3 we obtain speedup $S_p = O(\frac{N}{\log N})$ and efficiency $E_p = O\left(1\right)$. The method based on Theorem 3 is cost optimal.□

For the purpose of verification of theoretical results computational experiments have been realized in the *GPGPU* environment with the use of computation card *nVidia GeForce 9600M GS*.

The results are presented in Table 1. It can be observed that by such short computation times (of the order of thousandth part of second) the proposed methodology enables application of parallel methods of echo cancellation in real time circuits.

IV. 3. Single-run NLMS algorithm

The successor of the *LMS* algorithm, a single run acoustic echo cancellation algorithm that is described in this paper, can be considered the sequential *NLMS* (*Normalized LMS*) algorithm. Implementation of this algorithm to the RAM machine

Tab. 1. ParLMS algorithms runtimes

Filter length	Time [ms]		
	min.	max.	average
128	$1,7251 \cdot 10^{-4}$	$1,7306 \cdot 10^{-4}$	$1,7263 \cdot 10^{-4}$
256	$3,4417 \cdot 10^{-4}$	$3,4533 \cdot 10^{-4}$	$3,4456 \cdot 10^{-4}$
512	$6,8750 \cdot 10^{-4}$	$6,8996 \cdot 10^{-4}$	$6,8858 \cdot 10^{-4}$
1024	$1,3741 \cdot 10^{-3}$	$1,3791 \cdot 10^{-4}$	$1,3765 \cdot 10^{-3}$

described herein is an extension of the original *NLMS* algorithm with the flowchart and the proof for its computational complexity. This algorithm has a strict sequential character and can be executed on a single-processor RAM machine.

Similarly as for the LMS algorithm, the following designations were made:

$x(n)$ – input of an adaptive filter, containing the value of far-end speaker's signal function in n-th time moment,

$y(n)$ – a signal containing the echo of far-end speaker's speech and possibly near-end speaker's speech and noises in the n-th time moment,

$\hat{y}\,(n)$ – output of adaptive filter containing adaptively filtered signal x, being the estimate of echo signal in the n-th time moment,

$e\,(n) = y\,(n) - \hat{y}(n)$ – the error function in the n-th time moment,

N – the number of weight coefficients of the adaptive filter,

μ – the coefficient of adaptation rate ($0 < \mu < 1$),

γ – the coefficient with the close to zero value, aiming at prevention of zero value appearance in the denominator of the formula $\widehat{\boldsymbol{h}}\,(n+1) = \widehat{\boldsymbol{h}}\,(n) + \frac{\mu}{\gamma + \boldsymbol{x}^T(n)\boldsymbol{x}(n)}\boldsymbol{x}\,(n)\,e\,(n)$,

$\widehat{\boldsymbol{h}}(n)$ – weight coefficients of the adaptive filter in the n-th time moment.

The diagram of the sequential *NLMS* algorithm for the *RAM* machine has been presented in Figure 4.

Theorem 4. The *NLMS* algorithm may be executed in $O(N)$ time on single-processor *RAM* machine.

Prove. Step 0 is executed in constant time $O(1)$. Step 1 is also executed in constant time $O(1)$. Summation in step 2 is executed in time $O(N)$. Step 3 is executed in the constant time $O(1)$. Steps 4 and 5 require computational expenditure $O(N)$, whereas step 6 is executed again in constant time $O(1)$. The whole determines the time of step 2, 4 and 5 execution determining the time of algorithm performance time to $O(N)$.■

IV. 4. Single-run ParNLMS algorithm

The next of the single-run echo cancellation algorithms is the parallel *ParNLMS* algorithm. It constitutes a parallel version of the single-run classical *NLMS* algorithm. First, while describing the *ParNLMS* algorithm, we shall discuss the algorithm version working on N – processor *CREW PRAM* machine. We shall prove that the same algorithm can be executed on $O(\frac{N}{\log N})$ processor *CREW PRAM* machine preserving the time of computation at the level of $O(\log N)$.

Figure 5 presents the diagram of the parallel *NLMS* algorithm for the *SM SIMD* architecture (*Shared Memory SIMD*)

```
Algorithm 3. NLMS algorithm scheme
Step 0 n: = 0;
Step 1Read input x(n)and y(n),
Step 2 Calculate:
  $\hat{y}(n) = \sum_{i=0}^{N-2} \hat{h}_i(n)x(n-i)$;
Step 3 Calculate:
  $e(n) = y(n) - \hat{y}(n)$;
Step 4    // Updating of $\mu$ coefficient:
  $z := 0$;
  for $i := 0$ to $N-1$ do
    $z := z + x^2(n-i)$;
    $\mu(n) = \frac{\mu}{\gamma+z}$
Step 5    // Updating of filter coefficients:
  for $i := 0$ to $N-1$ do
  $\hat{h}_i(n+1) = \hat{h}_i(n) + 2\mu \cdot e(n) \cdot x(n-i)$;
Step 5
  $n := n+1$;
  if $n < L$ then go to Step 1;
```

Fig. 4. The diagram of sequential *NLMS* algorithm

with the p number of processors. Additional denotations have been accepted for the NLMS algorithm diagram, and so:$p = N$ – number of processors, id – identification (number) of the processor.

```
1  Algorithm 4. ParNLMSalgorithm scheme
2
3  Step 0 $n := 0$;
4  Step 1
5    if $id = 0$ then
6      Read input x(n) and y(n),
7  Step 2   Calculate:
8    $\hat{y}(id) := hathid(n)x(n-id)$;
9    for $i := 0$ to $\lceil \log(N-1) \rceil$ do
10     $\hat{y}(id) := \hat{y}(id) + \hat{y}(id + 2^i)$;
11 Step 3
12   if $id = 0$ then
13     $e(n) = y(n) - \hat{y}(n)$;
14 Step 4   // Updating of $\mu$ coefficients
15   $z(id) := x^2(n-id)$;
16   for $i := 0$ to $\lceil \log(N-1) \rceil$ do
17     $z(id) := z(id) + z(id + 2^i)$;
18   \mu(n) = \frac{\mu}{\gamma+z(0)};
19 Step 5   // Updating of filter coefficients
20   $\hat{h}_{id}(n+1) = \hat{h}_{id}(n) + 2\mu(n) \cdot e(n) \cdot x(n-id)$;
21 Step 6
22   $n := n + 1$;
23   if $n < L$ then go to Step 1;
```

Fig. 5. The diagram of parallel *ParNLMS* algorithm

Theorem 5. The *ParNLMS* algorithm can be executed in $O(\log N)$ time on the *CREW PRAM* machine with the use of N processors.

Prove. One may note that the *ParNLMS* algorithm construction is the extension of the *ParLMS* algorithm with the update statements of the μ coefficient. The proof is therefore in significant part analogous to the proof of theorem of computational complexity of the *ParLMS* algorithm on the N-processor *PRAM* machine. Step 0 is executed in the constant time $O(1)$. Step 1 is executed only by the zero processor also in constant time $O(1)$. The summation in step 2 is executed in time $O(\log N)$, in accordance with the "tree" computational diagram. Step 3 requires again constant computational input $O(1)$. The update of μcoefficient, that is performed in step 4 is based – as in the case of step 2 computation – on the tree summation diagram that requires the $O(\log N)$ computational input. In distinction from step 2 computation after summation, the result is used to compute the $\mu(n)$ coefficient value that requires constant computational time $O(1)$. The computational complexity of step 4 is determined by the operation of the "tree" summation. The computation in steps 5 and 6 demand constant time $O(1)$. The whole determines the time of step 2 and 4 execution determining the time of algorithm performance to $O(\log N)$.■

Conclusion 5. For the method based on Theorem 5 we obtain speedup $S_p = O(\frac{N}{\log N})$ and efficiency $E_p = O(\frac{1}{\log N})$.□

Theorem 6. The *ParNLMS* algorithm can be executed in $O(\log N)$ time on *CREW PRAM* machine with the use of $O(\frac{N}{\log N})$ processors.

Prove. Similarly to Theorem 5 that has been the extension of theorem 2, Theorem 6 is the extension of theorem 3 with the actions described in step 4 of the *ParNLMS* algorithm. Hence, the theorem proof is the complement of the Theorem 3 proof. The extension is made by the computation in step 4. However, the computational complexity in lines 13-15 in Theorem 6 is proved in the same way as the computational complexity in lines 5-7 in theorem 3.

Conclusion 6. For the method based on Theorem 5 we obtain speedup $S_p = O(\frac{N}{\log N})$ and efficiency $E_p = O(1)$.□

The presented method of *NLMS* algorithm parallelization is therefore optimal in cost.

IV. 5. Single-run sequential PNLMS algorithm

The successor of the *LMS, NLMS ParNLMS* and *ParNLMS* algorithms, a single run acoustic echo cancellation algorithm that is described in this paper can be considered the sequential *PNLMS* (*Proportional Normalized LMS*) algorithm. Implementation of this algorithm to the *RAM* machine described herein is an extension of the original *PNLMS* algorithm with the flowchart and the proof for its computational complexity. This algorithm has strict sequential character and can be executed on a single-processor *RAM* machine.
The diagram of the sequential *PNLMS* algorithm has been presented in Figure 6. The denotations have been enhanced by the coefficients g_j, $\delta_j\ (n+1)$, ρ and γ_p defined as follows:

$$g_j(n+1) = \frac{\delta_j(n{+}1)}{\frac{1}{L}\sum_{i=0}^{N-1}\delta_i(n)}, 0 \le j \le N-1,$$

$$\delta_j\ (n+1) = \max\{\delta_{\min}, \left|\hat{h}_j\ (n)\right|\}, 0 \le j \le N-1,$$

$$\delta_{\min} = \rho \max\{\gamma_p, \left|\hat{h}_0\ (n)\right|, \left|\hat{h}_1\ (n)\right|, \ldots, \left|\hat{h}_{N-1}\ (n)\right|\},$$

whereρ and γ_p are constants, usually taking the values $\rho = \frac{5}{N}$ and $\gamma_p = 0,01$ [17].

Theorem 7. The *PNLMS* algorithm may be executed in $O(N^2)$ time on single-processor *RAM* machine.

Prove. Step 0 is executed in constant time $O(1)$. Step 1 is also executed in constant time $O(1)$. Summation in step 2 is executed in time $O(N)$. Step 3 is executed in the constant time $O(1)$. Steps 4, 5 and 6 require $O(N)$ computational input, while step 7 requires denomination of N elements of $\hat{h}(n)$ vector, for each while performing the matrix multiplication in time $O(N)$ (the multiplied matrixes are vectors, or diagonal matrix of the N order). Hence, the time of step 7 execution is $O(N^2)$. The complete time is determined by

Algorithm 5. PNLMS algorithm scheme

```
Step 0 $n := 0$;
Step 1   Read input $x(n)$ and $y(n)$,
Step 2  Calculate:
  $\hat{y}(n) = \sum_{i=0}^{N-1} \hat{h}_i(n)x(n-i)$;
Step 3 Calculate:
  $e(n) = y(n) - \hat{y}(n)$;
Step 4 // Searching for $\delta_{min}$
  $max := \gamma_p$;
  for $i := 0$ to $N-1$ do
    if $(max < \hat{h}_i(n))$ then
      $max := \hat{h}_i(n)$;
  $\delta_{min} := \rho \cdot max$;
Step 5 // Calculating of $\delta_j(n+1)$
  for $j := 0$ to $N-1$ do
    if $(\delta min < |\hat{h}_j(n)|)$ then
      $\delta j(n+1) := \hat{h}_j(n)$;
    else
      $\delta j(n+1) := \delta_{min}$;
Step 6 // Calculating of $g_j(n+1)$
         // Only once-shared for each $g_j$
  $sum := 0$;
  $fori := 0$ to $N-1$ do
    $sum := sum + \delta i(n)$;
  $sum := sum/N$;
  for $j := 0$ to $N-1$ do
    $g_j(n+1) := \frac{n+1}{sum}$;
Step 7 // Updating of filter coef.
  for i:=0 to $N-1$ do
    $\hat{h}_i(n+1) = \hat{h}_i(n) + \frac{\mu}{\gamma + \boldsymbol{x}^T(n)\boldsymbol{G}(n)\boldsymbol{x}(n)}\boldsymbol{G}(n)\boldsymbol{x}(n)e(n)$;
Step 8
  $n := n+1$;
  if $n < L$ then go to Step 1;
```

Fig. 6. The diagram of sequential *PNLMS* algorithm

step seven execution time determining the time of algorithm performance time to $O(N^2)$.■

IV. 6. Single-run sequential PNLMS algorithm

Implementation of this algorithm to the P*RAM* machine described herein is a parallel extension of the original sequential *PNLMS* algorithm with the flowchart and the proof for its computational complexity.

Theorem 8. The parallel *ParNLMS* algorithm can be executed in $O(logN)$ time on the CREW PRAM machine with the use of N^2 processors.

Prove. Step 0 is executed in constant time $O(1)$. Step 1 is also executed in constant time $O(1)$. Summation in step 2 is executed in time $O(\log N)$. Step 3 is executed in the constant time $O(1)$. Steps 4 and 6 require $O(\log N)$ computational input, step 5 is performed in constant time $O(1)$, while step 7 requires denomination of N elements of $\hat{h}(n)$ vector, for each while performing the matrix multiplication in time $O(\log N)$ (using a tree diagram of N-processor summation). Hence, the time of step 7 execution is $O(\log N)$with the use of N^2 processors. The complete time is determined by step 7 execution time determining the time of algorithm performance time to $O(\log N)$.■

Conclusion 8. For the method based on Theorem 8 we obtain speedup $S_p = O(\frac{N^2}{\log N})$ and efficiency $E_p = O(\frac{1}{\log N})$.□

Theorem 9. The parallel *ParPNLMS* algorithm can be executed in $O(\log N)$ time on the *CREW PRAM* machine with the use of $O(\frac{N^2}{\log N})$ processors.

Prove. The proof method is similar to Theorem 8 with the difference that in step 7 there are used $O(\frac{N^2}{\log N})$ processors during the cost optimal scheme of N number summation, in accordance with the Brent Theorem.■

Conclusion 9. The proposed method in the proof of Theorem 9 is cost optimal with the efficiency $E_p = O(1)$. The speedup value for this method is $S_p = O(\frac{N^2}{\log N})$.□

The diagram of parallel *ParPNLMS* for the selected processor with the id p index – processor *CREW PRAM* machine has been presented in Figure 7.

Algorithm 6. ParPNLMSalgorithm scheme

```
Step 0 $n := 0$;
Step 1
   if $id = 0$ then
      Read input $x(n)$ and $y(n)$,
Step 2    Calculate:
$\hat{y}(id) := \hat{h}_{id}(n)x(n-id)$;
   for $i := 0$ to $\lceil \log(N-1) \rceil$ do
      $\hat{y}(id) := \hat{y}(id) + \hat{y}(id+2^i)$;
Step 3
   if $id = 0$ then
      $e(n) = y(n) - \hat{y}(n)$;
Step 4    // Searching for $\delta_{min}$ value
   $z(id) := \hat{h}_{id}(n)$;
   for $i := 0$ to $\lceil \log(N-1) \rceil$ do
      $z(id) := max(z(id), z(id+2^i))$;
   $max := max(z(0), \gamma_p)$;
   $\delta_{min} := \rho \cdot max$;
Step 5    // Calculations of $\delta(n+1)$
   if $(\delta_{min} < |\hat{h}_{id}(n)|)$ then
      $\delta_{id}(n+1) := \hat{h}id(n)$;
   else
      $\delta_{id}(n+1) := \delta_{min}$;
Step 6 // Calculation of $g(n+1)$
         // Evaluated only ones
   $z(id) := \delta_i(n)$;
   for $i := 0$ to $\lceil \log(N-1) \rceil$ do
      $z(id) := (z(id), z(id+2^i))$;
      $sum := \frac{z(0)}{N}$;
   $g_{id}(n+1) := \frac{n+1}{sum}$;
Step 7 // Updating of filter coefficients
$\hat{h}_{id}(n+1) = \hat{h}_{id}(n) + \frac{\mu}{\gamma \boldsymbol{x}^T(n)\boldsymbol{G}(n)\boldsymbol{x}(n)}\boldsymbol{G}(n)\boldsymbol{x}(n)e(n)$;
Step 8
   $n := n+1$;
   if $n < L$ then go to Step 1;
```

Fig. 7. The diagram of parallel *ParPNLMS* algorithm with the id index

V. CONCLUSIONS

For a serial and different parallel implementations of the LMS algorithm and two of its modifications, we derive and discuss the order of execution time of these algorithms. Using a general purpose graphics processing unit (GPGPU)-card they evaluated the execution time of the parallel LMS algorithm in an echo cancellation application for different filter lengths. The benefits of applying parallel algorithms from the computational time point of view basing on *PRAM* parallel computing model have been discussed. The theoretical results have been carried by numerical experiments conducted in *CPU* and *GPU* environments.

References

[1] H. Yusukawa, S. Shimada, *An acoustic echo canceller using subband sampling and decorrelation methods*, IEEE Trans. Signal Processing, **41**, 926-930 (1993).

[2] J. Chen, H. Bes, J. Vandewalle, P. Janssens, *A new structure for sub-band acoustic echo canceler*, Proc. IEEE ICASSP, 2574-2577 (1988).

[3] W. Kellermann, *Some aspects of the frequency-subband approach to the cancellation of acoustical echoes*, Proc. IEEE ICASSP, 2570-2573 (1988).

[4] B. Hatty, *Block Recursive Least Squares Adaptive Filters Using Multirate Systems for Cancellation of Acoustical Echoes*, Proc. IEEE ASSP Workshop on Application of Signal Processing to Audio and Acoustics, 1989.

[5] A. Gilliore, M. Vetterli, *Adaptive filtering in subbands with criticalsampling: analysis, experiments, and application to acoustic echo cancellation*, IEEE Trans. Signal Processing, **40**, 1862-1875 (1992).

[6] A. Herikstad, *Gpu sound processing*, December 2008. Specialization Project TDT4590, Complex Computer Systems, NTNU. 33, 36, 39, 47.

[7] A. Dobrucki, W. Bożejko, M. Walczyński, *Parallelizing of digital signal processing with using GPU*, Signal processing, algorithms, architectures, arrangements, and (2012) applications, SPA 2010 : conference proceedings, Poznan, 23-25th September 2010. [Poznań : Chapter Circuits and Systems Chapter Signal Processing Poland Section, Institute of Electrical and Electronics Engineers, 2010], 29-33.

[8] W. Bożejko, M. Walczyński, *Noise reduction with using parallel algorithms*, Noise Control '10 [electronic document]: 15th International Conference on Noise Control, 6-9 June 2010, Wałbrzych. Warszawa : Centralny Instytut Ochrony Pracy – Państwowy Instytut Badawczy, 2010, 1-8.

[9] M. Walczyński, *Zastosowanie algorytmów równoległych w problemie odszumiania sygnałów dźwiękowych i wizyjnych na przykładzie algorytmu LMS w mechatronice*, Mechatronika. Nauka dla gospodarki. Rzeszów 2011. ISBN 978-83-63151-01-0 (in polish).

[10] A. Dobrucki, S. Brachmański, P. Pruchnicki, P. Staroniewicz, P. Plaskota, M. Walczyński, *Subvocal speech recognition based on electromyography : Package WP7:Source codes of software for parameterization of recorded SVR signals and for management of files with signals. Technical documentation of SVR sensor. Package WP8: Test of the realized SVR sensor Results of subvocal speech recognition using sensor. Summary of results achived in project*, Report SPR series, 2012.

[11] A. Dobrucki, S. Brachmański, P. Pruchnicki, P. Staroniewicz, P. Plaskota, M. Walczyński, *Subvocal speech recognition based on electromyography : Package WP4: Purchased equipment and its running EMG-SVR parametrization algorithms; Database of test signals; Experiments with EMG signals for vocal and subvocal speech*, Report SPR series, 2011.

[12] A. Dobrucki, W. Bożejko, M. Walczyński, *LMS algorithms parallelization in GPGPU environment*, Elektronika (Warsaw), **52**(5), 49-53 (2011).

[13] M.M. Sondhi, *An adaptive echo canceller*, Bell Syst. Tech. J., **46**(3), 497-511 (1967).

[14] M.M. Sondhi and A.J. Presti, *A self-adaptive echo canceller*, Bell Syst. Tech. J., **45**(12), 1851-1854 (1966).

[15] W. Bożejko, *On single-walk parallelization of the job shop problem solving algorithms*, Computers & Operations Research **39**, 2258-2264 (2012).

[16] T.H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to algorithms, MIT Press, 2009.
[17] T. Gansler, J. Benesty, D. R. Morgan, M. M. Sondhi, S. L. Gay, *Advances in Network and Acoustic Echo Cancellation*, Springer-Verlag, Berlin, Germany, 2001.

Andrzej Dobrucki was born in Rawicz, Poland, in 1949. He received the MSc degree in 1971 from the Faculty of Electronics of Wroclaw University of Technology. In 1977 he received the PhD degree for a dissertation on vibration and sound radiation of conical shells. In 1993 he received the DSc degree (habilitation). In 2007 Andrzej Dobrucki received the title of professor. His research interests are in the construction and measurement of electroacoustic transducers, the numerical modeling of acoustic fields, vibrations in mechanical structures and signal processing. The total number of his publications is: 3 scientific monographs, 2 handbooks, over 40 papers in scientific journals, over 120 presentations at international and local conferences published in the conference's proceedings, 6 Polish patents. He was the supervisor in 14 PhD projects. Andrzej Dobrucki is the member of the Polish Acoustical Society and the Fellow of the Audio Engineering Society (AES). In 2014 he was elected the President of Polish Acoustical Society. He is also the Head of the Chair of Acoustics and Multimedia in Wrocław University of Technology.

Maciej Walczyński was born in Wrocław, Poland, in 1983. He received the MSc degree in 2008 from the Faculty of Physics of Wroclaw University. He is also PhD student at the Wroclaw University of Technology in Faculty of Electronics. His PhD dissertation is about acoustic echo cancellation with using parallel algorithms. His research interests include Digital Signal Processing, Algorithms, Parallel Processing. He is an author of about 15 published in books and conference proceedings from the field of parallel processing, Digital Signal Processing. He is also an Assistant Professor at General Tadeusz Kościuszko Military Academy of Land Forces in Wroclaw.

Wojciech Bożejko is an Associated Professor at Wrocław University of Technology. He obtained his MSc at the University of Wroclaw, Institute of Computer Science, in 1999, PhD in Wrocław University of Technology, Institute of Engineering Cybernetics in 2003 and D.Sc. (habilitation) in Wrocław University of Technology, Faculty of Electronics, in 2011. Since 2013 he has been working as the Associated Professor at the Institute of Computer Engineering, Control and Robotics. He is an author of over 160 papers (over 150 citations on ISI Web of Knowledge, h-index: 8) published in peer-reviewed journals and conference proceedings in the field of parallel processing, scheduling and optimization. He is interested in parallel algorithms, GPU computing, scheduling and discrete optimization.