Solving Bi-knapsack Problem Using Tiling Approach for Dynamic Programming

Benamar Sidi Boulenouar

LAMIH-ROI, UMR 8530,
ISTV - Le mont Houy -BP 311- 59304 Valenciennes Cedex- France
sidi@univ-valenciennes

Abstract. In this paper we present an efficient parallelization of the dynamic programming applied to bi-knapsack problem, in distributed memory machines(MMD). Our approach develops the tiling technique in order to control the grain parallelism and find the optimal granularity. Our proposed approach has been intensively validated on the Intel Paragon and IBM/SP2 using NX and MPI libraries. The experimental results show a linear acceleration, which enables to solve huge instances of the hardest known 0/1 bi-knapsack problems in a very reasonable time.

1 Introduction

For the knapsack problem, many parallel dynamic programming (DP) implementations are known [1,4,6,7]. This is not the case for bi-knapsack problem (BKP).

In this study, we discuss how to find optimal parallelism granularity on a distributed memory machine (DMM). The proposed approach is based on the tiling technique, which is a common method to improve the performance of loop programs on DMM. These techniques are developed very actively these last years around the automatic parallelization of the nest of loop [3,5,8].

Tiling consists in partitioning the iteration space into blocks called tile; each tile is executed by a single processor in an atomic way. Finding the optimal tiling parameters of the tile (shape and size) enables to minimize the execution time by reducing the extra cost of communications. The tiling technique is today well developed and widely used in case of Uniform Recurrent Equation(URE).

However, this technique cannot be applied directly to DP recurrences for the BKP because of the irregular nature of their dependencies, which vary with any problem instance.

We show here how the tiling technique can be extended and how it can be successively applied to a DMM DP implementation for the bi-knapsack problem.

2 Dynamic Programming for the Bi-knapsack

The bi-knapsack problem (BKP) is a classic NP-*hard* problem and can be formulated as follows: we are given a bi-knapsack of capacity c_1 and c_2, into which

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 560–565, 2001.

we may put m types of objects. Each object of type i has a *profit*, p_i, and two *weight*, (w_i, u_i), (w_i, u_i, p_i, m, c_1 and c_2 are all positive integers). Determine, for $i = 1 \ldots m$, the number x_i, of i-th type objects to be chosen so as to maximize the total profit without exceeding the capacity, i.e.,

$$\max \left\{ \sum_{i=1}^{m} p_i x_i \ : \ \sum_{i=1}^{m} w_i x_i \leq c_1, \ \sum_{i=1}^{m} u_i x_i \leq c_2 \ x_i \in \{0,1\}, \ i = 1, 2, \ldots, m \right\} \tag{1}$$

The dynamic programming method solves (BKP) following recurrences: let $\mathcal{D} = \{(i,j,k) : 0 \leq i \leq m \ ; \ 0 \leq j \leq c_1 \ ; \ 0 \leq k \leq c_2\}$ be the recurrence domain of iterations (2) ; for every couple (i,j,k) of $\mathcal{D}$, calculate.

$$f(i,j,k) = \begin{cases} f(i-1,j,k) & \text{if } i > 0 \text{ and } (j < w_k \ or \ k < u_k) \\ \max(f(i-1,j,k), p_i + f(i-1, j-w_i, k-u_i)) & \text{if } i > 0 \text{ and } (j \geq w_i \ and \ k \geq u_i) \end{cases} \tag{2}$$

The particularity of these dependencies that the values w_i and u_i vary in large interval with any index i and any instance of the problem. Figure 1 shows the dynamic characters of dependencies : the diagonal dependencies varies according to the values of w_i and u_i.

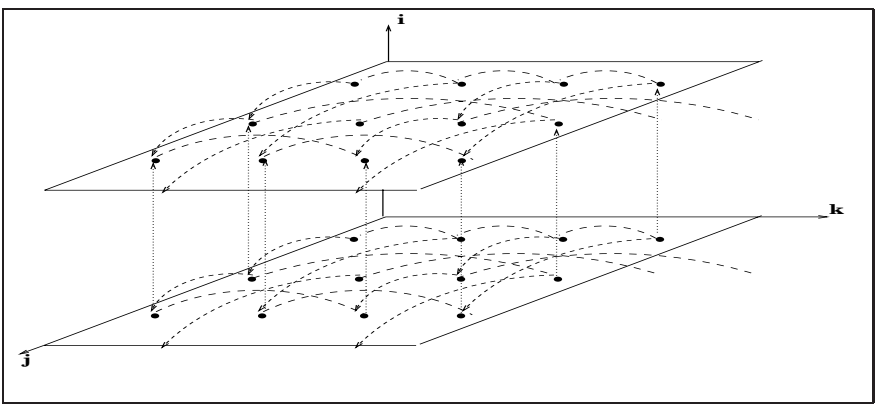

Fig. 1. Bi-knapsack dynamic dependencies

3 Tiling and Processors Allocation

The recurrences (2) are represented by three-dimensional nested loops. The associated iterations space is a cube of $(m \times c_1 \times c_2)$ size, in which the dependencies vectors are (1,0,0) and (1,w_i,u_i).

The particularity of these dependencies is that the values w_i and u_i vary in large interval with any index i and any instance of the problem. For these reasons, it is not possible to guaranty the local references and constant communications volume, when using tiling techniques on grid architecture.

In order to avoid that such dependencies involve no constant volume communications, the projection of the tile graph is made vertically on a p processors ring. In this case, by choosing an appropriate projection we can guaranty that the runtime dependencies require only local references.

The iteration space figure 2(a) is partitioned into rectangular parallelepipeds of size $x_1 \times x_2 \times x_3$. To compute each tile, the processor receive a message of $x_2 \times x_3$ elements from its left neighbor, and sends to its right neighbor message of the same size $x_2 \times x_3$. In this case, the important parameters to determine the optimal tile size are the values of x_1 and $x_2 \times x_3$. In order to simplify our model, we put $h = x_2 \times x_3$ and $W = m \times c_1 \times c_2$. Because the dependencies are orthogonal, we can easily demonstrate that 3D iteration space can be tiled which 2D tile [9]. By this transformation, we obtain a 2D iteration space tiled with rectangular surfaces as it is show in figure 2(b).

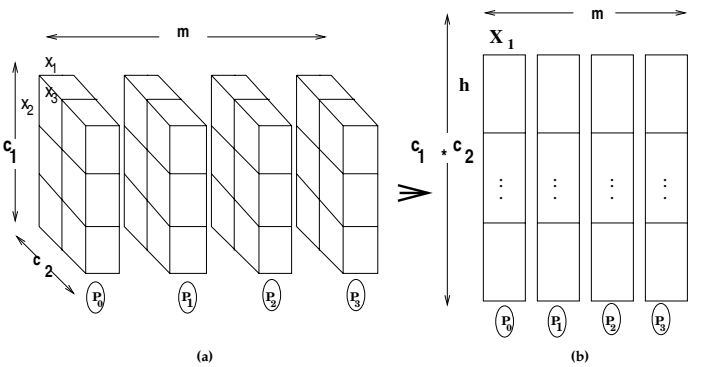

Fig. 2. Ring: transformation of a tile graph 3D (a) into a graph 2D(b)

Hence, we can apply the Andonov and Rajopadhye results [2] to determine the optimal tile. The execution time (according to x_1 and h) is given by the function:

$$T(x_1, h) = \frac{2W\beta}{px_1h} + (p-1)\alpha x_1 h + (p-1)\tau h + (p-1)\beta + \frac{W\alpha}{p} \tag{3}$$

The optimal tile size is given by:

$$(h^*, x_1^*) = \begin{cases} \left[\sqrt{\frac{2pc_1c_2\beta}{(p-1)(m\alpha+p\tau)}}, \frac{m}{p}\right] & \text{if } \lambda_2 > 0 \text{ (no cyclic solution)} \\ \left[1, \sqrt{\frac{2W\beta}{(p-1)p\alpha}}\right] & \text{else (cyclic solution)} \end{cases} \tag{4}$$

with $\lambda_2 = 2pc_1c_2\beta - (p-1)m\alpha$

The constants β and τ respectively correspond to the time to establishing communication and to the transfer of data. α correspond to the execution time of

a single instruction. In our case, λ_2 is always positive because $c_1 \times c_2 \gg m$ (BKP problem), $\beta > \alpha$ (Distributed Memory Machine) and $p \approx p - 1$). Therefore, in the rest of this paper, we consider the *no cyclic solution*.

4 Sensitivity of the Model

The same approach can be used in the case of dynamic dependencies of bi-knapsack type, only if the single instance computing is constant. This assumption is at the base of the previous results, but it is impossible to be considered here. The dynamic dependency (of w_i and u_i length) prevents the reference locality, the access time to data varies because the cache and pagination techniques. In this part, we propose a strategy assuring the stability of the result when α is not a constant. In order to simplify our study, we take a simple knapsack case with one constraint ($c_2 = 0$ and all $u_i = 0$).

The table 1 shows the computing times of a single instance obtained for different coefficient instances w_i generated randomly in a fixed interval. We can note that, the time for computing an instance increases with the coefficients value w_i, which confirms the memory effect.

Table 1. variation of α according to the values of coefficients w_i

w_i	$[1, 10]$	$[1, 5.10^2]$	$[10^3, 15.10^2]$	$[5.10^3, 10^4]$	$[10^4, 3.10^4]$
α(average)	$3\mu s$	$6\mu s$	$13\mu s$	$15\mu s$	$23\mu s$

Figure 3(c) shows the experimental curves which correspond to the total executing time, by varying the height of the tile (parameter h) for the w_i and coefficient values taken in the said interval. The size of the problem capacity ($c_1 = 20700$) and number of objects(m=1000) are kept fixed. We can notice that in a small interval the optimal value of h does not vary much, in spite of the important variation of the different values of α.

This observation led to consider the minimal value of α (obtained by fixing the w_i to 1) to calculate the optimal tile size. This strategy guarantees that the optimum calculated this way always stands on the right (hand) side of the real minimum. On this side, the time function has a weak slope, and the points in the neighborhood of the real minimum give a good approximation of the minimum value.

The following table describes a validation of this strategy. Each line represents a fixed instances of problem where the time value α is known (1st column). These values are taken into account to calculate h^* (using eq(4)) . The third column of the table shows the total executing time obtained for tile of h^* size. The fourth and the fifth columns show approximate values (h_{app}, T_{app}) obtained using the minimal value of α. $\triangle$ means $T_{opt} - T_{app}$. We can see in the last column, that the relative error does not exceed 3%.

α	$h^*(\alpha)$	T_{opt}(sec)	$h_{app}(\alpha = 1\mu s)$	T_{app}(sec)	$\triangle$	$\frac{\triangle}{T_{opt}}$
$1\mu s$	99	10.50	99	10.50	0.00	0.00
$3\mu s$	57	30.83	99	31.29	0.45	0.01
$6\mu s$	40	61.16	99	62.48	1.31	0.02
$13\mu s$	27	131.70	99	135.25	3.54	0.02
$15\mu s$	25	151.82	99	156.04	4.21	0.02
$23\mu s$	20	232.24	99	239.21	6.96	0.03

5 Experimental Results

We now describe some experimental results of our algorithm. The experiments were run on Intel Paragon at IRISA and IBM/SP2 at CINES, for bi-knapsack problems chosen randomly. The main characteristics of these machines are given in table 2.

Table 2. Technical characteristics of the Intel Paragon and the IBM SP2

	Intel Paragon	IBM SP2
number of processors	50	207
data cache per node	128KB	16KB
memory per processor	56MB	256MB
max peak processor speed	100Mflops	500Mflops
point to point bandwith	175MB/sec	35MB/sec
τ	$0.0015\mu s$	$0.0022\mu s$
β	$40\mu s$	$45.0\mu s$

Figure 3(a)(b)(d)(e) shows that the experimental curve is close to the theoretical curve; the optimum (h optimal) coincides with the estimate value. The experimental evaluation of the speed−up 3(f) allowed us to notice a linear speed-up, this being due to the memory effect. In fact, the size of the memory used by each processor is $(c_1 \times c_2 \times m/p)$, it decreases by increasing the number of processor (p), which in turn reduces the time access memory. If we choose the IBM/SP2, for big problems size, the choice of the optimal tiling is very important (Figure 3(e)).

6 Conclusion

This paper presents an efficient parallelization of the dynamic programming applied to bi-knapsack problem. The approach proposed develops the optimal tiling technique in order to have a best *computing/communication* ratio, for a particular case of dynamic dependencies. We analyze the sensitivity of the result for non-constant cycle times and we propose a strategy for this problem. Computational tests indicate that the strategy works well, giving linear speedup over

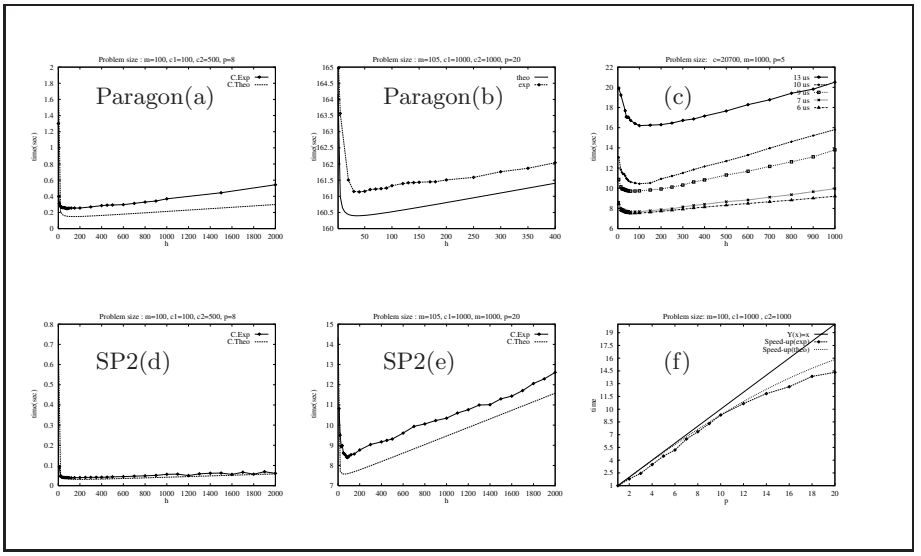

Fig. 3. Optimal tile size validation and acceleration obtained

a range of processor numbers and tracking theoretical performance predictions closely. According to our knowledge, it is the first parallel algorithm for the bi-knapsack problem.

References

1. R. Andonov and S. Rajopadhye. Knapsack on VLSI : from Algorithm to Optimal Circuit. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):545–562, 1997. 560
2. R. Andonov and S. Rajopadhye. Optimal Orthogonal Tiling of 2-D Iterations. *Journal of Parallel and Distributed Computing*, 45:159–165, September 1997. 562
3. P. Boulet, A. Darte, T. Risset, and Y. Robert. (Pen)-Ultimate Tiling? *INTEGRATION, the VLSI journal*, 17:33–51, Nov 1994. 560
4. G. H. Chen, M. S. Chern, and J. H. Jang. Pipeline Architectures for Dynamic Programming Algorithms. *Parallel Computing*, 13:111–117, 1990. 560
5. F. Irigoin and R. Triolet. Supernode Partitioning. In *15th ACM Symposium on Principles of Programming Languages*, pages 319–328. ACM, Jan 1988. 560
6. J. Lee, E. Shragowitz, and S. Sahni. A Hypercube Algorithm for the 0/1 Knapsack Problems. *J. of Parallel and Distributed Computing*, 5:438–456, 1988. 560
7. J. Lin and J. A. Storer. Processor-Efficient Hypercube Algorithm for the Knapsack Problem. *J. of Parallel and Distributed Computing*, 13:332–337, 1991. 560
8. J. Ramanujam and P. Sadayappan. Tiling Multidimensional Iteration Spaces for Non Shared-Memory Machines. In *Supercomputing 91*, pages 111–120, 1991. 560
9. B. Sidi Boulenouar, H. Bourzoufi, and R. Andonov. Tiling and processors allocation for three dimensional iteration space. *In International Conference on Higt Performance Computing (HiPC), ACM/ IEEE*, pages 125–129, India, 1999. 562