USING PARALLEL PROCESSES FOR SOLVING THE KNAPSACK PROBLEM

Marek Kňaze

Penka Martincová

Department of Informatics, Faculty of Management and Informatics, University of Zilina, Slovak Republic

Introduction

One of the most popular problems of discrete programming is the Knapsack problem. Given a set of items, each with cost and value, determine the number of each item to include in a collection so that the total cost is less than some given cost and the total value is as large as possible. This problem is known to be NP-hard.

We can express this problem as linear programming model as follows:

$$\max z = c_1 x_1 + c_2 x_2 + c_3 x_3 + \ldots + c_n x_n \tag{1}$$

under conditions

$$a_1 x_1 + a_2 x_2 + a_3 x_3 + \ldots + a_n x_n \leq k \tag{2}$$

$$x_j \in \{ 0,1 \} \quad \text{for } j = 1,2,..,n \tag{3}$$

where c_1, c_2, $c_{3,}$,...c_n - are costs of the particular items, which can be stored into the knapsack; a_1, a_2, a_3,...a_n - are value of the items; k - Knapsack capacity; n - a number of items, which can be stored in the knapsack; (1) is an object function of the problem; (2) is a condition which ensures that the total cost is less than the given cost; (3) is a condition which ensures *undevisibility of the items*, which means that bivalent variable x, assigned to the item has a value 1 if an item is included in the knapsack or 0 in the oposite case.

This problem is solvable by *branch and bounds* method, using binary tree. Tree branches represent solution of including or non-including the item into the knapsack. Sequential solution of this problem is as follows: at the beginning the upper bound is equal to the sum of costs of all itmes. As first is processed the first item. If its value is less than knapsack capacity, it is included to the collection. We proceed with adding items until the knapsack capacity is exceeded. In this moment backtracking is applied. This means, that we return back to the previous tree level and proceed with non-including this item to the collection. This process continues until the tree bottom (solution about the last item) is riched. Here again backtracking is applied and we select a branch, which wasn't treated yet.

This method is too long, because an algorithm treats also subtrees, which can't improve the cost of the solution, already found. For reducing this steps the upper bound is used. After each operation of including (or non-including) an item to the knapsack the upper bound for particular subtree is found. Upper bound equals to the cost sum of all included items and all items, which can be added further. So, if subtree treating wouldn't improve solution, found till this moment, algorithm will not treat it. This speeds up a solution.

1. Parallel implementation of the sequential algorithm

Knapsack problem can be parallelized using *partitioning method*. This means that binary tree is divided into subtrees and every process treats assigned subtree (Figure 1).

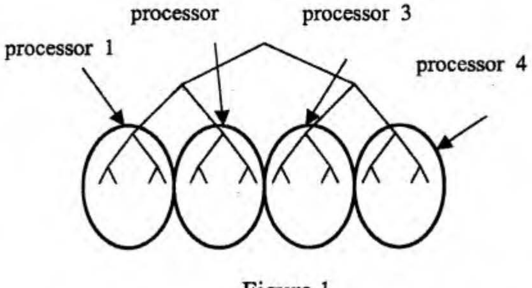

Figure 1

Master process divides a tree into subtrees and passes input parameters to the slave processes. After that every process treats one part from the tree and returns a value of its solution. Master process evaluates all solutions and selects the optimal solution of the problem.

Pseudo-codes of master and slave process for 2 processors are as follows:

Master proces:

```
Read_input_data();
Define_slave_input_params();
Send_data_to_slave();
Treat_subtree();
Get_solution_from_slave();
Find_optimal_solution().
```

Slave proces:

```
Read_input_data();
Get_input_params_from_the_master();
Treat_subtree();
Send_solution_to_the master().
```

2. Experiments

Knapsack problem was solved using Parallel Virtual Machine (PVM) running under Linux operating system on network of personal computers.

Algorithm described before was corrected and executed on 5 computers. Input data: n=40, knapsack capacity=266.

Results from the average executing time measuring are shown in figure 2.

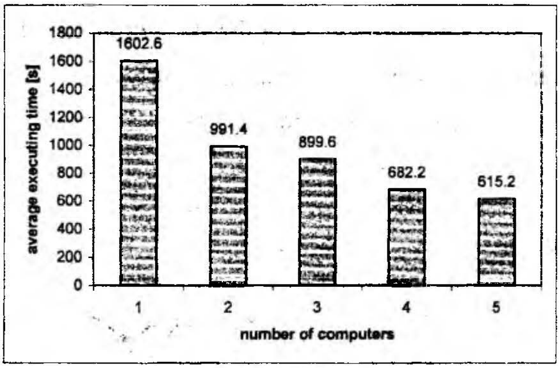

Figure 2

4. Conclusion

We can see from the figure 2 that executing time for solving of Knapsack problem became significantly shorter due to parallel processing. But also it is clear, that dependence between the executing time and the number of computers is not linear. The reason is as follows: when the number of computers is not equal to *n-th* power of 2, the status tree is not divisible to equal subtrees and some of processes have to treat bigger subtrees and consequently need longer time (see differences of executing times for 1 and 2 computers and 2 and 3 computers). The best results are achievable when the number of computers is 2^n, for n>0.

Using of parallel processes for solving time consuming optimization problems is reasonable and leads to significant improving of executing time. Parallel processes can be used also in the cases when expensive multiprocessor computer is not available.

References

Operačná analýza, Jaroslav Janáček, Žilinská univerzita, 1997

PVM: Parallel Virtual Machine,A.Geist&col. Massachusetts Institute of Technology, 1994

Teória grafov, Stanislav Palúch, Žilinská univerzita, 2001

www.csm.ornl.gov

www.netlib.org