Educational Software for Study the Performances of Some Known Parallel and Sequential Algorithms

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

Abstract: - The paper presents an educational software package, implemented in Java which shows and analyzes through visual simulation some well known sequential and parallel algorithms. It was study the memory access for PRAM model and some known algorithms for PRAM model. It was analyze also the most know sorting algorithms and the main graph algorithms in both variants sequential and parallel. It was also performed a comparative study between a classic sequential algorithm and a parallel algorithm in terms of execution times. The software is very useful to both students and teachers because the computer science especially computer programming is difficult to understand for most students.

Key-Words: - Java, educational software, parallel algorithms, Parallel sorting algorithms, performance analysis, 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 The informatics system design

In this paragraph will be presented the elaboration of an educational informatics system for study the performance and functionality of some know parallel and sequential algorithms.

2.1 System analysis

The informatics system will be described by presenting the use cases, using the UML (unified modeling language) [13].

Representation of the uses cases' diagram is shown in figure 1.

For each use case presented in the previous diagram we'll build activity diagram. Each diagram will specify the processes or algorithms which are behind the analysed use case. Figure 2 will present two of these diagrams. Fig. 2 a illustrate the activity diagram for the use case "EREW" from fig. 1 and fig 2b illustrate the activity diagram for the use case "Dijkstra - Analyse", also from fig.1.

Activity diagrams [14] are used:

- To capture actions to be performed when an operation is executing (most common purpose);
- To capture the internal work in an objects;
- To show how a set of related actions may be performed;
- To show how an instance of s use-case may be performed;
- To show a business works in terms of actors, workflows, organization, and objects.

2.2. System design

The conceptual modeling allows the identification of the most important concepts for the informatics system [17]. The existing inheritance relationships between the classes previously presented can be represented by means of relationship diagrams between classes. In fig. 3 is illustrating the class diagram for "Graphs" part of the software package.

For describe the flow of messages between objects it was use the sequence diagrams which focus on the order in which the messages are sent. They are very useful for describing the procedural flow through many objects. They are also quite useful for finding race conditions in concurrent systems.

In fig. 4 is illustrate such a sequence diagram for the use case "Analyze Prim Kruskal".

The sequence diagrams for this software are made with ArgoUML.

Collaboration diagram [15], on the other hand, focus upon the relationships between the objects. They are very useful for visualizing the way several objects collaborate to get a job done and for comparing a dynamic model with a static model.

Collaboration and sequence diagrams describe the same information, and can be transformed into one another without difficulty. The choice between the two depends upon what the designer wants to make visually apparent. In figure 5, and 6 are illustrates collaboration diagrams.

In fig. 7 is show the state diagram for sorting part of the software for graph part of the software.

2.3. The component diagram

A component represents a modular, replaceable piece in the system [16]. Of primary importance are two well-defined interfaces: The required interface specifies formally which functionality the component expects from its environment. The provided interface specifies the functionality the component is able to provide (to other components). In fig. 8 is shown the component diagram for the software package.

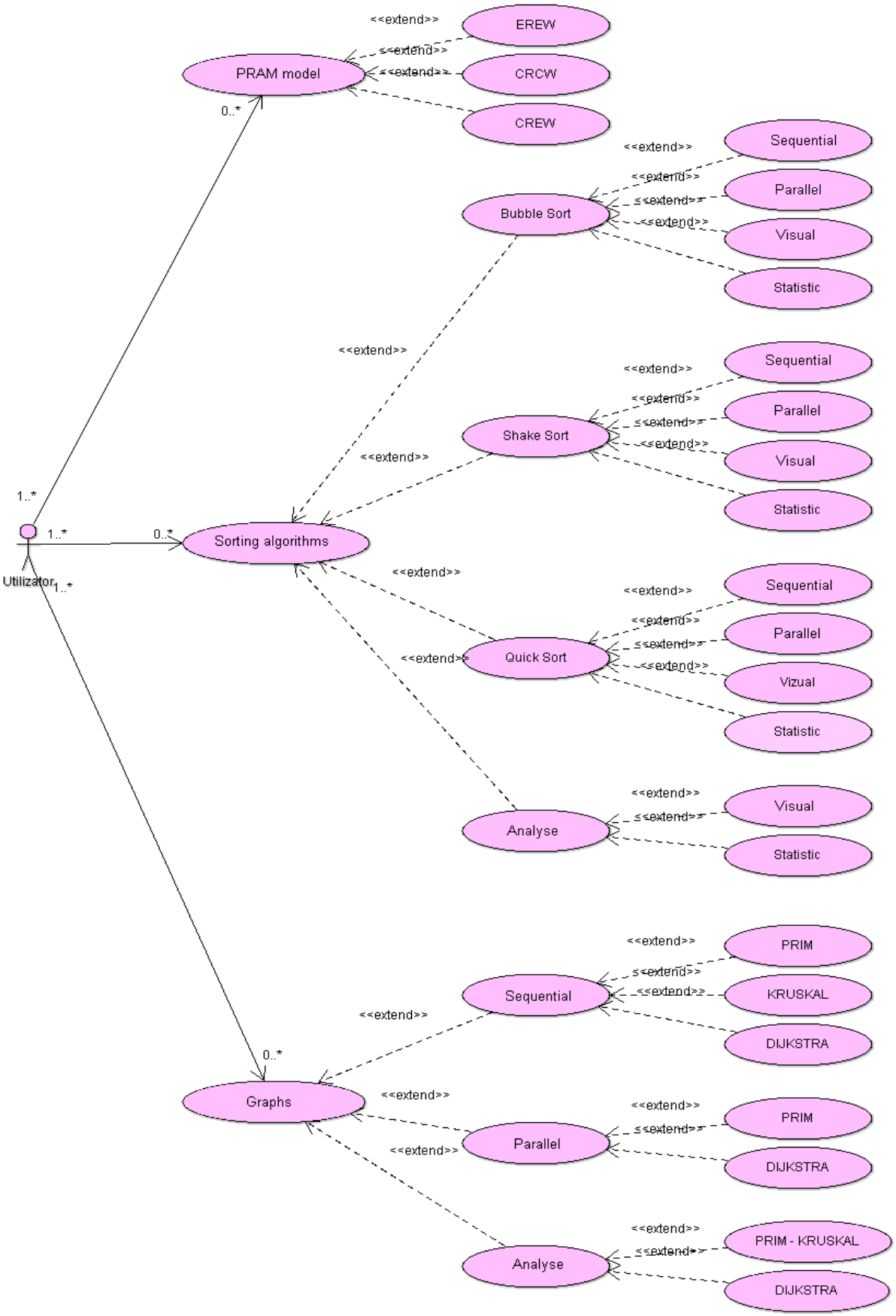
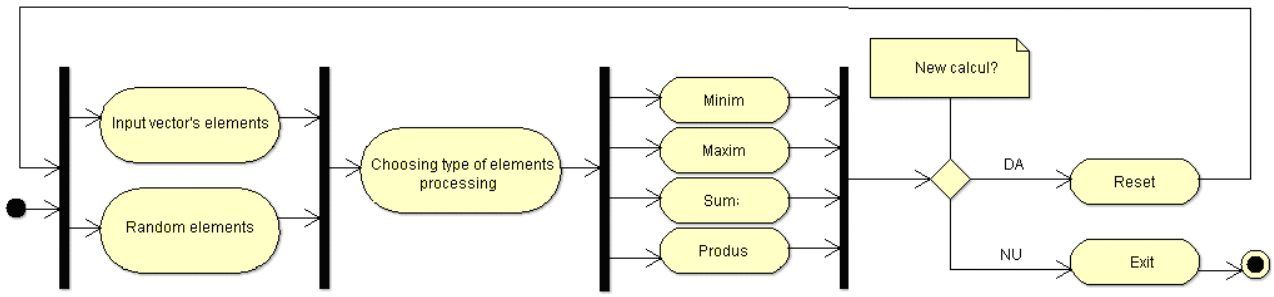
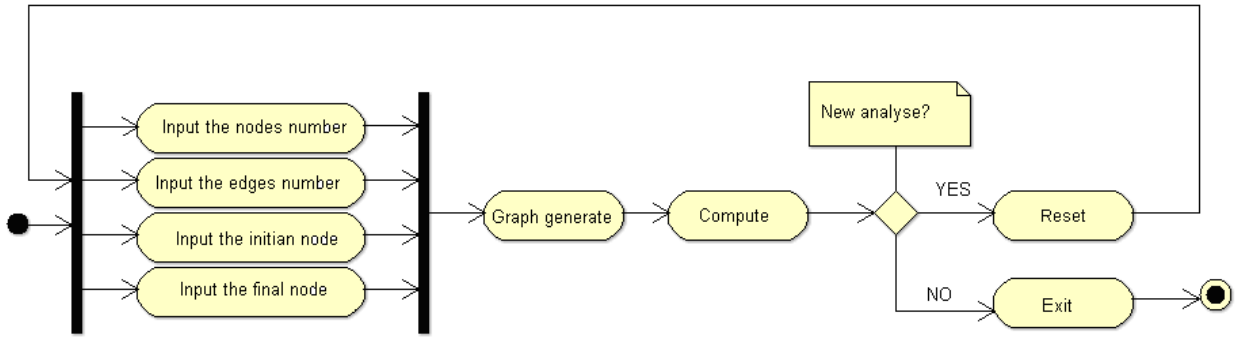


Fig. 1. The use case diagram



a) The activity diagram for "EREW"



b) Activity diagram for "Dijkstra analyse"

Fig 2. Activity diagrams

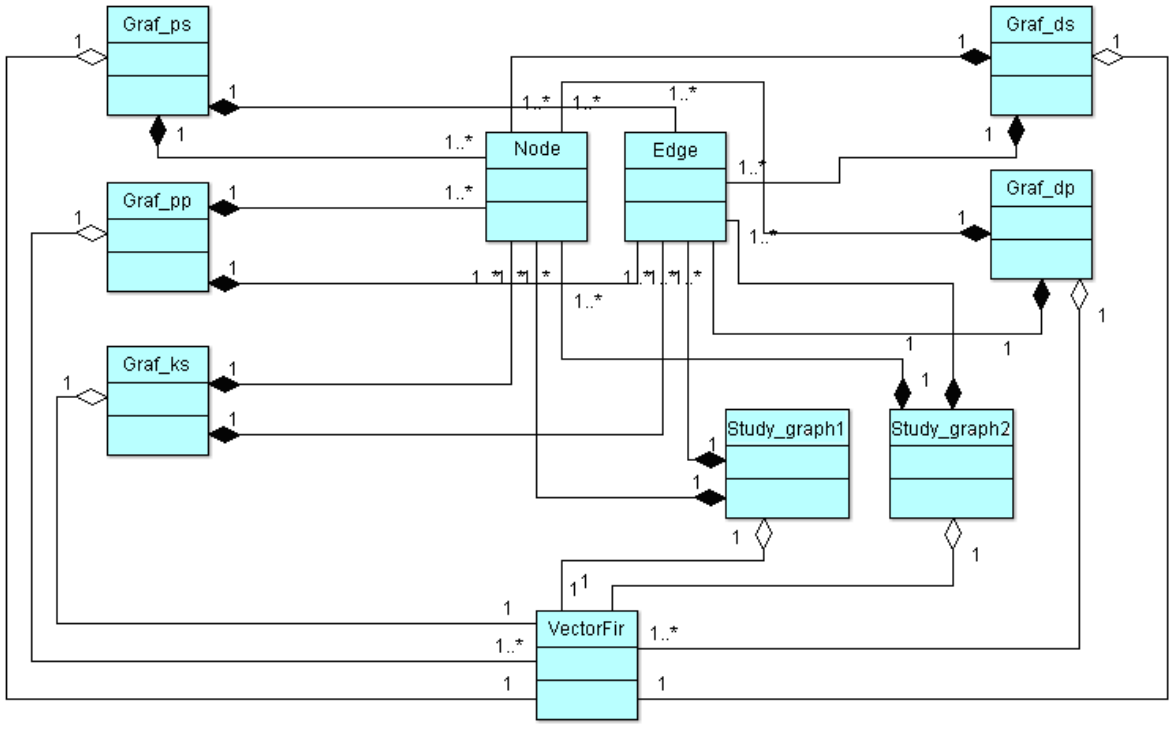


Fig. 3. The class diagram for "Graphs"

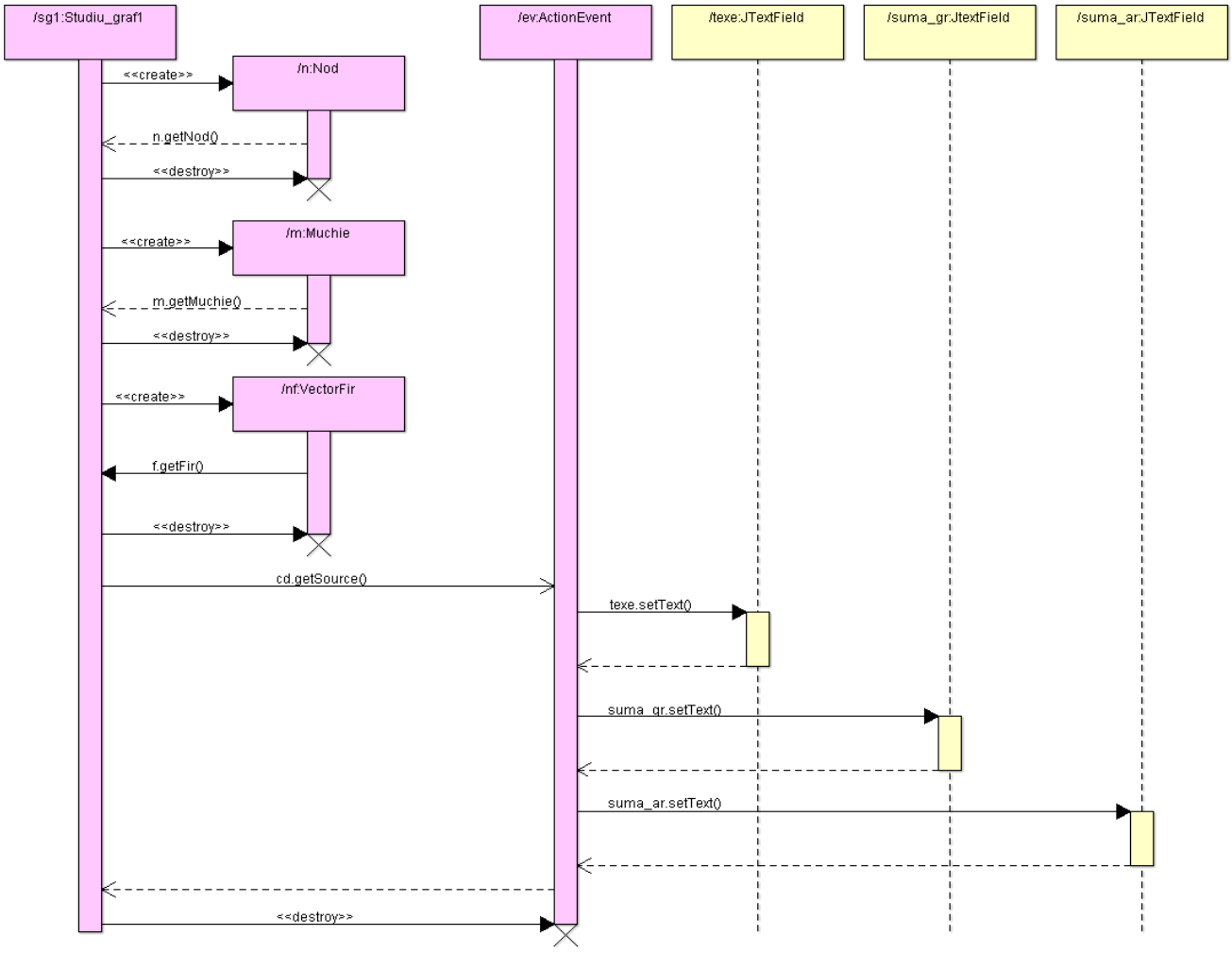


Fig. 4. The sequence diagram for the use case “Analyze Prim Kruskal”

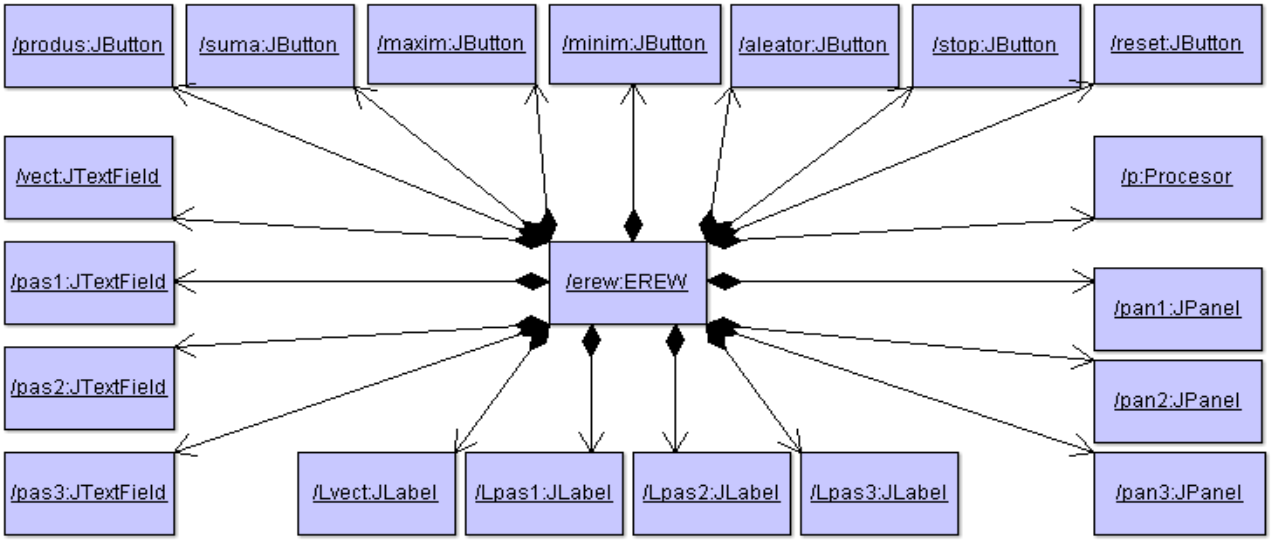


Fig. 5. The collaboration diagram for the use case “EREW”

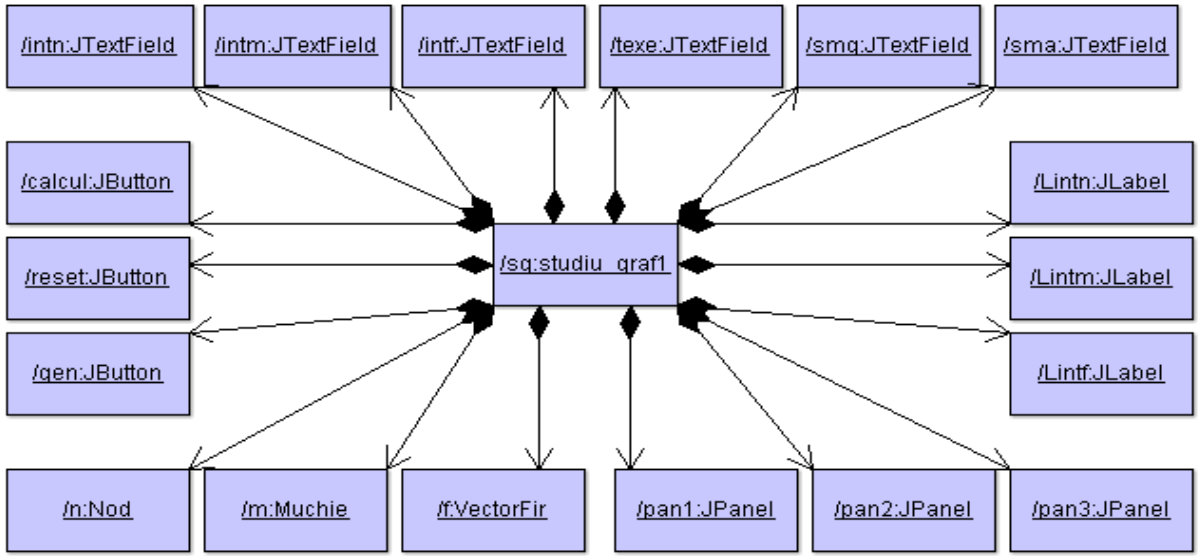


Fig. 6. The collaboration diagram for the use case "Analyze Prim Kruskal".

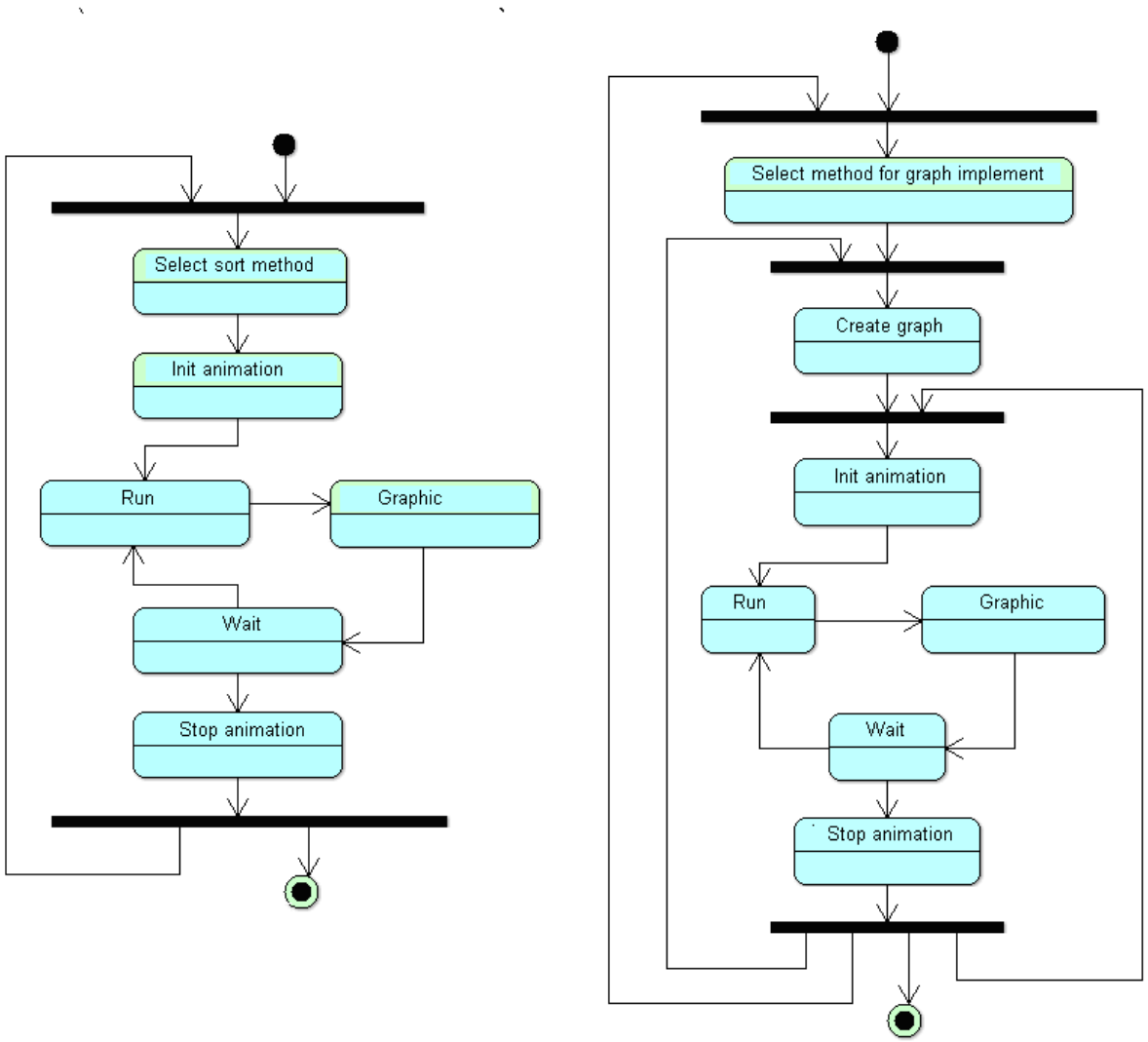


Fig. 7. The state diagrams for sorting algorithms and for graphs algorithms

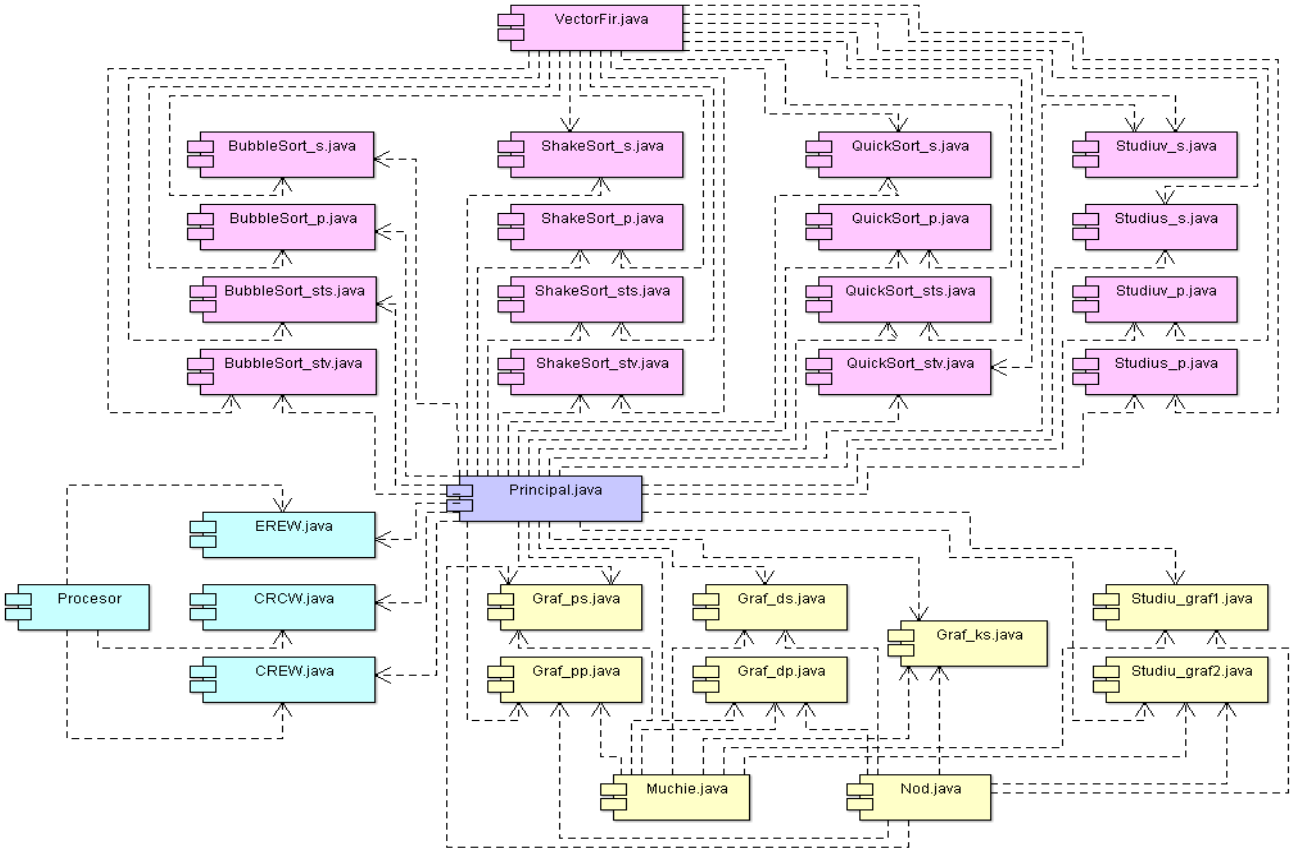


Fig. 8. The component diagram for the software package

3 The software implementation

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 Procesor extends Thread {

Procesor( String name ) { super( name
); }

public void run() {}

  int sum(int x,int y,int z,int w)
    { return x+y+z+w; }

  int sumP(int a, int b)
    { return a+b; }

  long prodP(long a, long b)
    { return a*b; }
  int product(int x, int y)
    { return x*y; }
  int compP1(int a, int b)
    { if (a>=b) { return 1; }
        else return -1;
    }

  int compP2(int a, int b)
    { if (a<=b) { return 1; }
        else return -1;
    }
  }
```

3.1 The Graphical User Interface for study PRAM model of parallel processing

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. X. Figure 9 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 EREW
      for i=1 to log n do
forall   Pj,   where   1≤ j ≤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.

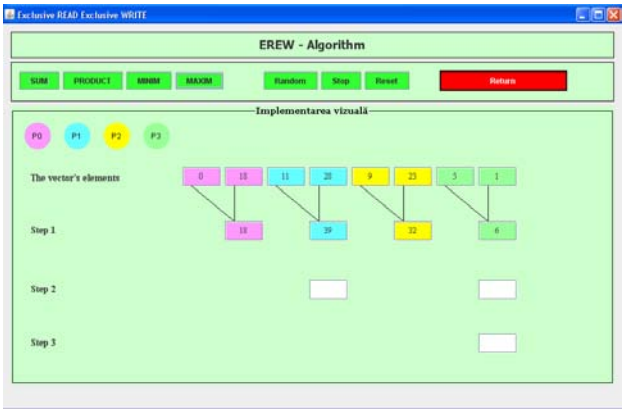


Fig. 9. The visual simulation for the parallel algorithm for EREW PRAM model

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 10. 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 and 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) = $n^2 \times O(1) = O(n^2)$.

Fig. 10. The visual simulation for the parallel algorithm for CRCW PRAM model

Figure 11 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≤i,j,k≤n
   do in parallel
C[i,j,k]← A[i,k]*B[k,j]
endfor
/* Step 2 */
for l =1 to log n do
forall  Pi,j,k,   where   1≤i,j,k≤n   &
1≤k≤n/2  do in parallel
if            (2·k mod 2^l)=0            then
C[i,j,2k]← 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≤i,j≤n  */
```

The algorithm complexity can be characterized by:

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

Fig. 11. The visual simulation for the matrix multiplication for CREW PRAM model

3.2 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 12 is show the GUI for visual simulation for parallel BubbleSort algorithm.

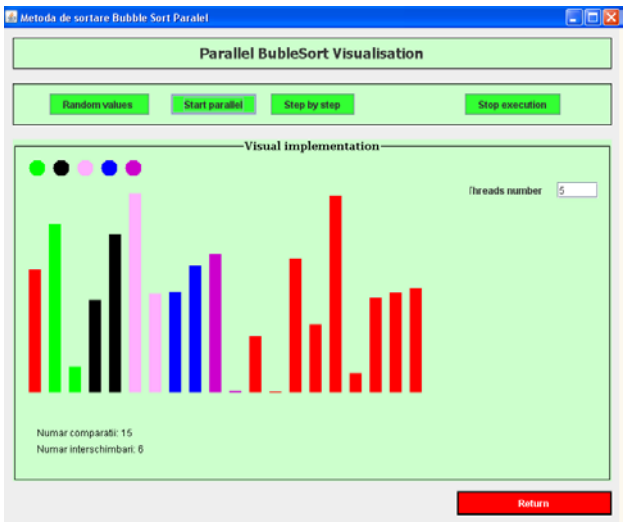


Fig 12. The visual simulation for parallel BubleSort algorithm.

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 13, for sequential algorithms and in fig. 14 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.

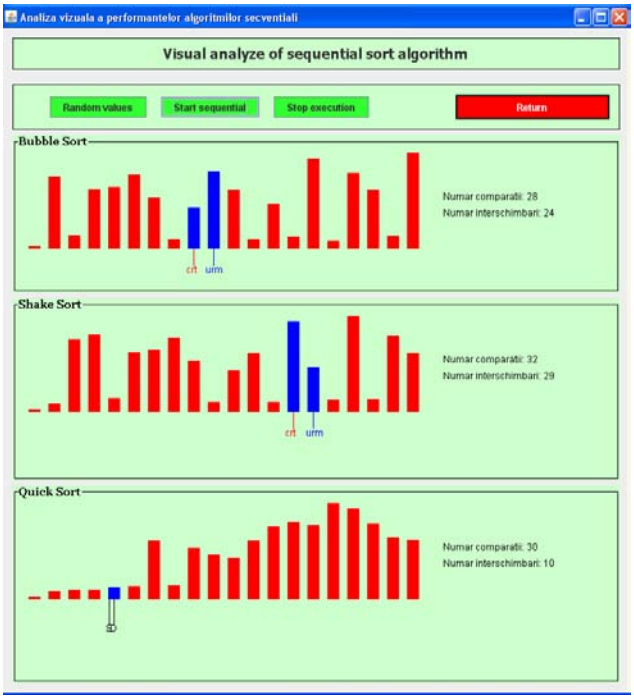


Fig. 13. The simulation for the three sequential sorting algorithms in parallel

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. 15. 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.

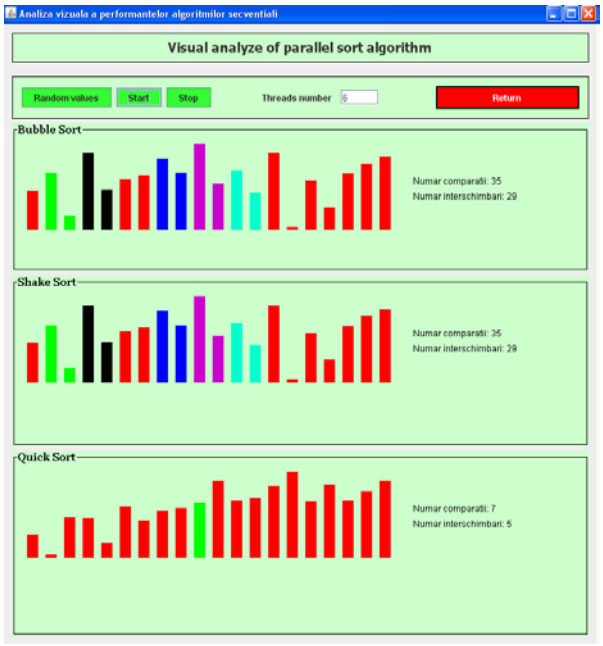


Fig. 14. The simulation for the three parallel sorting algorithms in parallel

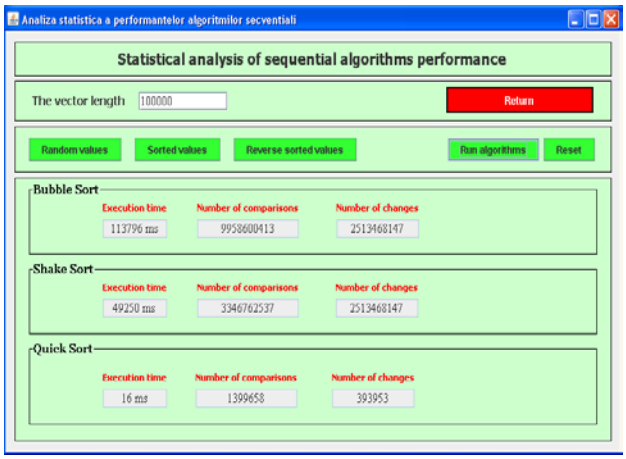


Fig. 15. The statistical analyze for the three sequential sorting algorithms

It was made such an options for statistical analyze of parallel sorting algorithms also. By running the application for several random data sets the students can better analyze the performance of these algorithms. In the table 1, 2 and 3 the performances are shown for randomly generating data and for 10 respectively 50 threads. For each case are shown the running time, the number of comparisons and the number of interchanges for a certain number of elements.

Table 1. The sorting algorithms performances for 1000 randomly elements

Sorting method	Number of threads	Run time	Number of comparisons	Number of interchanges
Bubble Sort	10	46 ms	488.511	251.081
Shake Sort	10	39 ms	488.511	251.081
Quick Sort	10	0 ms	7.327	2.722
Bubble Sort	50	78 ms	492.517	257.813
Shake Sort	50	72 ms	492.517	257.813
Quick Sort	50	0 ms	7.586	2.839

Table 2. The sorting algorithms performances for 10000 randomly elements

Sorting method	Number of threads	Run time	Number of comparisons	Number of interchanges
Bubble Sort	10	2.324 ms	49.335.066	25.025.395
Shake Sort	10	2.315 ms	49.335.066	25.025.395
Quick Sort	10	15 ms	107.726	34.743
Bubble Sort	50	2.813 ms	49.625.391	23.916.405
Shake Sort	50	2.806 ms	49.625.391	23.916.405
Quick Sort	50	16 ms	116.359	36.815

Table 3. The sorting algorithms performances for 100000 randomly elements

Sorting method	Number of threads	Run time	Number of comparisons	Number of interchanges
Bubble Sort	10	263.344 ms	4.994.850.051	2.496.049.103
Shake Sort	10	249.577 ms	4.994.850.051	2.496.049.103
Quick Sort	10	172 ms	1.310.092	430.877
Bubble Sort	50	596.284 ms	4.987.536.709	2.514.037.442
Shake Sort	50	574.328 ms	4.987.536.709	2.514.037.442
Quick Sort	50	164 ms	1.289.530	429.563

By analyze these results the students can see that the Quicksort method has been performing better. The run time increases with increasing the number of threads because of using a single processor computer**.** If the application is running to a multiprocessor computer, the run time decrease with the increasing of threads number. The students can run the application for a large number of elements or for other cases.

3.3. 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. 16 shows the visual simulation of the Prim' algorithm and fig 17 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.

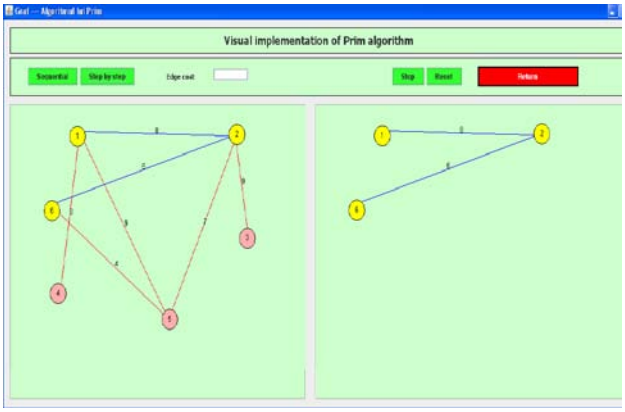


Fig. 16. The visual simulation for Prim's algorithm

The graphs nodes can be build by mouse click and drag in the left panel. The edges can be built by click in the start node, then drag to the end node. The steps of algorithm can be seeing in the right panel by using step and step option or by using an animation speed.

For study the graphs algorithms it was implement an option presented in fig. 18. In this window the students can see the run time and the total cost of the entire graph or of the minimum spanning tree in case of both algorithms, Prim and Kruskal.

In table 4 and 5 are shown some results obtained by using this option.

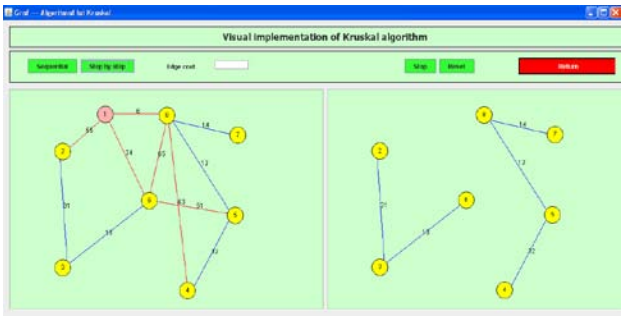


Fig. 17. The visual simulation for Kruskal's algorithm

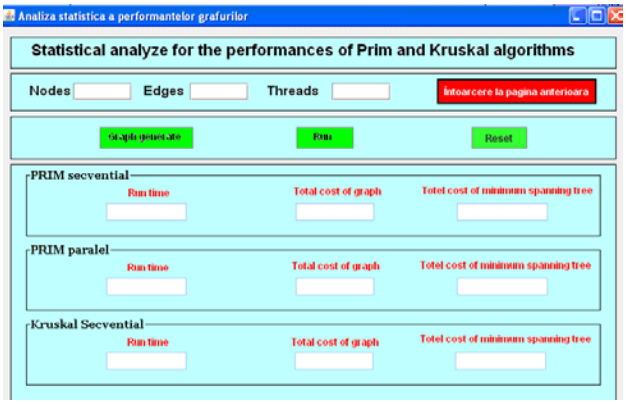


Fig. 18 Analyze of performances of Prim and Kruskal algorithm

.

Table 4. The algorithms results for 500 nodes, 500 edges and 10 threads

The algorithm	Run time	Total cost	The cost of minimum spanning tree
Prim sequential	38.734 ms	25.219	2.854
Prim parallel	30.281 ms	25.219	2.854
Kruskal sequential	2.609 ms	25.219	5.411

Table 5. The algorithms results for 100 nodes, 1000 edges and 10 threads

The algorithm	Run time	Total cost	The cost of minimum spanning tree
Prim sequential	2.110 ms	48.633	343
Prim parallel	1.625 ms	48.633	343
Kruskal sequential	375 ms	48.633	462

In the same way it was implemented an option for study the performances of Dijkstra algorithm both for sequential and parallel implementation. In fig. 19 is show the window for study the performance of of Dijkstra algorithm both for sequential and parallel implementation.

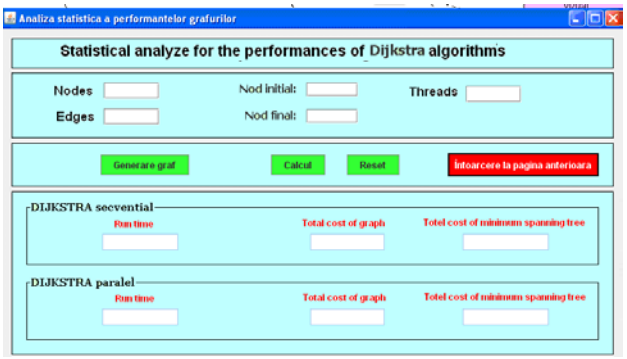


Fig. 19 Analyze of performances Dijkstra algorithms

In case of running application to an multiprocessor computer, the parallel implementation has better performances.

4. Conclusion

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(log4 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] [12] Michael J. Quinn, Narsingh Deo, Parallel graph algorithms, *ACM Computing Surveys*, Volume 16 , Issue 3 (September 1984), Pages: 319 - 348 , 1984

[13] G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1999

[14] R. Eshuis, R. Wieringa, *A formal semantics for UML activity diagrams – Formalising workflow models*, University of Twente, Departament of Computer Science, 2001

[15] D. Rosenberg, K. Scott, Use case Driven Object Modeling with UML, Addison Wesley, 1999

[16] J. Odell, Advanced Object Oriented Analysis& Design using UML, *Cambrige University Press*, 1998

[17] Iordan Anca, Development of Interactive Software for Teaching Three-Dimensional Analytic Geometry, *9th WSEAS International Conference on Distance Learning and WEB Engineering*, Budapest, Hungary, 3-5 Sept, 2009