Solving a 2D Knapsack Problem on an Associative Computer Augmented with a Linear Network

Darrell R. Ulm and Johnnie W. Baker
Department of Mathematics and Computer Science
Kent State University Kent, OH 44242
dulm@mcs.kent.edu jbaker@mcs.kent.edu

Abstract

This paper describes a parallelization of the sequential dynamic programming method for solving a 2D knapsack problem where multiples of n rectangular objects are optimally packed into a knapsack of size $L \times W$ and are only obtainable with guillotine-type (side to side) cuts. The parallel algorithm is described and analyzed for the associative model. The associative model (ASC) for parallel computation supports a generalized version of an associative style of computing that has been used since the introduction of associative SIMD computers in the early 1970's. In particular, this model supports data parallelism, constant time maximum and minimum operations, one or more instruction streams (ISs) which are sent to an equal number of partition sets of processors, and assignment of tasks to the ISs using control parallelism. This algorithm runs in $O(W(n+L+W))$ time using $O(L)$ processors, where $L \geq W$ for a 2D knapsack problem with a capacity of $L \times W$. This result is cost optimal with respect to the best sequential implementation. Moreover, an efficient ASC algorithm for this well-known problem should give insight to how the associative model compares to other parallel models.

Keywords : 2D knapsack problem, associative computing, ASC parallel algorithms, optimization.

1 Introduction

A knapsack problem requires finding a subset from a set of objects such that the sum of the object profits is maximized while not exceeding the knapsack size or violating any other constraints. Such problems appear in computer science and operations research, e.g. in cutting stock applications. The 2D problem is related to the well studied 0-1 knapsack problem which has been solved efficiently with linear systolic arrays[1].

Several 2D knapsack algorithms considering many problem constraints are known[2][3]. The problem examined herein is in the class NP, but it can be solved sequentially in $O(LW(n+L+W))$ [2], where n is the number of objects, and L and W are the dimensions of the knapsack. This running time is called pseudo-polynomial because in terms of the input size, the knapsack capacity is encoded in only $\log_2(L) + \log_2(W)$ bits. Past work on this problem has addressed solutions for more complex models using the hypercube or the mesh with multiple buses (MMB) networks[4][5].

2 The 2D knapsack problem

The 2D knapsack problem requires filling an area of dimensions (L, W) with n rectangles of size (l_i, w_i) where $i = 1, 2, \ldots, n$. The profits are non-negative values, $\Pi_1, \Pi_2, \ldots, \Pi_n$, associated with each rectangle. With these parameters, the maximum profit of $\Pi_1 z_1 + \Pi_2 z_2 + \ldots + \Pi_n z_n$ is to be computed where z_i is a non-negative integer such that the knapsack is partitioned into z_i multiples of rectangle i, having the size (l_i, w_i) [2]. This cutting problem allows only recursive side-to-side or *guillotine* cuts of the knapsack. Thus all cuts must be made perpendicular from one edge of a rectangle to the other. Objects may have a fixed orientation or be allowed to rotate 90^o. An additional n objects of dimensions (w_i, l_i) with profit Π_i, are added when rotations are allowed. Algorithms to solve this type of problem include tree searching or dynamic programming which this work uses[1][4][5][6].

The knapsack function $F(x,y)$, derived from dynamic programming techniques, is computed such that for a location (x,y), $F(x,y)$ is the largest profit obtainable from the rectangle created by the X and Y axes and the point (x,y). It satisfies the following inequalities corresponding to the guillotine cutting restrictions: $0 \leq x \leq L$, $0 \leq y \leq W$, $F(x,y) \geq 0$, $F(x_1+x_2,y) \geq F(x_1,y)+F(x_2,y)$, $F(x,y_1+y_2) \geq F(x,y_1)+F(x,y_2)$, $F(l_i,w_i) \geq \Pi_i \quad (i=1,\ldots,n)$[2].

3 The multiple IS associative computing model (ASC)

This section describes the associative model of computation presented in the IEEE Computer article "ASC: An Associative Computing paradigm," which is based on work done at Kent State University[7]. A more complete reference can be found in [7] and in *this proceedings* in a paper entitled "Virtual Parallelism by Self Simulation of the Multiple Instruction Stream Associative Model."[8]. In the most basic terms, the *associative model (ASC)* has an large number of processing elements (PEs) and one or more instruction streams (ISs) that broadcast their commands to partition sets in a dynamic partition of PEs. The number of ISs is normally expected to be small in comparison to the number of PEs. The multiple ISs supported by the ASC model allows greater efficiency, flexibility, and reconfigurability than is possible with only one IS. An ASC machine with j ISs and n PEs will be written as $ASC(n,j)$. An IS sends its instructions to the PEs over a bus, and the ISs should be considered to be local to the PEs in terms of hardware. Each PE has a local memory and ASC supports the associative processing concept which is to locate objects in the local memory of the PEs by content instead of location. This is accomplished by having each active PE search a specified field in its local memory for a given data item in parallel. Each PE is capable of performing local arithmetic and logical operations and the other usual functions of a sequential processor, but each PE is assumed to be reasonably basic so that the maximum number of PEs can be integrated on a chip. Additionally, the associative computer is assumed to be equipped with an interconnection network between the PEs. While there is no restriction on the network allowed for the ASC model, some of the most obvious choices are the $1D$, $2D$, and $3D$ mesh because of their ease to implement in VLSI and their expandability[9][7]. A diagram of ASC is shown in Figure 1.

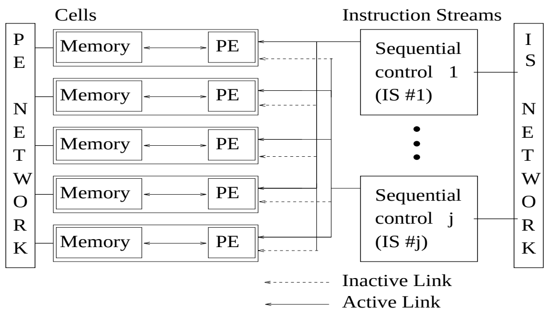

Figure 1: The ASC Model

4 The parallel algorithm

A 2D knapsack algorithm for the single IS model with a 1D mesh (linear) network is presented in this section. This simple network is needed to propagate profit values down the array as the algorithm progresses. The 2D knapsack example shows how an algorithm with complex parallel dependencies can be mapped efficiently to the ASC model, and solved the problem without the use of complex networks such as the hypercube. Also, this problem and other problems implemented in ASC provides greater insight to how well the ASC model compares to other parallel models.

The parallelization given in Figure 2 is accomplished by first noting that any at location in the array, $F_{a,b}$ is dependent on other $F_{i,j}$ entries at lower subscripted positions. In fact, the only values needed are those in row a and column j with $j \leq b$ and those in column b and row i with $i \leq a$. This data dependency prevents a row or column of F values from being computed in parallel, but a diagonal set of F values can be calculated concurrently. Since the number of diagonals traversed is $O(L+W)$, $O(\max\{L,W\})$ iterations are needed to scan every diagonal.

Let $P_{i,j}$ be a processor located inside the diagonal region, $diag/2 \leq i+j \leq diag$, where $diag$ is an integer denoting the current diagonal. The F value at this processor contributes to calculating $F_{diag-j,j}$ and $F_{i,diag-i}$. The parallel variables $V_{i,j}$ and $H_{i,j}$ contain shifted $F_{i,j}$ values that are added to the current location for vertical and horizontal directions respectively. In each iteration these values shift to the next location, and a new set of values is appended to the shifting parallel variables $V_{i,j}$ and $H_{i,j}$. These shifting operations use $1D$ mesh connections such that one horizontal shift for H is completed in unit time where there are $O(W)$ horizontal shifts required. Shifting V values inside each PE takes $O(W)$ time as does scanning for the largest F value in each PE. After the sums are collected in participating processors, the maximum of all sums is computed from variables $sumV_{i,j}$ and $sumH_{i,j}$ for each row and column, and the result is stored in the array location that row or column's diagonal. This is done for vertical sums using the constant time ASC maximum function $O(W)$ times, and in a second step for horizontal sums, the PEs scanning through F values in $O(W)$ iterations. Thus for each column, the maximum of $sumV_{diag-j,j}$ through $sumV_{(diag-j)/2,j}$ is computed, storing the result in location $F_{diag-j,j}$ on the current diagonal. Likewise, a for each row the maximum of $sumH_{i,diag-i}$ through $sumH_{i,(diag-i)/2}$ is computed, saving the result in $F_{i,diag-i}$.

1 if $(L < W)$
  swap(L, W)
  for $k \leftarrow 1$ to n swap(l_i, w_i)
2 for all processors P_i do in parallel
2.1 for $j \leftarrow 0$ to W do $F_{i,j} \leftarrow 0$, $V_{i,j} \leftarrow 0$, $H_{i,j} \leftarrow 0$
2.2 for $k \leftarrow 1$ to n
  for $j \leftarrow 0$ to W do
    if $(i \geq l_k \ and \ j \geq w_k)$ $F_{i,j} \leftarrow max\{F_{i,j}, \Pi_k\}$
3 $diag \leftarrow 2$
4 while $(diag \leq L+W)$ do
4.1 for $j \leftarrow 0$ to W do
    if $(diag \leq i+j \ and \ i > 0 \ and \ j > 0)$ $V_{i,j} \leftarrow V_{i-1,j}$, $H_{i,j} \leftarrow H_{i,j-1}$
4.2 for $j \leftarrow 0$ to W do
    if $(i*2+j = diag)$ $V_{i,j} \leftarrow F_{i,j}$
    if $(j*2+i = diag)$ $H_{i,j} \leftarrow F_{i,j}$
4.3 for $j \leftarrow 0$ to W do
    if $(diag/2 \leq i+j \leq diag)$ $sumV_{i,j} \leftarrow F_{i,j} + V_{i,j}$, $sumH_{i,j} \leftarrow F_{i,j} + H_{i,j}$
4.4 for $j \leftarrow 0$ to W do
    $F_{diag-j,j} \leftarrow ASCMax(sumV_{diag-j,j}, sumV_{diag-j-1,j}, ..., sumV_{(diag-j)/2,j})$
4.5 $max_i \leftarrow 0$
4.6 for $j \leftarrow 1$ to W do
    if $(max_i < sumH_{i,diag-1-j})$ $max_i \leftarrow sumH_{i,diag-1-j}$
4.7 $F_{i,diag-i} \leftarrow max_i$
4.8 $diag \leftarrow diag + 1$

Figure 2: (L,1)ASC Implementation of the 2D Knapsack Algorithm

5 Algorithm analysis

Analysis of the running time of the algorithm can be found by examining the steps in Figure 2. Step 1 takes $O(n)$ time to scan through the objects while 2.1 needs $O(W)$ to initialize the F, V, and H values since each of the L PEs holds these arrays of size W. Step 2.2 computes static profit maximums wherever an object will fit. This requires scanning the L parallel arrays of size W for each of the n objects, which requires $O(nW)$ time. The outer loop in step 4 scans $O(L+W)$ diagonals. Sub-steps 4.1, through 4.6 each take $O(W)$ steps because of the looping through the parallel arrays. The maximum function for ASC in step 4.5 completes in constant time. The asymptotic time to read the input of $O(n)$ objects is no greater than the running time of the algorithm itself. Thus the total execution time of the ASC algorithm is $O(n) + O(nW) + O(W(L+W))$ which reduces to $O(W(n+L+W))$. Preliminary run-times on the Wavetracer indicate that the analytical time accurately predicts the actual runtime on a SIMD machine.

The benefit of the associative version is that the number of processors is cost-optimal in relation to the sequential dynamic programming algorithm when the dimensions of the rectangular knapsack are asymptotically the same. That is, even though the speed of the resulting program is not as great as the CRCW PRAM version (i.e. $O(n+L+W)$ with LW PEs), the cost (i.e. the number of processors times the parallel execution time) equals the running time of the sequential version.

6 Summary

In this paper an algorithm for the $2D$ knapsack problem is given for the associative model with one IS augmented with a linear array network. This algorithm has a time complexity of $O(W(n+L+W))$ and requires $O(L)$ processors, where $L \geq W$. Table 1 shows comparative results for this algorithm and previous implementations. Lastly, by adding a sufficient (and non-constant) number of instruction streams to the model, a knapsack algorithm with a better runtime should be possible. Though there is not space here to make an adequate presentation, it is conjectured that ASC(LW,j) with a mesh network can solve the problem in $O((n+L+W)max(L,W)/j)$.

Table 1: Comparative Running Times for the 2D Knapsack Algorithm

Model	Time	Processors	Cost Optimal
Sequential	$O(LW(n+L+W))$	1	[2]
CRCW PRAM	$O(n+L+W)$	$O(LW)$	YES[4]
2D Mesh	$O(Max(L,W)(n+L+W))$	$O(LW)$	NO[4]
Hypercube	$O((log_2 Max(L,W))(n+L+W))$	$O(LW)$	NO[4]
MMB	$O(n+L+W)$	$O(LW)$	YES[5]
$ASC(L,1)+1D$ Mesh	$O(W(n+L+W))$ where $L \geq W$	$O(L)$	YES

References

[1] R. Andonov V. Aleksandrov. A systolic linear array for the knapsack problem. *Parallel and Distributed Processing*, 17:285–299, 1991.

[2] P.C. Gilmore and R.E. Gomory. Multistage cutting stock problems of two or more dimensions. *Operations Research*, 13:94–120, 1965.

[3] J.C. Herz. Recursive computational procedure for two-dimensional stock cutting. *IBM J. Res. Develop.*, 16:462–469, 1967.

[4] P.Y. Wang D. Ulm. Solving a two-dimensional knapsack problem on simd computers. In *International Conference on Parallel Processing*, volume 3, pages 181–184, 1992.

[5] D. Ulm and J.W. Baker. Solving a two-dimensional knapsack problem on a mesh with multiple buses. In *International Conference on Parallel Processing*, volume 3, pages 168–171, 1995.

[6] P.Y. Wang. Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research*, 31(3):573–586, 1983.

[7] J. Potter J. Baker S. Scott A. Bansal C. Leangsuksun C. Asthagiri. Asc: An associative computing paradigm. *IEEE Computer*, pages 19–25, November 1994.

[8] D. Ulm and J.W. Baker. Virtual parallelism by self simulation of the multiple instruction stream associative computer. In *Proceedings of the International Conference on Parallel and Distributed Processing*, Sunnyvale CA, August 1996.

[9] J.L. Potter. *Associative Computing — A Programming Paradigm for Massively Parallel Computers.* Plenum Publishing, N.Y., 1992.