computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific
Journal of Computing

EFFICIENCY ANALYSIS OF PARALLEL ROUTINE USING PROCESSOR TIME VISUALIZATION

Volodymyr Turchenko[1)-2)], Viktor Demchuk[1)]

[1)] Research Institute of Intelligent Computer Systems
Ternopil Academy of National Economy
3 Peremoga Square, Ternopil, 46004, UKRAINE, e-mail: {vtu}, {vde}@tanet.edu.te.ua
[2)] Center of Excellence of High Performance Computing, University of Calabria,
via. P. Bucci 22A, Rende (CS), 87036, ITALY, e-mail: vtu@hpcc.unical.it

Abstract: *The coarse-grain parallel algorithm of modular neural networks training with dynamic mapping onto processors of parallel computer is described in this paper. Parallelization of the algorithm is done on parallel computer Origin 300 using MPI technology. The efficiency of this algorithm is estimated using modification of MPE visualization library, which measures processor executing time of parallel routines.*

Keywords: *efficiency analysis, processor time, dynamic mapping, coarse-grain parallelization, neural networks, MPI, MPE.*

1. INTRODUCTION

Nowadays, parallel computers are advanced technologies that are pushing several applications towards new challenge in many fields. The computational capacity of one-processor computers does not always satisfy the constantly growing requirements of data processing in science, industry and business [1-3]. Parallel data processing is one of the approaches that would make feasible the execution of complicated algorithms with a large number of variables and iterations. A trivial parallelism consists in dividing the sequential program among the available processors and running it in parallel in order to reduce the total execution time. However, this simple approach might not be effective due to additional overhead spending in synchronization, communication and load imbalance among processors. As a result, several issues related to the development of effective parallel programs are still remaining urgent.

It is necessary to consider each of these issues separately in each specific application, because any application imposes certain limitations on the design of the parallel algorithm as well as on the parallel architecture and the parallel programming paradigms to be used. Therefore the issues of effective parallelization include effective parallel algorithms, parallel architectures, parallel programming languages and performance analysis of parallel programs [4-5].

The goal of this paper is efficiency analysis (by visualization) of coarse-grain parallel algorithm of dynamic mapping of Integrating Historical Data Neural Networks [6] used for sensor drift prediction. The original method of training set forming of Integrating Historical Data Neural Networks (IHDNN's) gives the possibility to solve the problem of the sensor drift prediction much better than the well-known mathematical methods [7-8]. The advantages of this method consist in ability of the neural networks to improve the accuracy of the sensor data acquisition and processing in several times (3-5) on increased duration of inter-calibration interval (6-12 times) using small number of input data as training set [8]. However, this method requires a considerable computational load (approx. 40 min. for one data acquisition channel). Description of historical data integration method, choice of the architecture of IHDNNs, choice of the approach of task parallelization and choice of the level of parallelism are out of scope of this paper. These issues are described in details in papers [6, 9].

This study is organized as follows: the development of dynamic mapping algorithm of IHDNNs parallelization and the results of experimental researches are described in Section 2, the efficiency analysis based on visualization technique is described in Section 3 and some final remarks in Section 4 conclude this paper.

2. COARSE-GRAIN PARALLEL ALGORITHMS OF IHDNNS TRAINING WITH DYNAMIC MAPPING

2.1. Design of the algorithm

As it is shown in [6, 9], M IHDNNs are used to solve integration historical data algorithm, M is equal to 50 for the experiments. In mentioned papers we showed, that it is necessary to choose coarse-grain parallelization level of neural networks, i.e. each module of neural network is executing on separate processor of parallel computer.

Parallel algorithm (Fig. 1) is developed by using a "centralized" planning approach with only one processor (*Master*, $cp = 0$) having the role of task planner and each of the other processors (called *Slaves*) will train the IHDNNs assigned by the *Master*. Once a *Slave* has finished its task, it asks the *Master* for a new one till no tasks are left. We note here that, besides the role of planner, the *Master* does not fulfill any further calculation. The communications between the *Master* and the *Slaves* are ensured by using the standard MPI sending/receiving functions *MPI_Send()* and *MPI_Recv()* [10, 11].

The sequential part of the algorithm includes two operations: (i) reading the input data with the sensor drift and (ii) defining the sequential numbers of the IHDNNs based on the input data. However these operations are showed separately in Fig. 1 in order to simplify the description of the algorithm. The parallel part of the algorithm starts with the call of the *MPI_Init()* function (Fig. 1b). Here the index $< cn >$ denotes the sequential number of IHDNN and the $< cp >$ index refers to the processor ID. All the communications between the *Master* and the *Slave* include the index $< cn >$, which is useful for each process to pick up the data corresponding to the IHDNN to be trained. This scheme allows a considerable decrease of the length of the messages and consequently the latency time of the communications.

The *Master* starts with assigning the first M IHDNNs to the M available processors and then continues the execution of the mapping procedure consisting of assigning dynamically the tasks to the *Slaves* as soon as they become idle. The *Master* receives also the results of the already trained IHDNNs by using the function *MPI_Recv()* with the *MPI_ANY_SOURCE* parameter and saves its content in the appropriate cell of the output matrix $ComM$. The stopping condition of the algorithm is to check that all the IHDNNs have been already mapped.

Each *Slave* checks the availability of a new message from the *Master* by using the function *MPI_Probe()*. Among all the received messages, each *Slave* should consider only those having an index $< cp >$ corresponding to its proper ID.

On the basis of the IHDNN index $< cn >$ to be trained each *Slave* performs the following operations:

- Form the training set of the IHDNN based on the values *row*[*cn*] and *col*[*cn*] for the appropriate IHDNN based on steps described in [12];
- Train the IHDNN as multi-layer perceptron using back propagation error training algorithm;
- Run the historical data integration procedure and send the results, together with the IHDNN index $< cn >$, to the *Master*.

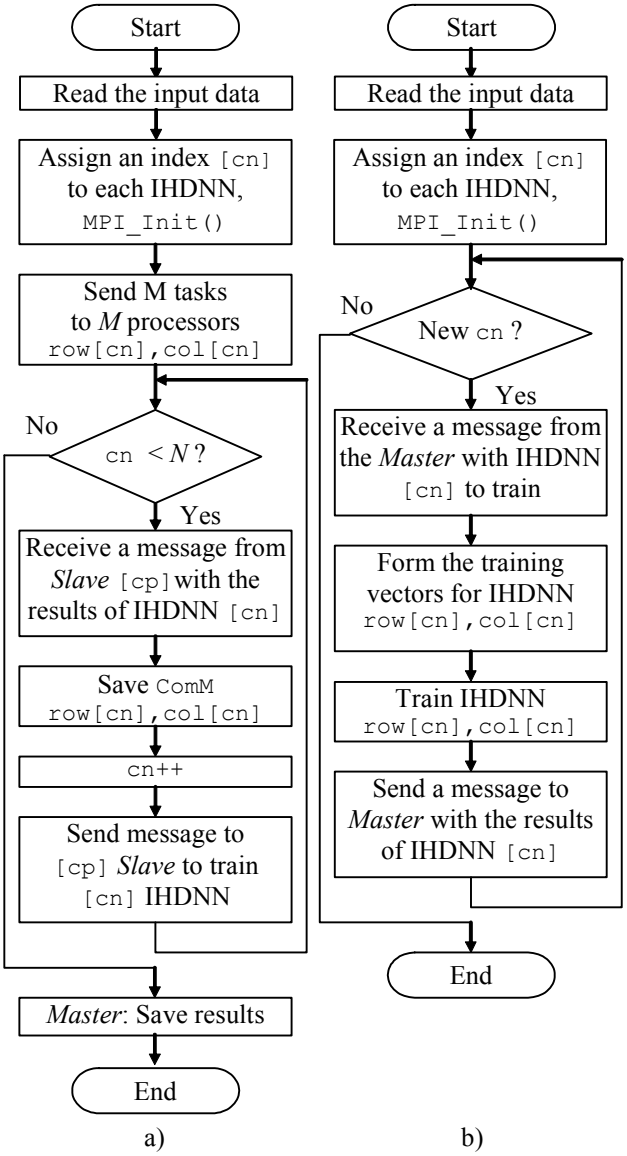

Fig. 1 - Coarse-grain parallel algorithm of IHDNNs training with dynamical mapping: (a) *Master*'s procedure, (b) *Slave*'s procedure

On the basis of the above description it is clear that our parallel algorithm with dynamic mapping does not make use of any synchronization point.

Therefore, the only source of efficiency loss in our implementation can derive from the overhead caused by the message passing communication.

2.2. Experimental researches

The experimental researches of developed parallel algorithm with dynamical mapping were fulfilled using the sensor drift "with saturation" [8]. We have measured execution time of 50 IHDNNs on 2, 4 and 8 *Slaves* of Origin 300 (placed in the Center of Excellence of High Performance Computing, University of Calabria, Italy), one additional processor have been used as *Master*. The sequential and parallel routines are developed on C language using MPI library v.1.2 [10, 11] and performance visualization library MPE [13, 14].

The experiments mentioned above have been fulfilled on the parallel computer Origin300 containing 8 RISC processors MIPS R14000 with clock rate 500 MHz and 4 Gb total RAM. Each processor has 2 Mb internal cash memory. Computer Origin300 operates under operation system UNIX (IRIX64 6.5). The execution time, speedup and efficiency of developed parallel routine are shown in the Table 1, Fig. 2 and Fig. 3 respectively. As it is seen, the speedup is non-linear, the effectiveness is decreasing as during increasing sum-squared training error (SSE) of neural networks, as well as during increasing the number of parallel processors used.

Table 1. Execution time in seconds: drift with "saturation"

Processors	1	2	4	8
SSE=10^{-3}	23.41	12.07	6.14	3.16
SSE=10^{-4}	173.14	88.70	45.22	25.23
SSE=10^{-5}	637.70	326.00	234.70	219.76
SSE=10^{-7}	1114.53	570.70	304.33	282.93

3. ANALYSIS OF PARALLELIZATION EFFICIENCY

Taking into account the nature of the developed parallel algorithm by using message passing approach among the leaves (against usage of synchronization and barriers), the efficiency can be decreased by:

- Non-accuracy of time measurement of parallel routine execution;
- Latency during communication among leaves;
- Load imbalance among processors.

For efficiency reduction estimation we have used MPE visualization library [13, 14]. Our case study was: 50 IHDNNs have been parallelized among 8 processors of Origin 300 at SSE=10^{-7} for drift "with saturation" (Fig. 4).

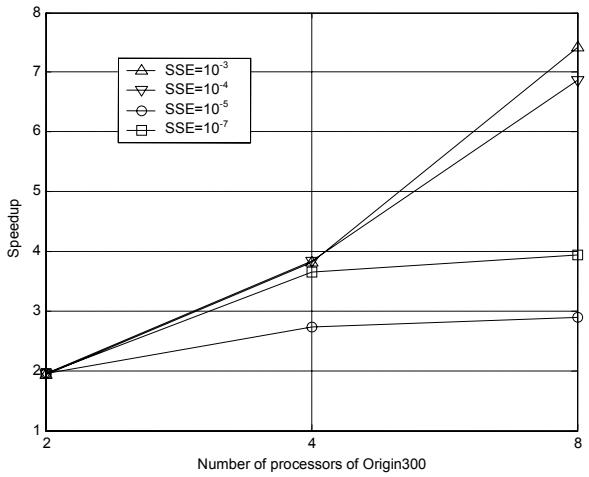

Fig. 2 - Speedup vs number of processors for drift "with saturation"

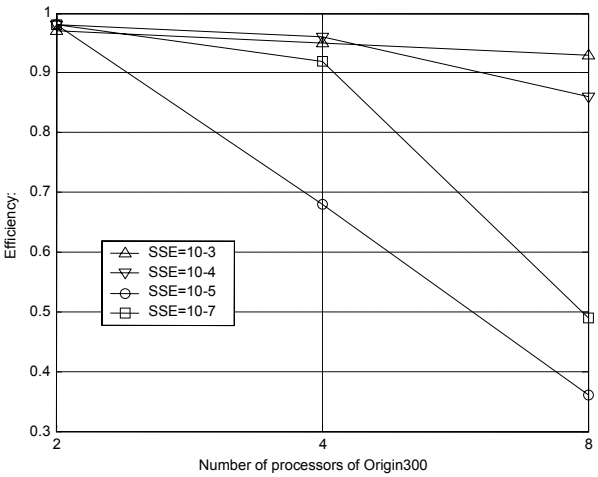

Fig. 3 - Efficiency vs number of processors for drift "with saturation"

Dark color shows execution time of each processor, light color shows waiting time of the *Master* (number 0) during reducing the data from the *Slaves*. Thin dark strips on the processing time rectangle of the *Master* show its working time for receiving the data from the *Slaves* and the communication time.

The visualization results from Fig. 4 show, that total executing time of parallel routine (see field *View Final Time* on Fig. 4) is considerably bigger (482.81 seconds) than execution time of this routine on 8 processors from Table 1 (282.93 seconds). After several investigations of this situation we showed, that MPE library provides non-accurate time management. In particular, for time measurement inside of parallel routine we have used function *times()*, which measure pure processor time of parallel routine. However, MPE library measure real (astronomic) time of parallel routine operation, which depends on loading of parallel computer by another system or user programs. In other words, MPE library measure the time interval between two

events in the past and the future using some functions of astronomic time measurement. All system and user programs, which are executed by a parallel computer during this period of time, influence on the duration of this interval. In this case the resource of each processor is distributed among all the routines of parallel computer.

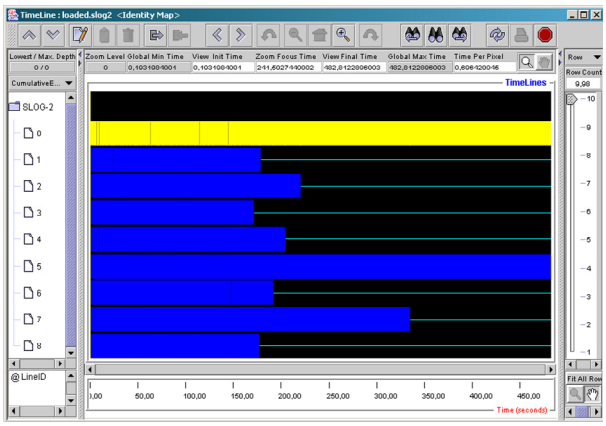

Fig. 4 Visualization of IHDNNs training at SSE=10^{-7} on 8 processors of Origin 300 using astronomic time

This fact causes non-accurate calculation of speedup and efficiency of parallelization. It is possible to avoid this disadvantage by measurement of the routine's processor time as we have done for our parallel code. Processor time can be measured by using processor ticks approach. This measurement technology allows avoiding the influence of other running programs. We can note at least two ways of this problem solving: (i) to modify the existing MPE library and (ii) to develop new visualization tool based on processor time measurement.

It is obvious, that the first way is easier, therefore we have replaced function of astronomic time measurement *CLOG_timestamp()* of existing MPE library. We have done it by adding new function *CLOG_timestamp2(),* which uses function of processor time measurement *times()*. More detailed description of this modification is described in [15].

During experimental research of modified MPI library we have chosen such a situation, when our parallel routine is run on the parallel computer Origin 300 loaded by execution of other routines. The results of modified MPE library in Fig. 5 are similar by form to Fig. 4, however they are practically the same with the results, presented in Table 1. For example, total execution time of parallel routine by using modified MPE library (see box *View Final Time* in Fig. 5) is equal to 280.41 seconds, which is practically the same to 282.93 seconds spent by Origin 300 to parallelize this algorithm in Table 1. This small difference between these two experiments can be explained by error of processor time estimation by modified MPE library. Thus, using the modified MPE library with processor time measurement allows ignoring the influence of other system and user computational tasks on Origin 300.

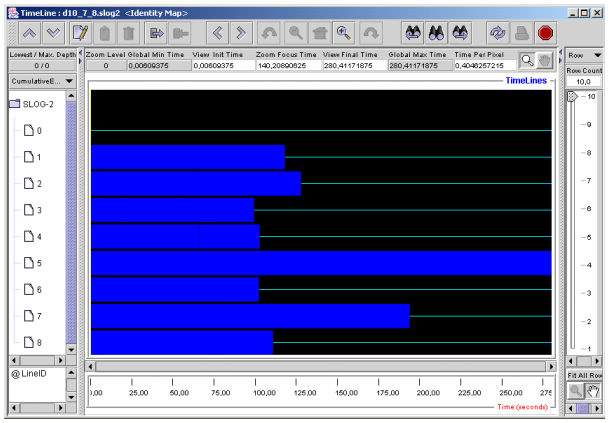

Fig. 5 Visualization of IHDNNs training at SSE=10^{-7} on 8 processors of Origin 300 using processor time

It is necessary to note, that some modern parallel computers have built-in functions of processor time calculation of running parallel programs. Parallel computer TX-7 [16] by NEC Corporation is one of such computers. Similar experiments described above were fulfilled on the model *i9010* of this parallel computer, which is also placed in the Center of Excellence of High Performance Computing. Model i9010 contains 16 processors Intel® Itanium-II with clock rate 1GHz, 3Mb L3 cash-memory and 64 Gb total RAM. The results of experimental researches have shown [6] that the values of time calculated both by functions of real (astronomic) time and processor time are the same. Therefore it is expedient to use the functions of processor time calculation on any parallel computer in order to provide the accurate results in general and modified MPE library in particular.

Therefore in our case the time of parallel routine executing is measured accurately. This time measurement does not influence on decreasing of parallelization efficiency.

Latency during communication among the *Master* and the *Slaves* is expedient to estimate using MPE profiling. For this purpose we have used *slog* (scalable logging) file and *jumpshot-4* software tool [17]. *Jumpshot-4* is implemented on *Java* programming language and therefore it requires *j2sdk* (Software Development Kit) packet, needed to be installed in order to provide the visualization. *Jumpshot-4* transforms the binary data from "log"-file as color rectangles (see Fig. 4 and Fig. 5), which show operation (by color) and duration of this operation (by length of the rectangle on X axis). Also this tool allows increasing a visualization step

and showing smaller time intervals of parallel programs running. For example, we have shown IHDNN's training time by dark color and communication time between *Master* and 1st *Slave* by light color in Fig. 6. It is visible that communication time among two processors does not exceed 0.08 second. According to Fig. 1 spending this time the *Master* (i) receives the results of IHDNN training from 1st *Slave*, (ii) saves this result in own memory and (iii) sends next IHDNN (with bigger reference number) to 1st *Slave*. Generally low communication latency is confirmed by very thin dark strips (they are communication time) on the light rectangle of the *Master* (it is waiting time) in Fig. 4 and Fig. 5. Therefore the latency practically does not influence on decreasing of parallelization efficiency. It also can be explained by the fact that we have chosen a coarse-grain approach to parallelize modular neural networks [6, 9].

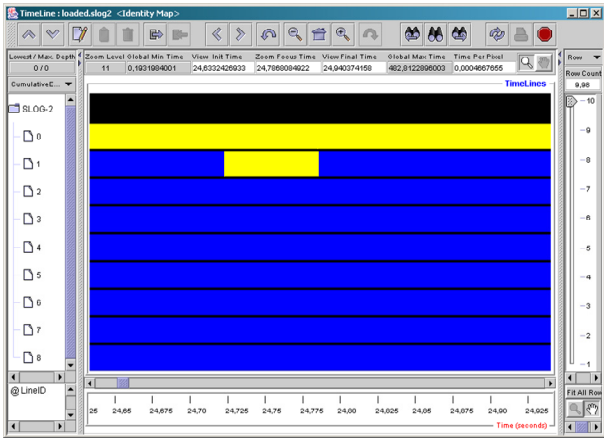

Fig. 6 - Communication time between the *Master* and 1st *Slave* does not exceed 0.08 second

The analysis of Fig. 4 and Fig. 5 clearly shows that efficiency decreasing of developed parallel algorithm with dynamic mapping caused by load imbalance among processors, i.e. the processing time of 5th *Slave* is biggest one. It is connected with non-stationary and non-uniform distribution of the training time of all IHDNNs. For example, we have shown a distribution of the training time of all 50th IHDNNs in Fig. 7.

Thus, summarizing the analysis above, it is expedient to consider two following approaches to parallelize this task in a case of non-uniform distribution of the training time of each module of IHDNN:

- If we can predict the training time of each IHDNN module before the parallelization, then it is possible to increase the efficiency by optimal mapping of the IHDNNs using predicted training time as mapping criteria. In this case long tasks should be mapped first on separate processors;

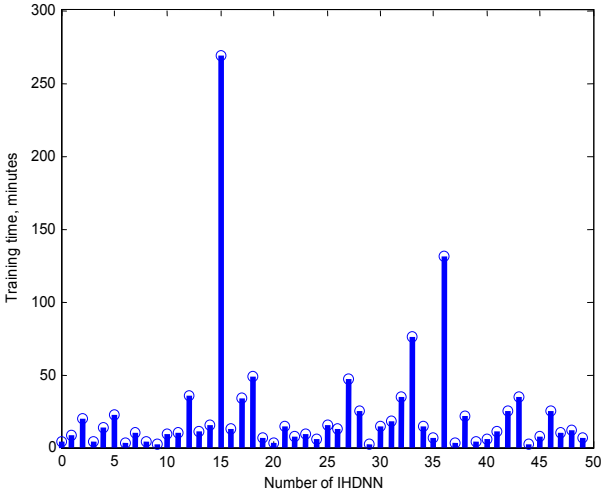

Fig. 7 - Distribution of the training time of all 50th IHDNNs

- If we can not predict the training time of each IHDNN module before the parallelization, then it is expedient to develop another methods of IHDNNs parallel training by using fine-grain approach and to apply this fine-grain algorithm dynamically to such IHDNN module, which remains training when other modules have already finished the training process.

4. CONCLUSION AND FUTURE RESEARCHES

The efficiency analysis of coarse-grain parallel algorithm of modular neural networks training using visualization technique is considered in this paper. A standard MPE library is used for visualization. During researches there is corrected the disadvantage of a standard MPE library, based on astronomic time measurement. This disadvantage has become apparent on the parallel computer Origin 300. Modification of standard MPE library by usage of functions of processor time measurement allows accurate visualizing parallel routine executing without any influence of other computational tasks of parallel computer. As a result, decreasing of efficiency of IHDNNs parallel training algorithm is caused by load imbalance among processors not latency during communication. This load imbalance is characterized by non-stationary and non-uniform distribution of the IHDNNs training time. Therefore we pay our attention for development of fine-grain parallelization techniques for artificial neural networks in the future, which allows parallelizing each module of neural network independently.

5. ACKNOWLEDGEMENTS

Dr. Volodymyr O. Turchenko, the corresponding author of this paper, would like to thank European Organization INTAS for financial support of this research within Postdoctoral Fellowship INTAS

YSF 03-55-2493 "Development of Parallel Neural Networks Training Algorithms on Advanced High Performance Systems". This support is gratefully acknowledged.

6. REFERENCES

[1] Parallel processing of information: 5 volumes / AS USSR. Phys. – mech. institute. – K.: "Naukova dumka", 1984-1990. – Vol.5: Problem-oriented and specialized means of information processing / Aksenov A., Aristov V., Barsylovych E. et al.; Eds. B. Malinovsky and V. Hrytsyk. – 504 P.

[2] V. Kumar, A. Grama, A. Gupta, G. Karypis. Introduction to Parallel Computing. – CA (USA): Benjamin/Cummings, 1994.

[3] B.H.V. Topping, J. Sziveri, A. Bahreinejad, J.P.B. Leite, B. Cheng. Parallel processing, neural networks and genetic algorithms // Advances in Engineering Software. – 1998. – Vol. 29, No. 10. – P. 763–786.

[4] Chang L.-C., Chang F.-J. An efficient parallel algorithm for LISSOM neural network // Parallel Computing. – 2002. – Vol. 28, No. 11. – P. 1611-1633.

[5] Estévez P. A., Paugam-Moisy H., Puzenat D. et al. A scalable parallel algorithm for training a hierarchical mixture of neural experts // Parallel Computing. – 2002. – Vol. 28, No. 6. – P. 861-891.

[6] V. Turchenko. Parallel Algorithm of Dynamic Mapping of Integrating Historical Data Neural Networks // Information Technologies and Systems. – 2004. – No. 1. – Vol. 7. – No. 1. – pp. 45-52.

[7] Patent #50380 Ukraine, IPC 7 G06F15/18. Method of the training set formation for neural network predicting drift of data acquisition device / A.Sachenko (UA), V.Kochan (UA), V.Turchenko (UA), V.Golovko (BY), J.Savitsky (BY), T.Laopoulos (GR). – Filled 04 Jan 2000; Issued 15 Nov 2002. – 14 p.

[8] Sachenko, V. Kochan, V. Turchenko. Instrumentation for Data Gathering // IEEE Instrumentation and Measurement Magazine. – 2003. – Vol. 6, No. 3. – P. 34-40.

[9] V. Turchenko. Static Mapping of Integrating Historical Data Neural Networks on Parallel Computer // Proceedings of the 16th IASTED International Conference Parallel and Distributed Computing and Systems. – 2004. – Cambridge (MA, USA). – P. 884-889.

[10] National Science Foundation Science and Technology Center (NSF), MPI: A Message-Passing Interface Standard, 1995 – 239 p.

[11] J. Dongarra, D. Laforenza, S. Orlando (Eds), Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science #2840. – Berlin: Springer-Verlag, 2003. – P. 188-195.

[12] V. Turchenko, V. Kochan, A. Sachenko et al. Enhanced method of historical data integration using neural networks // Sensors and Systems. – 2002. – Vol. 7 (38). – P. 35-38.

[13] Chan, W. Gropp, E. Lusk. User's Guide for MPE: Extensions for MPI Programs. Technical report ANL/MCS-TM-ANL-98/xx, Argonne National Laboratory, 1998, pp. 1-31

[14] S. Moore, D. Cronk, K. London and J. Dongarra. Review of Performance Analysis Tools for MPI Parallel Programs, 2001, 8 p (http://icl.cs.utk.edu/publications/pub-papers/2001/perftools-review2.pdf).

[15] V. Turchenko, B. Demchuk. Visualization Tools of Processor Time of Parallel Programs, Scientific Journal of Khmelnitsky National University, 2005, Vol. 2, No. 4, pp. 146-151

[16] T. Senta, A. Takahashi, T. Kadoi et al. Itanium2 32 way Server System Architecture // NEC Research and Development. – 2003. – Vol. 44. – P. 8-12.

[17] http://www.cs.indiana.edu/classes/b673/notes/HTML/jumpshot.html

Volodymyr Turchenko *received his M.S. degree from Brest Polytechnic Institute, Belarus in 1995 and Ph.D. degree from Lviv National Technical University, Ukraine in 2001 both in computer engineering.*

Now he is an Assistant Professor of the Information Computing System and Control Department of the Institute of Computer Information Technologies. He is the leader of Neural Network Research Group. His main research interests are intelligent instrumentation, distributed sensor networks and neural networks.

Viktor Demchuk *from 2002 year student of Institute of computers information technologies of the Ternopil academy of national economy. From 2004 year works as the technician of educational computer laboratory of department of the informative computer systems and management. Now he is an engineer of International Scientific Journal of Computing. Areas of scientific interests: neural networks, parallel algorithms, high performance computing, grid.*