IEEE - ICSCN 2007, MIT Campus, Anna University, Chennai, India. Feb. 22-24, 2007. pp.218-221.

Analyzing Time Complexity of Parallel Algorithms for Knapsack Problem

S. Padmavathi[1] and S. Mercy Shalinie[1]

Abstract : **In this paper, we have analyzed various parallel algorithms for solving knapsack problem and proposed an efficient approach for solving knapsack problem using an approximation algorithm. We also discussed the time complexity of different algorithm. Backtracking is not discussed here due to it's inherent sequential property. The validity of the proposed algorithm is demonstrated on a worked out example and it shows the superiority of the proposed algorithm.**

Keywords: **Parallel algorithm, approximation algorithm, Dynamic programming, polynomial time.**

I. INTRODUCTION

The knapsack problem is one of the most powerful problems in combinatorial optimization, and is known to be NP-Hard. Using Dynamic programming; the problem can be solved serially in pseudo-polynomial time. For many combinatorial problems, exhaustive search is unfeasible. When using Dynamic programming (DP), the computational cost is so high and the memory requirement is also high. Parallel computers greatly reduce the computation time for solving large scale problem since there are a number of parallel operations that occur in the evaluation of the DP recursive formulae.

The parallel implementations of knapsack problem in hybercube and ring topology are based on precedence graph. Here, we have proposed another version known as summing the subset from original input set and check to see whether it satisfies the objective function.

This paper is categorized as follows: Section [1] gives the mathematical formulation of knapsack

Problem. Section [2] presents the mathematical models and problem descriptions. Section [3] elaborates another version of knapsack problem i.e. subset sum principle. Section [4] deals with simulation study and Section [5] gives overview of parallel algorithms for knapsack problem based on different interconnect architecture. Section [6] discusses the issues in parallel implementation of knapsack problem.

[1] Department of Computer Science and Engineering, Thiagarajar College of engineering, Madurai-15
Email: spmcse@tce.edu, shalinie@tce.edu

II. APPLICATION OF DYNAMIC PROGRAMMING TO KNAPSACK PROBLEM

A. Mathematical formulation

An instance KNAP (A,c) of the Knapsack problem consists of a set A of m objects, and an integer Knapsack capacity denoted by c. Each object a_i of A has a positive weigh w_i. For each object of type a_i that is placed in the knapsack, a positive profit p_i is earned. The objects placed in the knapsack will be called as items. The problem objective is to fill the knapsack in such a way that the total profit is maximized.

The problem can be formulated as:

Find integers $x_i \geq 0$ (for $0 \leq i \leq m-1$) to

Maximize $\sum_{i=0}^{m-1} p_i x_i$

Subject to the constraint $\sum_{i=0}^{m-1} x_i w_i \leq c$

Define the following function

$$f(k,g) = \max\left\{\sum_{i=0}^{k} p_i\, x_i \,|\, \sum_{i=0}^{k} x_i w_i \leq g, x_i \text{ ppositive integers, i=0,……,k }\right\}$$

Given a knapsack problem instances KNAP(A,c),the maximal profit is equal to f (m-1,c).The Dynamic programming iterations for integers $0 \leq i < m$ and $0 \leq g \leq c$ is defined as follows:

$$f(k,g) = \text{Max}\ \{f(k-1,g),\ p_k + f(k,g-w_k)\}$$

$$\text{With} \begin{cases} f(-1,g) = 0 \\ f(k,0) = 0 \\ f(k,y) = -\infty, y < 0 \end{cases}$$

There are two main properties which are very useful to reduce the search space when solving the Knapsack problem: dominance and periodicity. For both these properties we have to focus on the ratio p_i/w_i for each object. For the dominance the main idea is to find objects that may never occur on the optimal solutions. For example, if there are three objects a_1,a_2,a_3 such that $w_1+w_2=w_3$ and $p1+p2 \geq p3$, there exists an optimal solution without objects a_3. There are several, including some very recent, work elaborated dominance

relations [4, 9].The period icity states that beyond a certain capacity only the objects a_2 with the bigger ratio ($p2/w2=max_1$ p1/w1) contributes to the optimal solution (ie) g for sufficiently large if the function f (k,g) (defined in section 2.1).

We have, f (m, g) =f (m, g-w_j) +p_j.

These two properties will reduce the search space significantly.

III. KNAPSACK PROBLEM USING THE SUBSET SUM PRINCIPLE

In this section, we proposed an approximation algorithm for the knapsack problem. Approximation algorithm is a polynomial time algorithm that when the given input I, outputs an element of FS (I), where FS (I) is the feasible solution for an instance I. Here, we relate the solution for knapsack problem with the subset sum problem, where the objective is to select among subsets that give a maximum sum of objects

A. Simplified knapsack problem

In this approach, the profit for each object is the same as its size. Thus the input is an integer C and a sequence (s1,s2,…sn).The algorithm can be extended to the general Knapsack problem by starting with the list of objects in order by" profit density"; that is sorting so that p1/x1≥p2/x2 ≥…..pn/xn. The theorems about the closeness of the approximations can be easily carried over to the general knapsack problem too.

Input: Integer C, the capacity and s1, s2… sn are a sequence of objects.
Output: OUT, a subset of N= {0, 1…N}, an object to hold OUT is passed in and the algorithm that fits its fields.Also, the approximation returns Msum, the sum of s_i for i<OUT.
sKnap$_k$ (C, S, OUT)
int msum,sum;
index T-new Index ;
int j;
OUT=Ø; msum=0;
for each subset $T \subseteq N$ *with at atmost k elements;*
sum= $\sum_{i \in T} s_i$;
if (sum $\leq C$)
//Consider remaining objects
for each j not in T.
if (sum+$s_j \leq s_j$)
sum+= s_j;
T=T U {j};
//check whether obtained best optimal solution
if (msum <sum)
msum=sum;
copy fields of T into input.
//continue with next subset of atmost k indexes.
Return msum;

Fig. 1. Simplified Knapsack Algorithm

Table 1: Knapsack Problem using Simplified Algorithm

	Subset of size k	Objects added by inner for loop	Sum
K=0	Ø	{54,45,1}	100
Objects taken : {54,45,1}		Max. sum=100	
K=1	{54}	{45,1}	100
	{45}	{54,1}	100
	{43}	{54,1]	98
	{29}	{54,23,1}	107
	{23}	{54,29,1}	107
	{21}	{54,29,1}	107
	{14}	{54,29,1}	98
	{1}	{54,45,}	100
Objects taken:{29,54,23,1}		Max. sum=107	
K=2	{54,45}	{1}	100
	{43,29}	{23,14,1}	110
	{23,21}	{54,1}	99
	{14,1}	{54,45}	104
Objects taken:{43,29,23,14,1}		Max. sum=110	

There is a simple strategy where M is the maximum profit of any object in the input. First go through the sequence of objects and put it in the knapsack that it fits. Let V be the sum of the values of the objects whose profit is M.

We present a sequence of polynomially bounded algorithm sKnap$_k$ for which the ratio of the optimal solution to the algorithm's output is 1/1+k.However,the amount of work done by sKnap$_k$ is in $o(kn^{k+1})$.Thus the closer approximation, the higher the degree of the polynomial describing the time bound. Using the main idea in this algorithm with an additional trick, it is possible to get a sequence of algorithms that achieve equally good results but run in time $o(n+k^2 n)$.

For k ≥0, the algorithm sKnap$_k$ considers each subset T with at most k elements. If $\sum_{i \in T} s_i \leq C$, it goes through the remaining objects $\{s_i \mid i \notin T\}$,and greedily adds objects to the Knapsack as long as they still fit. The output is the set so obtained that gives the largest sum.

Theorem: for k>0, algorithm sknap$_k$ does O(kn^{k+1}) operations;skanp$_0$ does O(n).hence sknap$_k$ ЄP for k≥0.

Proof: There are $\binom{n}{j}$ subsets containing j elements of N, so the outer loop is executed $\sum_{j=0}^{k}\binom{n}{j}$ times. Since $\binom{n}{j} \leq n^j$ and $\binom{n}{0}$=1, $\sum_{j=0}^{k}\binom{n}{j} \leq kn^k+1$.the amount of work done in one pass through the loop is in O (n) so for all passes it is in O (kn^{k+1}+n).

Table 2: Comparison of Time Complexity for Different Sequential Algorithm

Algorithm	Time complexity
Greedy algorithm	O(n log n)
Dynamic programming	O(n+W)

IV. CASE STUDY

The input for the problem C=110 and a sequences of objects that are arranged in nondecreasing order (54, 45, 43, 29, 23, 21, 14, 1). The optimal solution includes (43, 29, 23, 14, and 1) and fills the knapsack completely. This solution found by $sKnap_2$. Table 1 shows the simulation of the knapsack problem using proposed algorithm.

V. OVERVIEW OF SOLUTIONS FOR KNAPSACK IN PARALLEL

Several algorithms have been proposed in the recent years and are also implemented in parallel to solve knapsack problem. They are responsible for certain architectures which are based on interconnection networks.

A. Parallel algorithm for hypercube

There are two well-known hypercube algorithms to solve the knapsack problem. In [5], the idea is to solve sub problems of the original problem in parallel using dynamic programming and then to combine the results. For a problem with m types of objects and a knapsack capacity c on p processors, the time is O (mc/p+c2).This algorithms specially designed for hypercube.

Lin and Storer [6] proposed an efficient algorithm that solves a knapsack on a PRAM within time O (mc/p) (where p≤c).The Obvious implementations on hypercube has time complexity O (mc/p log p) and efficiency O (1/log p). One can easily verify that the LIN and Storer's algorithm can be seen as execution of a precedence graph on a PRAM. Given p (≤c) processors, the graph is partitioned into sets of [c/p] columns, each one being allocated to a different processor.

Table 2.1: Complexity Table for Parallel Knapsack Problem in Hypercube

Algorithm	Knapsack	Conditions
Goldman and Trystram [16]	$O\left(mc / p * w_{\max} / w_{\min}\right)$	$p < \frac{c}{\log w_{\max}}$

B. Parallel algorithm for systolic arrays

In [2], an algorithm was proposed which shows that the best possible time to solve the knapsack problem on a linear array with limited memory. In [3], the authors used a one-dimensional systolic array for it.

Table 2.2: Complexity Table for Parallel Knapsack on Ring Network

Algorithm	Knapsack	conditions
Chen et al [8]	$O\left(mc / p + m\right)$	P ≤m
Andonov and Rajopadhye [6]	$O\left(mc / p\lceil w_{\max} + w_{\min} / \alpha \rceil\right)$	P ≤m

The best known serial algorithm to date in terms of running time continues to be the two-list Horowitz and Sahni [1] algorithm, which solves the Knapsack in O ($\sqrt{2}^n$) in time and space. The Lou and Cheng suggested more recently another two-list parallel algorithm, also based on a shared memory SIMD machine with the same amount of $O(2^{n/8})$ processors. Their algorithm implies a bound on $O(2^{n/4})$ for the memory and $O((n/8)^2)$ in time.

C. Parallel algorithm for knapsack using dynamic programming:

Solving knapsack using DP is by parallel serial monadic algorithm. The algorithm can be categorized by two facts:

(1) The sub problem at all levels depends only on the results of immediately preceding results and (2) The functional equation contains a recursive term. The parallelization can be done: parallelizing the state variables, parallelizing the decision variables.

Parallel states algorithm : This make use of parallel processing arrived out in the state level (ie.objects level).Each slave process initially receives from the master a subset of quantized states at stage k.Every single permitted quantized decision has to be checked for every quantized state.

Parallel Decision algorithm: This algorithm is carried out in two parts at each stage k. The first part, each slave processor receives from the master only a subset of the admissible decisions and, subsequently performs the optimization over all the quantized states at stage k, using this subset of decisions.Thus, at each slave processor obtains a local maximum that it is sent to the master. In the second part, the master, once all the local optima have been gathered, computes the actual global optimum.

VI. EXPERIMENTAL RESULTS

The proposed algorithm was executed in Debian Linux Clusters using MPI libraries. The Linux cluster environment consists of 25 HP P-IV 3 GHz processors, connected through 100 Mbps Ethernet technologies. Fig. 2 and 3 shows the performance and runtime of parallelized knapsack using Dynamic programming

VII. ISSUES IN PARALLEL DP (PDP)

The sub-problems in dynamic programming are overlapping by nature. If each overlapping sub-problem is required to be solved only once, the overlap creates a data dependency. In structured PDP, if there is negligible communication delay between sub problems within the same PE, the sub problems in a structured dynamic programming problem can be grouped into blocks to increase computational granularity.

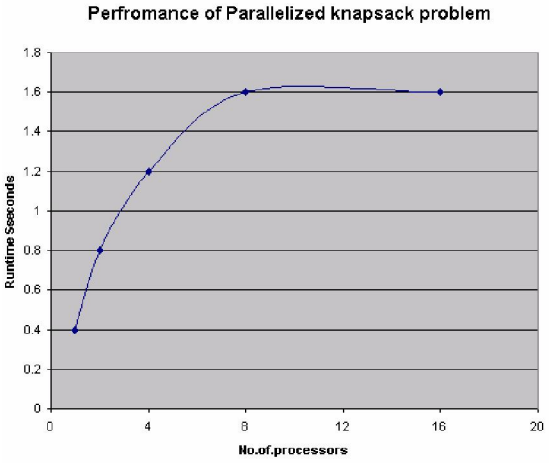

Fig. 2. Performance of Parallelized Knapsack using DP

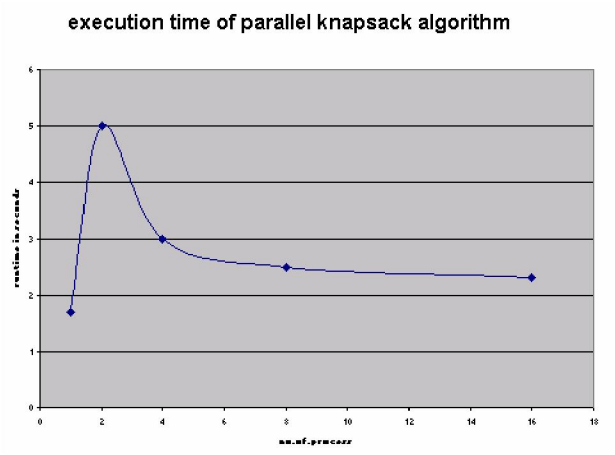

Fig. 3. Execution Time of Parallelized Knapsack using DP

Using the Knapsack problem as an example, given the table of sub problems to solve, both data and computation can be distributed to multiple PEs. In [8] unstructured PDP, the key to parallelizing the unstructured dynamic programming problem is the assignment of subject, Directed Acyclic Graph (DAG) nodes and their associated sub-problems to parallel PEs. The technology of mapping problem can be parallelized by solving independent sub trees of the subject DAG. Given the various matching primitive DAGs at a node, a "ghost zone" the edge of the sub trees can be established, where sub-problems are buffered such that they are only communicated once. However, exploring this communication vs. computation trade-off is a non-trivial task in a net list data structure.

VIII. CONCLUSION

In this paper, we have presented an approximation algorithm for solving knapsack problem sequentially and also presented certain issues and factors to be considered during parallel implementation strategies. We have also presented different parallel algorithms based on interconnection architecture for solving knapsack in parallel. In future, we planned to use some machine learning techniques to increase the performance of combinatorial problem like Traveling salesman problem and Knapsack problem.

ACKNOWLEDGEMENT

The authors acknowledge with thanks the Management and Principal of Thiagarajar College of Engineering for their encouragement and financial assistance.

REFERRENCES

[1] E.Horowitz, S.Sahini, Computing partitions for the knapsack problem, Journal of ACM 21(2) (1974)277-292.

[2] R.Andonov, P.Quinton, Efficient linear systolic array for knapsack problem, CONPAR'92. Lecture notes on Computer Science, Springer, vol.634, bErlin, 1992, pp.247-258

[3] V.Aleksandrov, S.Fidanova, On the expected execution time for a class of non uniform recurrence equations mapped onto 1darray, Parallel algorithm, Appl.1 (1994)303-314.

[4] Gilles Brassard, Paul Bratley, Fundamentals of Algorithmics, PHI, 2003.

[5] j.Lee,E.Shragowitz, S.Sahni,A Hybercube algorithm for the 0/1 knapsack problem,J.Parallel and Distributed Comput. 5(4)(1988) 438-456.

[6] J.Lin,J.Storer,Processor efficient hybercube algorithm for the knapsackproblem,Journal of Parallel Distrib. Comput.13 (3) (1991)332-337.

[7] A.Goldman,D.Trystram,An efficient parallel algorithm for solving the knapsack problem on hypercubes, J.Parallel and Distributed Computing, 64(2004) 1213-1222.

[8] Sebastian Dormido Canto,Angel P.deMadrid&Sebastian Dormido Bencomo,Parallel Dynamic Programming on Clusters of Workstations ,IEEE Transactions On Parallel and Distributed Systems, vol.16, no.9, September-2005.

[9] D.ElBaz, M.Elkihel, Load balancing methods&parallel dynamic programming algorithm using dominance technique applied to 0/1 knapsack problem,J.Parallel and Distributed Computing,65(2005)74-84.

[10] Gilles Brassard, PaulBratley, Fundamentals of Algorithmics, Prentice Hall of India Pvt Ltd, 2003.

[11] Michael J.Quinn, Parallel programming in C with MPI &OpenMP, TMH, 2003.