Available online at www.sciencedirect.com
SCIENCE DIRECT®
J. Parallel Distrib. Comput. 65 (2005) 74–84
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Load balancing methods and parallel dynamic programming algorithm using dominance technique applied to the 0–1 knapsack problem

D. El Baz*, M. Elkihel

LAAS-CNRS, 7, Avenue du Colonel Roche, 31077 Toulouse, Cedex 4, France

Received 7 July 2004

Abstract

The parallelization on a supercomputer of a one list dynamic programming algorithm using dominance technique and processor cooperation for the 0–1 knapsack problem is presented. Such a technique generates irregular data structure, moreover the number of undominated states is unforeseeable. Original and efficient load balancing strategies are proposed. Finally, computational results obtained with an Origin 3800 supercomputer are displayed and analyzed. To the best of our knowledge, this is the first time for which computational experiments on a supercomputer are presented for a parallel dynamic programming algorithm using dominance technique.

Keywords: Parallel computing; Load balancing; 0–1 knapsack problem; Dynamic programming; Dominance technique

1. Introduction

The 0–1 knapsack problem has been intensively studied in the literature (see for example [1,12,14,19,21,23–26]). The objective of this paper is to concentrate on the parallelization on a supercomputer of a one list dynamic programming method using dominance technique and processor cooperation and to propose efficient load balancing strategies in order to achieve good performance.

Many authors have considered dense dynamic programming, i.e., an approach for which one takes into account all possible states (see [2–5,7,15]). In this case, the number of states is equal to the capacity of the knapsack. In this paper, we study a different approach proposed by Ahrens and Finke (see [1]) which permits one to limit the number of states by using dominance technique. In this case, the number of states or undominated pairs generated is unforeseeable. If we compare the two approaches, then we note that the former will generate a regular data structure, from which one can easily deduce the total amount of work needed in order to treat the list. This approach leads generally to an easy parallelization; its main drawback is that it produces large lists in the case of problems with large capacity. The later approach presents the advantage to generate lists which are smaller; its main drawback is the creation of an irregular data structure. As a consequence, the parallelization of the later approach is not easy and the design of an efficient load balancing strategy is very important.

In [11], we have presented an original parallelization of the one list dynamic programming method using dominance technique. The cooperation via data exchange of processors of the architecture is the main feature of the proposed parallel algorithm. A first load balancing strategy was also proposed in [11]. In this paper, we develop the parallel algorithm, specially on a theoretical point of view and propose several original load balancing strategies.

Our contribution is different from the other works in the literature devoted to parallel dynamic programming for 0–1 knapsack problems. In particular, it is different from [6,18], where the authors have considered approaches based on massive parallelism. More precisely, in the above quoted papers, the authors have proposed solution for arrays with up to $O(2^{n/8})$ processors, where n is the number of variables in

* Corresponding author. Fax: +33 5 6133 6969.
E-mail addresses: elbaz@laas.fr (D. El Baz), elkihel@laas.fr (M. Elkihel).

0743-7315/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2004.10.004

the knapsack problem. Our work is also different from [8], where the authors have studied the parallel implementation of a two lists algorithm on an MIMD architecture for the solution of a particular class of 0–1 knapsack problems: the exact subset sum problem where profits are equal to weights. In this later approach, total work is decomposed initially and processors do not cooperate. Note that our parallel algorithm is designed for a broad class of 0–1 knapsack problems including subset sum problems. Moreover, our parallel algorithm presents better time complexity than the parallel algorithm studied in [8], as we shall see in the sequel. Reference is also made to [7,13] for different approaches concerning the parallelization of the dynamic programming method.

Section 2 deals with the 0–1 knapsack problem and its solution via dynamic programming. Parallel algorithm is studied in Section 3. Original load balancing strategies are proposed in Section 4. Finally, computational results obtained with an Origin 3800 supercomputer are displayed and analyzed in Section 5.

2. The 0–1 knapsack problem

The 0–1 unidimensional knapsack problem is defined as follows:

$$\max\left\{\sum_{j=1}^{n} p_j x_j \,|\, \sum_{j=1}^{n} w_j x_j \leqslant C;\ x_j \in \{0,1\},\ j=1,2,\ldots,n\right\}, \tag{1}$$

where C denotes the capacity of the knapsack, n the number of items considered, p_j and w_j, respectively, the profit and weight, respectively, associated with the jth item. Without loss of generality, we assume that all the data are positive integers. In order to avoid trivial solutions, we assume that we have: $\sum_{j=1}^{n} w_j > C$ and $w_j < C$ for all $j \in \{1,\ldots,n\}$. Several methods have been proposed in order to solve problem (1). We can quote for example: branch and bound methods proposed by Fayard and Plateau (see [12]), Lauriere (see [17]) and Martello and Toth (see [19]), methods based on dynamic programming studied by Horowitz and Sahni (see [14]), Ahrens and Finke (see [1]) and Toth (see [27]) and finally mixed algorithms combining dynamic programming and branch and bound methods presented by Martello and Toth (see [20]) and Plateau and Elkihel (see [26]).

In this paper, we concentrate on a dynamic programming method proposed by Ahrens and Finke whose time and space complexity are $O(\min\{2^n, nC\})$ (see [1]) and which is based on the concepts of list and dominance. We shall generate recursively as follows lists L_k of pairs (w, p), $k = 1, 2, \ldots, n$; where w is a weight and p a profit. Initially, we have $L_0 = \{(0, 0)\}$.

Let us define the set N_k of new pairs generated at stage k; where new pairs results from the fact that a new item, i.e., the kth item, is taken into account.

$$N_k = \{(w + w_k, p + p_k) \,|\, (w, p) \in L_{k-1}, w + w_k \leqslant C\}. \tag{2}$$

According to the dominance principle, which is a consequence of Bellman's optimality principle, all pairs $(w, p) \in L_{k-1} \cup N_k$ obtained by construction such that there exists a pair $(w', p') \in L_{k-1} \cup N_k$, $(w', p') \neq (w, p)$, which satisfies: $w' \leqslant w$ and $p \leqslant p'$, must not belong to a list L_k. In this case, we usually say that the pair (w', p') dominates the pair (w, p). As a consequence, any two pairs (w', p'), (w'', p'') of the list L_k must satisfy: $p' < p''$ if $w' < w''$. Thus, we can define the set D_k of dominated pairs at stage k as follows:

$$D_k = \{(w, p) \,|\, (w, p) \in L_{k-1} \cup N_k, \exists (w', p') \in L_{k-1} \cup N_k \text{ with } w' \leqslant w,\ p \leqslant p',\ (w', p') \neq (w, p)\}. \tag{3}$$

As a consequence, for all positive integers k, the dynamic programming recursive list L_k is defined as follows:

$$L_k = L_{k-1} \cup N_k - D_k. \tag{4}$$

Note that the lists L_k are organized as sets of monotonically increasing ordered pairs in both weight and profit. As a consequence, the largest pair of the list L_n, corresponds to the optimal value of the knapsack problem. We illustrate the dynamic programming procedure presented in this section on a simple example displayed as follows:

$$n = 6 \text{ and } C = 16, \tag{5}$$

$$(w_1, \ldots, w_n) = (5, 3, 2, 1, 5, 9), \tag{6}$$

$$p_1, \ldots, p_n) = (20, 8, 5, 4, 14, 27). \tag{7}$$

Table 1 shows the contents of the lists N_k, D_k and L_k, for $k = 1$–6.

We note that the optimal pair is $(16, 52) \in N_6$, the optimal solution corresponds to $(x_1, \ldots, x_n) = (1, 0, 1, 0, 0, 1)$. We see that infeasible pairs with total weight greater than $C = 16$, such as for example $(8+9, 29+27) = (17, 56)$ and $(9+9, 32+27) = (18, 59)$ do not belong to the list N_6. Finally, we note that the cardinality of D_k can become relatively large when k increases.

3. Parallel algorithm

In this section, we detail the parallelization of the one list dynamic programming method using dominance technique. The parallel algorithm which was briefly presented in [11] is designed according to the single program multiple data (SPMD) model for a parallel architecture that can be viewed as a shared memory machine on a logical point of view. As we shall see in detail in Section 5 experiments have been carried out on a nonuniform memory access (NUMA) supercomputer Origin 3800 by using the Open MP environment.