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 an 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:

```java
class Processor extends Thread {
    Processor(String nume) {
        super(nume);
    }
    public void run() {
        // Code here
    }
    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≤j≤n/2
        do in parallel
            if (2j modulo 2^i)=0 then
    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 stopped and restarted. 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 maximum (or minimum) 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
    endfor
for i<-1 to n, do in parallel
    if m[i]=true then max<-A[i]
endfor
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 x O(1) = O(n^2).
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 \(1 \leq i,j,k \leq n\) do in parallel
\[C[i,j,k] \leftarrow A[i,k] \times B[k,j]\]
endfor

/* Step 2 */
for \(l = 1\) to \(\log n\) do
forall Pi,j,k, where \(1 \leq i,j,k \leq n\) & \(1 \leq k \leq n/2\) do in parallel
if \((2 \cdot k \mod 2^l) = 0\) 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 \(C[i,j,n], 1 \leq i,j \leq n\) */

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:

```java
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:

```java
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<nf-1;i1++,i1++)
{ cmp++;
j++;
if(f1[j].compare(a[i1],a[i1+1])>0)
{ ok=false; chng++;
}
```

The algorithm complexity can be characterized by:

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

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.
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.

![Fig. 7. The statistical analyze for the three sequential sorting algorithms](image)

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 Dijkstra’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](image)

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](image)

### 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


