AN IMPLEMENTATION OF THE GREEDY ALGORITHM FOR MULTICORE SYSTEMS

Doru Anastasiu POPESCU
University of Pitești, Romania
Faculty of Mathematics and Computer Sciences
dopopan@gmail.com

Keywords: Parallel programming, algorithm, programming techniques, Java

Abstract: *The great variety of parallel systems that appeared in the last period and use multicore chips leads inevitably to the development of applications with parallel algorithms. A multicore algorithm uses several threads in the process of searching information. In this paper we will present a general multicore algorithm for Greedy method and an implementation in Java programming language of an algorithm that uses this method.*

1. INTRODUCTION

The multi-core processors are characterized by the fact that they split their problems in problems solved by simpler cores. This leads to the increase of the number of the problems that can be solved and occasionally to the reduction of solving time of these.

Technical aspects regarding the functioning of the systems using multicore processors are presented in [1], [5] and [6].

Computers from the last generation are equipped with multicore processors, so they impose the usage of parallel algorithms for solving problems, in order to benefit from their advantages. Versions of parallel algorithms were written for various programming methods in order to make specific implementations. In papers [2], [3] and [4] various types of parallel algorithms for backtracking methods are presented and their implementation is made with threads from Java language.

In the next sections we will present the general form of Greedy method by using a parallel version and an example of its usage. The implementation of the parallel algorithm will be made using threads in Java programming language.

An important notion used in parallel algorithms is *thread* which can be defined as an execution entity inside a process (details in [8]), formed from a context and a sequence of execution instructions.

These have some important characteristics:

- Threads are used to create programs formed from concurrent processing units.
- The entity thread executes a sequence time by encapsulated instructions in function of thread.
- The execution of a thread could be interrupted for permitting the processor to give control to a thread.
- Threads are treated independent by the process itself or by the operating system nucleus. The component system (process or nucleus) which manages the threads depends on their implementing method.

2. GREEDY METHOD WITH PARALLEL ALGORITHMS

Using Greedy method problems with a set of data denoted by A as input data, a condition and a subset B of set A that verifies the given condition is required can be solved. Generally, the condition from the statement is an optimal one and therefore it should be mathematically demonstrated that Greedy method leads to a correct solution.

Greedy method provides a single solution that verifies the condition from the statement, unlike backtracking method, which determines all the solutions. This would be a disadvantage of Greedy method, but a major advantage is that the execution time is polynomial related to the dimension of set A, unlike backtracking method where time is exponential.

Greedy method is used in many areas of Informatics such as Graph Theory, Image and Signal Processing, Bioinformatics etc. Kruskal's algorithm, Prim's algorithm for determining the minimum spanning tree, the continuous version of knapsack algorithm are some greatly important algorithms that use Greedy method.

The general form of the parallel algorithm for Greedy method is presented in the next rows.

Step 1. The elements of set A are read or determined by various methods.

Step 2. The number of elements n of set B is read.

Step 3. $B \leftarrow \emptyset$; $k \leftarrow 0$

Step 4. The elements from set A are rearranged, eventually using information derived from the condition that set B must verify.

Step 5. *while* $A \neq \emptyset$ *and* $k < n$ *do*

a from A that verifies a propriety

obtained from the problem conditions using

a secondary thread is determined

if a exists then

$k \leftarrow k + 1$

$B \leftarrow B \cup \{a\}$

$A \leftarrow A \setminus \{a\}$

end if

end while

Step 6. The elements from B are written.

Observations

1. The algorithm is working properly when the problem has solution. The condition from while instruction should be completed to signalize the non-existence of a solution for some input data and to avoid a cyclic redundancy in step 5.
2. Every step of while instruction from step 5 should be executed with a single thread to obtain an efficient algorithm regarding execution time. If the algorithm is used in this way, the algorithm concurrence must be taken into account, i.e. the situations when the same values for a from set A are obtained for two or more threads. This aspects are particular with the problems.
3. The correctness of the algorithm must be demonstrated mathematically or by another means.

3. CASE STUDY

We suggest to solve a problem using the algorithm described in section 1.

Statement

A natural number n with maximum 7 digits is given. Determine the first n natural numbers which can be written as a sum of two primes (e.g. 4=2+2, 5=3+2, 6=3+3).

Solution

The Java program that uses threads write every element of set B as it is determined. The set A is formed from the primes lower or equal with a high enough value so that every number from set B could be written as a sum of two primes.

We will present next the source code that uses multi-core Greedy method.

```
import java.io.*;
class Fir extends Thread
{
public static long numar, sw, a, b;

public Fir(long numar) {this.numar = numar;}

public long prim(long k){
//check if k is prime
long i;
if (k<2)
 return 0;
for(i=2;i<=Math.sqrt(k);i++)
 if(k%i==0)
  return 0;
return 1;
}

public void run()
{
//write the source code thread
long i;
for(i=2; i <= numar; i++)
 if(prim(i)==1 && prim(numar-i)==1){
  sw=1;a=i;b=numar-i;
  break;
 }
}

public static void main(String[] args)
{
long n, k;
System.out.print("n=");
n=Long.parseLong(cin.linie());
k=0;numar=4;
while(k<=n){
 sw=0;
 new Fir(numar).start();
 // we launch a thread execution
 if(sw==1){
  System.out.println(numar+"="+a+"+"+b);
```

```
  k++;
 }
 numar++;
}
}
}
```

The **cin** class presented below was used to ease the way of data reading.

```
import java.io.*;
public class cin
{
       static String linie()
       {
              String sir="";
              char ch;
              try
              {
                     while(
(ch=(char)System.in.read())!=13)
                                  sir=sir+ch;
              }
              catch(IOException e){}
                     try{
System.in.read(); }
              catch(IOException e){}
              return sir;
       }
}
```

The results obtained with this program are presented in figure 1.

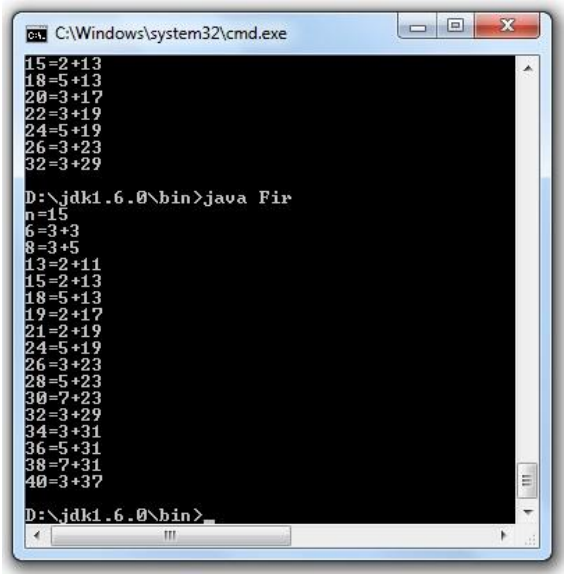

Fig. 1 The results for n=15

A more complex problem would be the determination of the first n natural numbers that can be written as a sum of p distinct primes, being given a natural number lower or equal with 10.

In this case a solution would propose using backtracking method in the form from [2], [3] or [4] to obtain an algorithm with a better execution time.

4. ADVATAGES AND DISADVANTAGES OF PARALLEL ALGORITHMS

The usage of parallel algorithms in solving some problems can be a good or bad solution because the performance of these algorithms depends on various factors, from which we remind:

- the unbalance of processors (the impossibility of distribution in perfect equally tasks and the variation of parallelism point inside the algorithm);
- the additional calculus which appear in case the fastest sequential algorithm cannot be paralleled and a slow parallel algorithm which can be paralleled is chosen;
- the communication between processes;
- the concurrence for the shared data set.

One of the most important characteristics of a parallel algorithm is the partition of the problem in sub-problems which can be solved on several threads.

For the projection of a parallel algorithm a series of approaches can be considered. The first one would be the parallelization of an existent sequential algorithm. For this the parallelism which appears naturally inside a sequential algorithm must be determined.

Sometimes, finding a different solution from the one offered by the sequential algorithm is preferred and this supposes re-thinking the entire algorithm. Indifferently of the approach manner inside a parallel algorithm, a series of important considerations cannot be ignored. One of these is the communication cost between the processes, [9].

If the cost or complexity of a sequential algorithm is expressed in space (the volume of the occupied memory) and time (the number of operations made) necessary for executing a program, at the parallel one the way of communication between the processes must be considered.

Related with the algorithm presented in section 3, advantages to the sequential variant do not exist either from the memory point of view. For the values used at testing the execution times for the two variants were close. But a simple modification of the parallel algorithm from the section 3 regarding the increment of *k* in *run* function and not in *main* led to obtaining less good times in the parallel variant than in the sequential one.

5. CONCLUSIONS

The paper proposed a model of using Greedy method with threads in Java programming language, in order to emphasize a category of applications specific to multi-core systems.

In papers [5] and [6] there are presented multiple aspects specific to multi-core systems that should be taken into account when systems with parallel algorithms are projected.

In order to have at hand general mechanisms of solving problems by parallel programming, general methods of solving problems are useful to be known, such as backtracking, greedy, divide et impera, dynamic programming in multicore versions.

6. REFERENCES

[1]. Chu M., Ravindran R., Mahlke S., *Data Access Partitioning for Fine-grain Parallelism on Multicore Architectures*, Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, 2007, 369-380.

[2]. Hamadi Y., Bessire C., Quinqueton J., *Backtracking in distributed constraint networks*, in Proceedings ECAI98, Brighton, UK, 1998, 219-223.

[3]. Giurgiu C.-F., *An implementation of the backtracking algorithm for distributed systems*, Analele Universitatii de Vest din Timisoara, Seria Matematica-Informatica, Vol- ume XLVI, Issue 1, 2008, 61-74.

[4]. Giurgiu C.-F., *An implementation of the backtracking algorithm for multicore systems*, ROMJIST, Volume 13, Number 3, 2010, 241–254

[5]. Pankratius V., Schaefer C., Jannesari A., Tichy W.F., *Software engineering for multicore systems: an experience report*, Proceedings of the 1st international workshop on Multicore software engineering, Leipzig, Germany, 2008, 53-60.

[6]. Zivan R., Meisels A., *Synchronous and Asynchronous Search on DisCSPs*, in Proceedings of the First European Workshop on Multi-Agent Systems (EUMA), Oxford, UK, 2003.

[7]. http://ro.wikipedia.org/wiki/Algoritmi_de_calcul_paralel

[8]. Allen I. Holub, Taming Java Programming Language Threads, JavaOne, 2001

[9]. Herbert Schildt, Java: The Complete Reference, Seventh Edition, 2007 by The McGraw-Hill Companies.