Interactive Environment for comparative Analysis of Sequential and Parallel Algorithms

MANUELA PANOIU, IONEL MUSCALAGIU, CAIUS PANOIU, MARIA RAICH
Electrical Engineering and Industrial Informatics Department
Polytechnic University of Timisoara
Revolutiei Street, no 5 Hunedoara, cod 331128
ROMANIA
{m.panoiu, mionel, c.panoiu }@fih.upt.ro

Abstract: - this paper presents interactive software which shows and analyzes through visual simulation some sequential and parallel algorithms. It was study the memory access for PRAM model and some significant algorithms for PRAM model. It was analyze also some sorting algorithms and some graph algorithms in both variants sequential and parallel. The software was implemented in Java. It was also performed a comparative study between a classic sequential algorithm and a parallel algorithm in terms of execution times.

Key-Words: - Java, educational software, parallel algorithms, simulation software

1 Introduction

Today, the computers are used in many areas, including in education, from the elementary school to the universities. The purpose of using computers in education can be realized by using new didactical teaching methods based on educational software that is programs useful in teaching-learning process [1], [2].

The advantages of using computers in education are obviously: the students have the possibility to interact with virtual environment, this method being preferred in comparison with lecturing of a classic material. By using the computer in didactical activity the learning productivity is increased: the necessary information is faster and accurate obtained, process and sends, by eliminate unnecessary time delay. It can be showing to the student some information which is inaccessible otherwise: dynamic diagrams, moving images, sounds.

Teaching computers to novices has proven to be a challenge for both teachers and students. Many students find the computers programming module difficult and disheartening and this could have an impact on their attitude to software development throughout the course and as a career choice. For staff involved in teaching computers programming it can also be very disheartening when student apparently fail to understand and be able to use even the basic data structures [2].

In computer science, a parallel algorithm, as opposed to a traditional sequential (or serial) algorithm, is an algorithm which can be executed a piece at a time on many different processing devices, and then put back together again at the end to get the correct result.

Most of today's algorithms are sequential, that is, they specify a sequence of steps in which each step consists of a single operation. These algorithms are well suited to today's computers, which basically perform operations in a sequential fashion. Although the speed at which sequential computers operate has been improving at an exponential rate for many years, the improvement is now coming at greater and greater cost. As a consequence, researchers have sought more cost-effective improvements by building "parallel" computers – computers that perform multiple operations in a single step [3].

In order to solve a problem efficiently on a parallel machine, it is usually necessary to design an algorithm that specifies multiple operations on each step, i.e., a parallel algorithm. The designer of a sequential algorithm typically formulates the algorithm using an abstract model of computation called the random-access machine (RAM). In this model, the machine consists of a single processor connected to a memory system [3].

Modeling parallel computations is more complicated than modeling sequential computations because in practice parallel computers tend to vary more in organization than do sequential computers [8]. As a consequence, a large portion of the research on parallel algorithms has gone into the question of modeling, and many debates have raged over what the "right" model is, or about how practical various models are [3], [8].

A multiprocessor model is a generalization of the sequential RAM model in which there is more than one processor. Multiprocessor models can be classified into three basic types: local memory machine models, modular memory machine models, and parallel random-access machine (PRAM) models. The purpose of the theoretical models for parallel computation is to give frameworks by which we can describe and analyze algorithms. These ideal models are used to obtain performance bounds and complexity estimates. One of the models that have been used extensively is the parallel random access machine (PRAM) model [3].

2. Description of the interactive environment

The target of this application is to help students in understanding the parallel and sequential algorithms. Another goal for this paper is to show a comparative study between a classic sequential algorithm and a parallel algorithm in terms of execution times.

The application was implemented in Java as independent application. The application can easily convert in a Java applet.

The main options of the application are: Parallel algorithms simulation for the PRAM model, Sorting algorithms (sequential and parallel) and Graphs algorithms. For parallel execution simulation we implement a class Processor:

```
class Processor extends Thread {
Processor( String nume )
      { super( nume ); }
      public void run() { }
int sum(int x,int y,int z,int w)
      { return x+y+z+w; }
      int product(int x, int y)
      { return x*y; } }
```

2.1 The Graphical User Interface for parallel algorithms simulation on PRAM model

This option present several algorithms developed for the theoretical PRAM model: EREW PRAM model, the CRCW PRAM model, the CREW PRAM model. The main interface is show in fig. 1.

Figure 1 show the simulation for the algorithm that calculates the sum of the elements of an array using the most restrictive PRAM algorithm, i.e. EREW algorithm. The algorithm is [3]:

```
Algorithm Sum_EREW
for i=1 to log n do
  forall $P_j$, where $1 \le j \le n/2$
           do in parallel
      if (2j modulo $2^i$)=0 then
        A[2j] ← A[2j] + A[2j-$2^{i-1}$]
        endif
    endfor
endfor
```

The algorithm complexity can be characterized by:

– Run time, T(n) = O(log n).
– Number of processors, P(n) = n/2.
– Cost, C(n) = O(n log n).

The students can see which processors works at the time and how the sum (or product, or minimum or maximum) of the elements of an array is determined. The animation can be stooped and restart. For a better accuracy the processors are displayed with different colors.

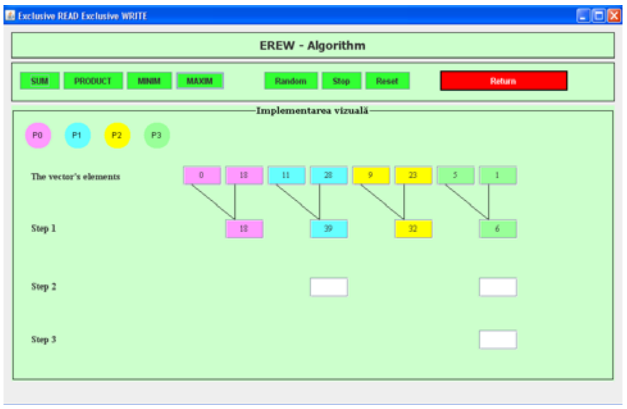


Fig. 1. The visual simulation for the parallel algorithm for EREW PRAM model

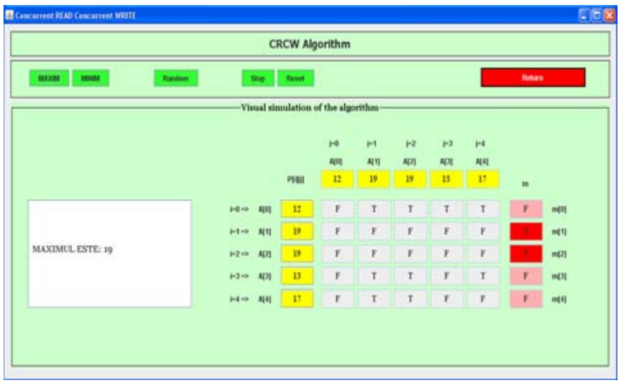
Another option of the application is for the CRCW PRAM model and simulates an algorithm for obtaining maximum (or minimum) element from an array. The Graphical User Interface presenting this option is show in figure 2. It can be seeing the active processors (n^2 in this case). These processors can determine the maxim (or minim) in only three steps regardless of the array length.

The algorithm used here for simulation is show bellow:

```
n<-length[A]
for i<-1 to n, do in parallel
                m[i]<-true
 for i<-1 to n,j<-1 to n,
   do in parallel
     if A[i] < A[j] then m[i]<-false
for i<-1 to n, do in parallel
     if m[i]=true then max<-A[i]
       return max
```

The algorithm complexity can be characterized by:

– Run time, T(n) = O(1).
– Number of processors, P(n) = n x n.
– Cost, C(n) = $n^2 \times$ O(1) =O(n^2).


Concurrent READ Concurrent WRITE
CRCW Algorithm
MAXIM
MINIM
Random
Stop
Reset
Return
Visual simulation of the algorithm

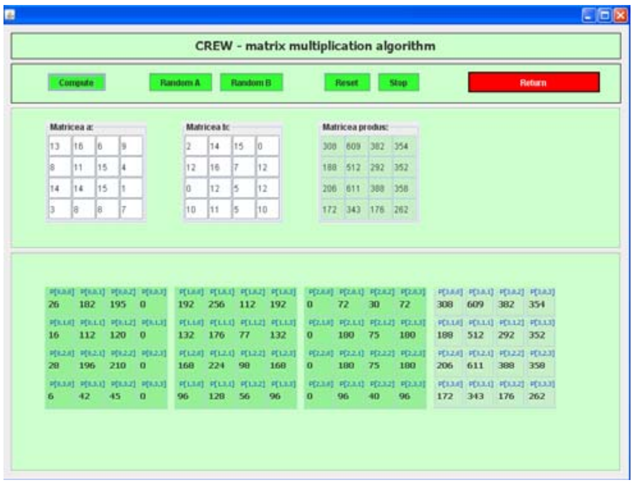
MAXIMUL ESTE: 19

		j=0	j=1	j=2	j=3	j=4		
		A[0]	A[1]	A[2]	A[3]	A[4]		
	P[i][j]	12	19	19	13	17	m	
i=0 -> A[0]	12	F	T	T	T	T	F	m[0]
i=1 -> A[1]	19	F	F	F	F	F	T	m[1]
i=2 -> A[2]	19	F	F	F	F	F	T	m[2]
i=3 -> A[3]	13	F	T	T	F	T	F	m[3]
i=4 -> A[4]	17	F	T	T	F	F	F	m[4]

Fig. 2. The visual simulation for the parallel algorithm for CRCW PRAM model

Figure 3 show the simulation for the matrix multiplication for the CREW PRAM model. The algorithm used for simulation here is:

```
/* Step 1 */
forall  Pi,j,k where $1 \le i,j,k \le n$
   do in parallel
$C[i,j,k] \leftarrow A[i,k] * B[k,j]$
endfor
/* Step 2 */
for l =1 to log n do
forall Pi,j,k, where $1 \le i,j,k \le n$ &
$1 \le k \le n/2$ do in parallel
if $(2 \cdot k \bmod 2^l) = 0$ then
$C[i,j,2k] \leftarrow C[i,j,2k] + C[i,j,2k-2^{l-1}]$
endif
endfor
/* The output matrix is stored in
locations $C[i,j,n]$, $1 \le i,j \le n$   */
```



Matricea a:

13	16	6	9
8	11	15	4
14	14	15	1
3	8	8	7

Matricea b:

2	14	15	0
12	16	7	12
0	12	5	12
10	11	5	10

Matricea produs:

308	609	382	354
188	512	292	352
206	611	388	358
172	343	176	262

P[0,0,0]	P[0,0,1]	P[0,0,2]	P[0,0,3]	P[1,0,0]	P[1,0,1]	P[1,0,2]	P[1,0,3]	P[2,0,0]	P[2,0,1]	P[2,0,2]	P[2,0,3]	P[3,0,0]	P[3,0,1]	P[3,0,2]	P[3,0,3]
26	182	195	0	192	256	112	192	0	72	30	72	308	609	382	354
P[0,1,0]	P[0,1,1]	P[0,1,2]	P[0,1,3]	P[1,1,0]	P[1,1,1]	P[1,1,2]	P[1,1,3]	P[2,1,0]	P[2,1,1]	P[2,1,2]	P[2,1,3]	P[3,1,0]	P[3,1,1]	P[3,1,2]	P[3,1,3]
16	112	120	0	132	176	77	132	0	180	75	180	188	512	292	352
P[0,2,0]	P[0,2,1]	P[0,2,2]	P[0,2,3]	P[1,2,0]	P[1,2,1]	P[1,2,2]	P[1,2,3]	P[2,2,0]	P[2,2,1]	P[2,2,2]	P[2,2,3]	P[3,2,0]	P[3,2,1]	P[3,2,2]	P[3,2,3]
28	196	210	0	168	224	98	168	0	180	75	180	206	611	388	358
P[0,3,0]	P[0,3,1]	P[0,3,2]	P[0,3,3]	P[1,3,0]	P[1,3,1]	P[1,3,2]	P[1,3,3]	P[2,3,0]	P[2,3,1]	P[2,3,2]	P[2,3,3]	P[3,3,0]	P[3,3,1]	P[3,3,2]	P[3,3,3]
6	42	45	0	96	128	56	96	0	96	40	96	172	343	176	262

Fig. 3. The visual simulation for the matrix multiplication for CREW PRAM model

The algorithm complexity can be characterized by:

– Run time, T(n) = O(log n).
– Number of processors, P(n) = n3.
– Cost, C(n) = O(n^3 log n).

2.2. The Graphical User Interface for analyzing sorting algorithms

Since the dawn of computing, the sorting problem has attracted a great deal of research, due to the complexity of solving it efficiently despite its simple, familiar statement [10]. For example, bubble sort was analyzed as early as 1956. Although many consider it a solved problem, useful new sorting algorithms are still being invented (for example, library sort was first published in 2004). Sorting algorithms are prevalent in introductory computer science classes, where the abundance of algorithms for the problem provides a gentle introduction to a variety of core algorithm concepts, such as divide and conquer algorithms, data structures, randomized algorithms, best, worst and average case analysis, time-space tradeoffs, and lower bounds.

Sorting is one of the fundamental problems of computer science, and parallel algorithms for sorting have been studied since the beginning of parallel computing [5], [6], [7], and [11]. For parallel implementation of the sort algorithms was implement a class. So each processor is represented here by a separate java class, extended from Thread class:

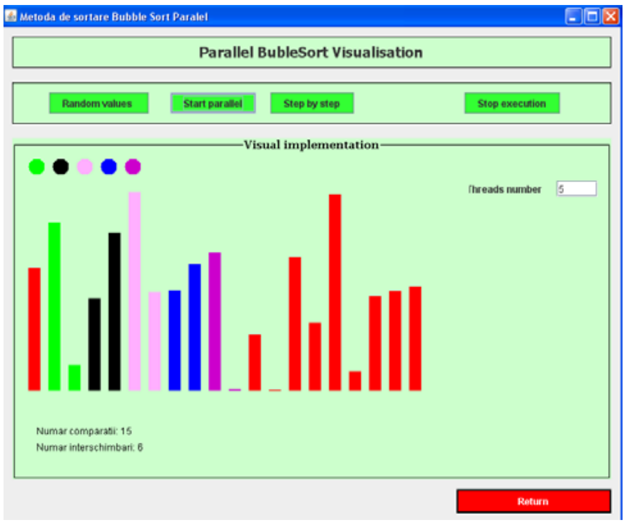
```
public    class    ThreadSort    extends
Thread{
 Color cf;
public ThreadSort (Color cf) {
    this.cf = cf; }
int compare(int a,int b)
{ if (a>b) { return 1; }
      else return -1;
}
```

The application presented here show in an interactive way, both sequential and parallel sorting algorithms. The algorithm for parallel BubbleSort is:

```
int aux,i,i1,i2,j=-1,cmp=0,chng=0,nf;
boolean ok;
ThreadSort f[] = new ThreadSort[nf];
for(i=0;i<nf;i++) {
    f[i] = new ThreadSort(null);
    f[i].start();
}
do {
     ok=true;
     j=-1;
   for (i1=0;i1<(n-1);i1++,i1++)
   { cmp++;
     j++;
    if(f1[j].compare(a[i1],a[i1+1])>0)
      { ok=false; chng++;
```

```
        aux=a[i1]; a[i1]=a[i1+1];
                    a[i1+1]=aux;
                   }
  if(j==(nf-1)||j!=(nf-1)&&i1==(n-2))
          j=-1;
          j1=-1;
  for (i2=1;i2<(n-1);i2++,i2++)
   { cmp++; j++;
    if f1[j].compare(a[i2],a[i2+1])>0)
      { ok=false; chng++;
        aux=a[i2]; a[i2]=a[i2+1];
                     a[i2+1]=aux;
                }
  if(j==(nf-1)||j!=(nf-1)&&i2==(n-3))
       j=-1;
      }
} while(!ok);
```

In figure 4 is show the GUI for visual simulation for parallel BubbleSort algorithm.



Threads number 5

Numar comparatii: 15
Numar interschimbari: 6

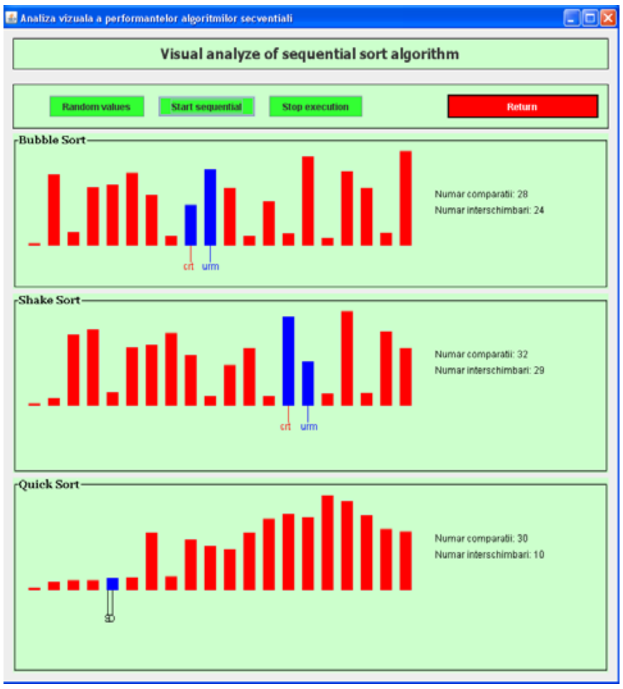
Fig 4. The visual simulation for parallel BubleSort algorithm

The visualization can be performed with a certain animation speed or step by step. It could be observe that the students have the opportunity to see each step of the algorithm by using the button “step by step” or the algorithm can be run with an animation speed. It is possible also to choose the number of processors before starting the algorithm. Each processor and the operation performed by this processor are represented with a different color for a better understanding.

In the same manner are implemented another two sort algorithms (sequential and parallel).

For comparing these sort algorithms another option was implemented. This option of the program allow to students to see the simulation for the three sorting algorithms in parallel. The GUI-s for this option is showing in figure 5, for sequential algorithms and in fig. 6 for parallel algorithms. It can be see and compare the algorithms performances, i.e. number of comparisons, number of assignments.

The comparative analyze is made both for sequential and parallel algorithms.

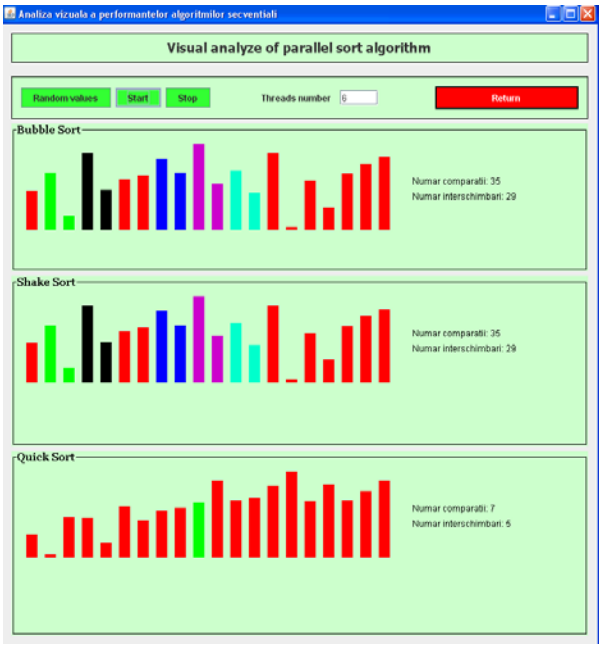

Analiza vizuala a performantelor algoritmilor secventiali
Visual analyze of sequential sort algorithm
Random values
Start sequential
Stop execution
Return
Bubble Sort
crt
urm
Shake Sort
crt
urm
Quick Sort

Bubble Sort
Numar comparatii: 28
Numar interschimbari: 24

Shake Sort
Numar comparatii: 32
Numar interschimbari: 29

Quick Sort
Numar comparatii: 30
Numar interschimbari: 10

Fig. 5. The simulation for the three sequential sorting algorithms in parallel


Analiza vizuala a performantelor algoritmilor secventiali
Visual analyze of parallel sort algorithm
Random values
Start
Stop
Return
Bubble Sort
Shake Sort
Quick Sort

Threads number 6

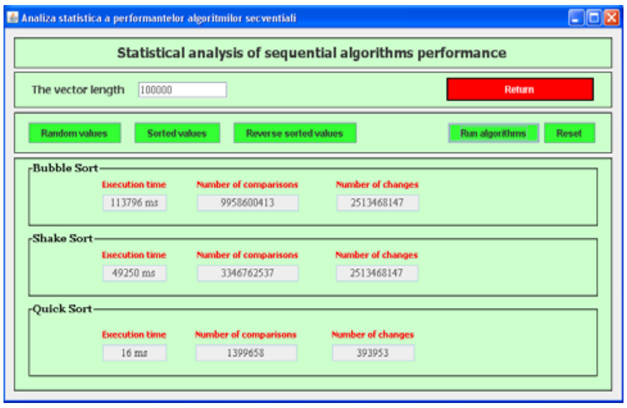
Bubble Sort
Numar comparatii: 35
Numar interschimbari: 29

Shake Sort
Numar comparatii: 35
Numar interschimbari: 29

Quick Sort
Numar comparatii: 7
Numar interschimbari: 6

Fig. 6. The simulation for the three parallel sorting algorithms in parallel

For comparing the order of complexity of these algorithms was made another option. In this case, the students can choose the numbers of the elements, as can be seen in fig. 7. To compare the execution time is recommended to use a large number of elements. Here, the students can generate random a list of integer numbers to be sorted, or the numbers can be provided sorted or reverse sorted. In this case the algorithms are in background running using different threads and then their performances are displayed.



Statistical analysis of sequential algorithms performance

The vector length 100000

	Execution time	Number of comparisons	Number of changes
Bubble Sort	113796 ms	9958600413	2513468147
Shake Sort	49250 ms	3346762537	2513468147
Quick Sort	16 ms	1399658	393953

Fig. 7. The statistical analyze for the three sequential sorting algorithms

It was made such an options for statistical analyze of parallel sorting algorithms also.

2.3. The Graphical User Interface for analyzing graph algorithms

Graphs and graph algorithms are at the centre of the solutions to many real-world problems. Graph algorithms have been studied extensively for many years [12].

The application present here show a visual simulation of some classic, well know graph algorithms like Prim algorithm, Kruskal algorithm and Dijkstra's algorithm. It was made visual simulation for sequential algorithms. For Prim's algorithm and Djikstra's algorithm was made visual simulation also for parallel algorithm.

Fig. 8 shows the visual simulation of the Prim' algorithm and fig 9 show the visual simulation of the Kruskal' algorithm. The algorithms can be seeing in two ways: step by step or by using animation with a choose animation speed.

In this way the students can understand better each algorithm step and how it is implemented.

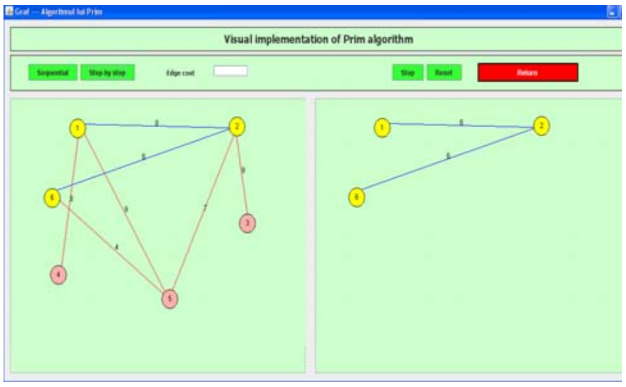


Fig. 8. The visual simulation for Prim's algorithm

The graph can be build by mouse click and drag in the left panel. The steps of algorithm can be seeing in the right panel.

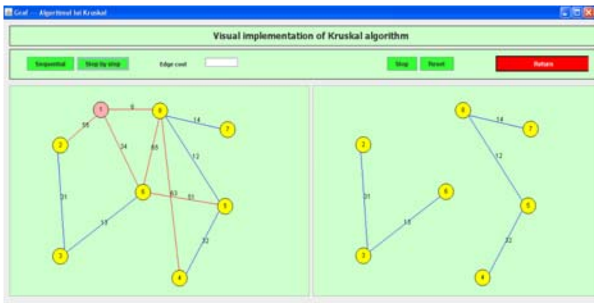


Fig. 9. The visual simulation for Kruskal's algorithm

Conclusions

By using visual simulations in computer assisted learning the efficiency of learning is increased. There are some subjects in fundamentals of computer science like sorting algorithms, parallel algorithms that are very difficult to learn and understand for some students. By using educational software these concepts are showing to the students in an attractive way, visually using animation and having the opportunity to interact with the application [9].

The best option is to use graphical and interactive interface of the software in two ways. On one hand, these software help the teacher in the classroom, while on the other hand, the students can work and experiment with them making their own examples, out the classroom.

In general, interactive educational software is very good aids for learning algorithms and data structures, as they improve comprehension and the satisfaction of the students, as well as the interest and motivation amongst students when the teacher makes use of them [4].

References

[1]Glenn, W., Rowe, Gregor, P., „*A computer based learning system for teaching computing: implementation and evaluation*" Computer and Education, Elsevier Science, 1999.

[2] Khamis N, Idris S, *Issues and Solutions in Assessing Object-oriented Programming Skills in the Core Education of Computer Science and Information Technology*, 12th WSEAS International Conference on Computers, JUL 23-25, 2008 Heraklion, GREECE, Pages: 458-463, 2008

[3] Hesham El-Rewini, Mostafa Abd-El-Barr, Advanced Computer Architecture and Parallel Processing, Wiley-Interscience, A Son Wiley &Sons Inc Publication, 2005

[4] L. M. Carmona, C. Escribano, A. Giraldo, M. A. Sastre, Interactive Web Tutorial for Integer and Modular Arithmetic and its Applications. Proceedings of the 6th WSEAS International Conference on Education and Educational Technology (EDU'07), Venice, Italy, November 2007

[5]Alak Kumar Dattaa, Ranjan Kumar Sen, o($\log^4$ n) time parallel maximal matching algorithm using linear number of processors, Parallel Algorithms and Applications, Vol. 19 (1) March 2004, pp. 19–32

[6]Nancy M. Amato Ravishankar Iyer Sharad Sundaresan Yan Wu, A Comparison of Parallel Sorting Algorithms on Different Architectures, Technical Report 98-029 Department of Computer Science Texas A&M University, January 1996

[7]Robert Rönngren, Rassul Ayani, A Comparative Study of Parallel and Sequential Priority Queue Algorithms, ACM Transactions on Modeling and Computer Simulation, Vol. 7, No. 2, April 1997, Pages 157–209.

[8]Guy E. Blelloch, Vector Models for Data-Parallel Computing, The MIT Press Cambridge, Massachusetts

[9]Zamfir PDA, Impact of using computer applications in education on teaching-learning process, 7th WSEAS International Conference on Applied Computer and Applied Computational Science, Hangzhou, Pages: 684-688, 2008

[10] D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*.

[11] http://www.sorting-algorithms.com/

[12] Michael J. Quinn, Narsingh Deo, Parallel graph algorithms, ACM Computing Surveys, Volume 16 , Issue 3 (September 1984), Pages: 319 - 348 , 1984