International Journal of Parallel Programming, Vol. 21, No. 5, 1992

A Parallel Algorithm for the 0–1 Knapsack Problem

W. Loots[1] and T. H. C. Smith[1]

Received July 1992; Revised April 1993

Research efforts on parallel exact algorithms for the 0–1 knapsack problem have up to now concentrated on solving small problems (at most 1,000 objects) and in many cases results have only been obtained by simulation of the parallel algorithm. After a brief review of a well known sequential branch-and-bound algorithm we discuss a new parallel algorithm for the 0–1 knapsack problem which exploits the potential parallelism that exists during the backtracking steps of the branch-and-bound algorithm. We report results for our parallel algorithm on a transputer network for problems with up to 20,000 objects. The speedup obtained is nearly linear for 2, 4, and 8 processors except when there is a strong correlation between the profit and weight of the objects.

KEY WORDS: Knapsack problem; parallel processing; branch-and-bound method; transputer.

1. INTRODUCTION

The **0–1 knapsack problem** can be informally stated as follows: if a number of different objects are available, each with an associated profit and weight, from which a knapsack with a certain weight limit has to be filled, the problem is to find a combination of objects so that the total profit of the selected objects is maximized without exceeding the weight capacity of the knapsack.

Formally the 0–1 knapsack problem can be stated as the following 0–1 integer programming problem.[1] Let W denote the total weight capacity of the knapsack. Let n denote the number of available objects while p_j and w_j

[1] Department of Computer Science, Rand Afrikaans University, P.O. Box 524, Johannesburg, 2000, South Africa.

0885-7458/92/1000-0349$06.50/0 © 1992 Plenum Publishing Corporation

denote the profit and weight of the jth object, $j = 1, 2, \ldots, n$. The problem is then

$$\text{maximize} \quad \sum_{j=1}^{n} x_j p_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} x_j w_j \leqslant W \qquad \text{with} \quad x_j \in \{0, 1\}$$

The 0–1 knapsack problem is known to be a hard optimization problem. Furthermore the knapsack problem has a variety of practical applications such as the cargo loading problem and capital budgeting and often arises as a subproblem in other optimization problems such as the cutting stock problem.[1]

2. SEQUENTIAL AND PARALLEL ALGORITHMS

A number of sequential algorithms exist for the 0–1 knapsack problem. The survey of Martello and Toth[1] describes both dynamic programming and branch-and-bound algorithms. The branch-and-bound algorithms first sort the objects in nonincreasing order of their profit/weight ratios. Therefore the computation time consists of the sorting time and the branch-and-bound time. Computational results with n up to 10,000 are presented in the survey and also in more recent publications.[2, 3]

The first published research on parallel branch-and-bound algorithms for the 0–1 knapsack problem is by Lai and Sahni.[4] They consider the anomalies that can occur when parallelizing branch-and-bound algorithms by estimating the possible speedups using a simulation of a parallel version of the branch-and-bound algorithm of Horowitz and Sahni.[5] With $n = 100$ average speedups of 2.19, 3.69, and 6.47 are obtained for 2, 4, and 8 processors respectively. It is however important to remember that these simulated results ignore factors such as communication cost.

The paper by Janakiram *et al.*[6] describes a parallel branch-and-bound algorithm that employs randomness in its solution method. They use the 0–1 knapsack problem and the algorithm of Horowitz and Sahni to analyze the performance of their parallel algorithm. Their results were also obtained through simulation with $n = 30$. Speedups obtained range from 1.23 to 1.78 for 2 processors, 1.5 to 2.51 for 4 processors and 1.73 to 6.08 for 8 processors.

Kindervater and Trienekens[7] examine the implementation of combinatorial algorithms on three parallel computers. One of the problems considered is the 0–1 knapsack problem. The three computers used are the ILC-DAP (a SIMD computer), the CDC-Cyber-205 (a vector computer)