Solving a 2D Knapsack Problem Using a Hybrid Data-Parallel/Control Style of Computing

Darrell R. Ulm
Department of Computer Science
University of Akron
Akron, OH 44325, USA
dulm@cs.uakron.edu

Johnnie W. Baker
Department of Computer Science
Kent State University
Kent, OH 44242, USA
jbaker@cs.kent.edu

Michael C. Scherger
Department of Computer Science
Kent State University
Kent, OH 44242, USA
mscherge@cs.kent.edu

Abstract - This paper describes a parallel solution of the sequential dynamic programming method for solving a NP class, 2D knapsack (or cutting-stock) problem which is the optimal packing of multiples of n rectangular objects into a knapsack of size $L \times W$ and are only obtainable with guillotine-type (side to side) cuts. Here, we describe and analyze this problem for the associative model. Since the introduction of associative SIMD computers over a quarter of a century ago, associative computing and the data-parallel paradigm remain popular. The MASC (Multiple instruction stream ASsociative Computer) parallel model supports a generalized version of an associative style of computing. 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 ISs using control parallelism. We solve this NP class problem with a parallel algorithm that runs in O(W(n+L+W)) time using L processors, where $L \geq W$ for a 2D knapsack problem with a capacity of $L \times W$. The new multiple IS version using LW processors and max{L,M} ISs runs in O(n+L+W) given practical hardware considerations. Both of these results are cost optimal with respect to the best sequential implementation. Moreover, an efficient MASC algorithm for this well-known problem should give insight to how the associative model compares to other parallel models such as PRAM.

1 Introduction

A knapsack problem requires finding a subset from a set of objects while maximizing the sum of the object profits and 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 a variety of problem constraints are known, and these problems are frequently easier to solve by adding constraints or using approximation methods [2][3][10]. The problem examined herein is in the class NP, yet a best solution is computable 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 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]. This paper will solve the problem using the MASC model with a single IS and multiple ISs. Figure 1 shows a solution to a simple knapsack problem.

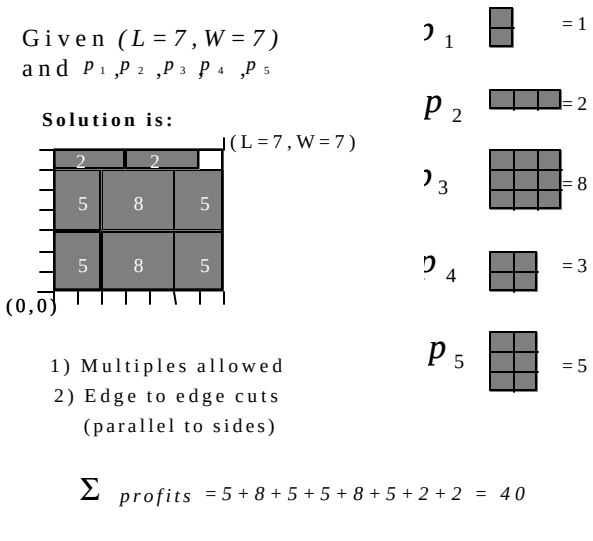

Figure 1: A possible 2D cutting stock solution

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, $p_1, p_2, \ldots, p_n$ associated with each rectangle. With these parameters, the maximum profit of $p_1 z_1 + p_2 z_2 + \ldots + p_n z_n$ 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 are made perpendicular from one edge of a rectangle to the other. Objects may have a fixed orientation or be rotated 90°. An additional *n* objects of dimensions (w_i, l_i) with profit p_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:

$$0 \le x \le L,$$
$$0 \le y \le W,$$
$$F(x,y) \ge 0,$$
$$F(x1+x2, y) \ge F(x1, y) + F(x2, y),$$
$$F(x, y1+y2) \ge F(x, y1) + F(x, y2),$$
$$F(l_i, w_i) \ge p_i, \ (i = 1,\ldots,n)$$ [2].

2.1 Sequential dynamic programming algorithm

A solution to the problem without an exponential running time is obtained using dynamic programming. Each rectangular object will be considered in turn, and a table of the best cutting profits for each location in the knapsack will be kept. Then the entire table is scanned, and lengths and widths are summed to create a configuration of rectangles that produces the optimal profit. The sequential dynamic programming algorithm is presented in Figure 2. The following relations are used to derive the sequential algorithm [2]:

$$F_0(x,y) = \max\{0, p_j \mid l_j \le x \wedge w_j \le y\}$$

$$F_k(x,y) = \max\{F_{k-1}(x,y),\ F_{k-1}(x_1, y) + F_{k-1}(x_2, y),$$
$$F_{k-1}(x, y_1) + F_{k-1}(x, y_2)\}$$

$$0 < x_1 \le x_2,\ x_1 + x_2 \le x,\ 0 < y_1 \le y_2,\ y_1 + y_2 \le y$$ [2].

In step 1 of this algorithm, all locations in the knapsack are set to zero. In step 2, each object is considered, placing the highest profit values in all locations where an object fits. Step 3 scans the 2D table from the lowest row to the highest, summing all possible combinations of vertical, horizontal cuts at each location, and retaining the two objects whose sum of profits is the largest. The code in Figure 2 shows how *F(x,y)* values are computed iteratively. Since only guillotine cuts are used, the partial rectangular solution, at location *(i,j)*, is cut into two pieces with a cut parallel to the x axis at some value of y. Because of symmetry, only x-cuts from *x=0* to i/2, and y-cuts from *y=0* to *j/2* are considered for any given *(i,j)*. The sequential algorithm requires time $O(LW(n+L+W))$, where n is the number of objects, and *L,W* are the dimensions of the knapsack[2].

```
//                                  Sequential Algorithm
for i=0 to L                        // Step 1
  for j  0 to W
    F_{i,j} = 0
for k=1 to n                        // Step 2
  for i=0 to L
    for j=0 to W
      if ( l_k ≤ i  and  w_k ≤ j and p_k > F_{i,j} )
        F_{i,j} = p_k
for i=0 to L                        // Step 3
  for j=0 to W {
    for k=0 to ⌊i/2⌋
      { sum = F_{k,j} + F_{i-k,j}
        if (sum > F_{i,j}) F_{i,j} = sum }
    for k=0 to ⌊j/2⌋
      { sum = F_{i,k} + F_{i,j-k}
        if (sum > F_{i,j}) F_{i,j} = sum }}
```

Figure 2: Sequential dynamic programming algorithm to solve 2D cutting stock

3 MASC: Multiple instruction stream ASsociative Computer

This section describes the associative model of computation presented in the IEEE Computer article "MASC: An Associative Computing Paradigm," which is based on work done at Kent State University [14]. Also, see [1][2][12][13]. The vast majority of existing platforms

already efficiently support MASC, and there is an actual language called ASC [14]. Moreover, MASC is a model for a multi-purpose parallel system which supports a large array of applications from grand challenge applications requiring massive parallelism to on chip parallelism (such as MMX or 3DNOW). The model permits task allocation when needed and embodies the intuitive and well-accepted method of data parallel programming.

Hillis and Steele defined a data parallel programming paradigm in the early 1980s, but this work did not provide a complete computational model [11]. The MASC model extends this concept to be a complete programming and computational model that embodies existing parallel compilers and systems. It is appropriate to have a well-defined model that emphasizes data parallel programming, even though other computational models may be capable of supporting it.

The data parallel style of programming is essentially very sequential in nature, and therefore is much easier to master by traditional sequential programmers than MIMD programming which employs task allocation, load balancing, synchronization points, etc. A standard associative paradigm provides true portability for parallel algorithms, and no specific machine architecture is required to effectively use the generalized MASC model [14]. This model standardizes the concept of associative computing and provides a basis for complexity analysis for data parallel and associative algorithms [9].

3.1 Description of MASC

The MASC model is a hybrid SIMD/MIMD model and is capable of both styles of programming. A frequent criticism of SIMD programming is that several PEs may be idle during *if-else* or *case* statements. The instruction streams provide a way to concurrently process conditional statements by partitioning the PEs among the ISs. Figure 3 shows a diagram of MASC.

The associative model (MASC) has an array of processing elements (PEs) and one or more instruction streams (ISs) that each broadcast their instructions to mutually exclusive partitions of PEs. Most applications require a small number of ISs in comparison to the number of PEs (although there are no firm restrictions). Each PE (or cell) has a local memory, and MASC locates objects by content or location in the combined local memory of the PEs. Each PE is capable of performing local arithmetic, logical operations and the usual functions of a sequential processor other than issuing instructions. A MASC machine with *j* ISs and *n* PEs is written as *MASC(n, j)*.

Cells may be active, inactive, or idle. Active cells execute the program which is broadcast from an IS. An inactive cell is considered in a group of IS cells, but does not execute instructions until the IS instructs inactive cells to become active again. Idle cells are currently inactive, not listening to any IS, and contain no essential program data but may be re-assigned as an active cell later.

ISs can be active or idle. An active IS issues instructions to a group of cells. An idle IS is not assigned to any PEs and is waiting until another IS forks, partitioning its PEs between itself and a new previously inactive IS. All PEs may be assigned using local data and comparisons. If an IS broadcasts some value to a set of PEs, the PEs could set this value to their active IS in the next instruction cycle, or choose not to switch. That is, a PE can change the IS to which it listens dynamically.

MASC supports data parallel reduction operations *and, or, min and max*; one or more instruction streams (ISs), each of which is sent to a distinct set in a dynamic partition of the processors; broadcasting from the ISs; and task assignment to ISs using control parallelism or data locality which allows PEs to switch ISs based on local data. The MASC model shown in Figure 3 has three networks, real or virtual: the PE interconnection network, the IS interconnection network, and the network between the PEs and ISs.

There are no restrictions on the type of cell network used with the MASC model. The programmer does not need to worry about the actual network hardware or the routing scheme, but only that MASC is capable of generalized routing with some latency.

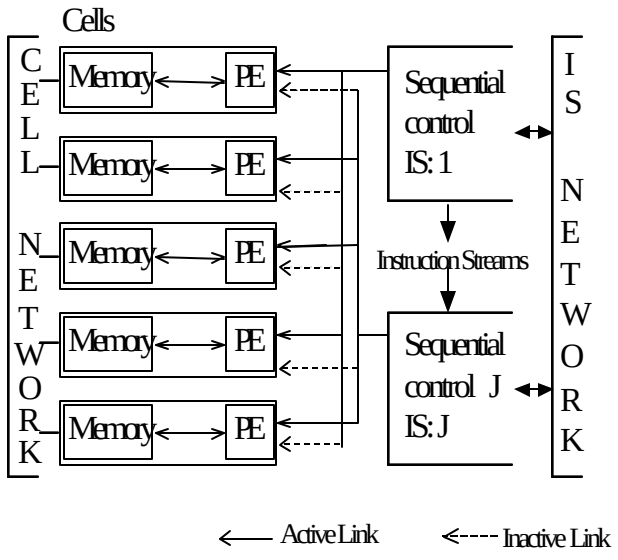

Figure 3: The MASC model

3.2 List of MASC Properties

The properties of cells (referred to as PEs):

- Each cell has local memory and a processing element.
- A network connects all n PE cells.

The instruction stream (IS) properties:

- Each IS has a copy of the program being executed and may broadcast an instruction or a word to its cells.
- The IS processors communicate control information with each other when groups of PEs are split or merged.
- Each cell is assigned to *only* one IS.
- Some or all cells may switch to a different IS in response to instructions from the current IS.
- An active cell executes the instruction sent from its IS, while inactive cells wait until they are commanded to be active again.
- An IS may unconditionally activate all assigned cells.
- An IS may conditionally force certain cells assigned to it to become idle.
- Idle cells are not assigned to any IS.

Associative properties:

- An IS can have all of its active cells execute an associative search.
- Active cells that perform a successful search are called responders.
- Unsuccessful active cells in the search are called non-responders.
- The IS may force either the set of responders or non-responders to be active.
- Previous sets of active cells can be restored by an IS.
- An IS may select one arbitrary cell from the list of active cells.

Global operations:

- An IS may compute the global ***AND***, ***OR***, ***MAX*** or ***MIN*** of data in all active cells.
- An IS may broadcast data to all its active cells.
- An IS may either read data from the memory of *one* specific cell or write data to the memory of *one* specific cell.

Control parallelism:

- Collectively, the ISs operate as a group of MIMD processors, although their primary purpose is to issue instructions to PEs and not to embody the main processing power of MASC. The standard control parallelism paradigms apply to instruction streams.
- The group of ISs primarily employ control parallelism providing fork and join operations, transfer of control information, coordination of the cells, and dynamic reallocation of idle cells.
- The ISs communicate control information and the transfer of active or inactive cells with each other during a fork or join operation.
- One way to transfer data between ISs is handled by reassignment of cells during forks and joins. The other way is to directly use the IS network.

Data routing:

- A routing operation on data held in each PE can be performed using a network that connects the PEs. This routing allows combining of data involving an arithmetic or logical operation (ex. +, max, min), which involves a cost in time based on the characteristics of the PE interconnection network.

4 Single instruction stream algorithm

A 2D knapsack algorithm for the single IS model with a 1D mesh (linear) network is presented in Figure 4. We require a simple network to propagate profit values down the array as the algorithm progresses, rather than a more complex network such as the hypercube.

// SINGLE IS PARALLEL ALGORITHM

1) if (L < W) {
   swap(L,W)
   for k = 1 to n swap(l_i,w_i) }
2) for all processors P_i do in parallel listening to IS_0 {
2.1) for j = 0 to W do { $F_{i,j}$ =0 ; $V_{i,j}$ =0 ; $H_{i,j}$ =0}
2.2) for k=1 to n
   for j = 0 to W do
   if ($i \geq l_k$ and $j \geq w_k$) $F_{i,j}$ = max{ $F_{i,j}$, p_k }
3) diag = 2
4) while (diag $\leq$ L +W) do {
4.1) for j = 0 to W do
   if (diag $\leq$ i + j and i > 0 and j > 0)
   { $V_{i,j} = V_{i-1,j}$; $H_{i,j} = H_{i,j-1}$ }
4.2) for j = 0 to W do {
   if (i * 2 + j = diag) $V_{i,j} = F_{i,j}$
   if (j * 2 + i = diag) $H_{i,j} = F_{i,j}$ }
4.3) for j = 0 to W do
   if (diag/2 $\leq$ i + j $\leq$ diag) {
   $sumV_{i,j} = F_{i,j} + V_{i,j}$
   $sumH_{i,j} = F_{i,j} + H_{i,j}$ }
4.4) for j = 0 to W do
   $F_{diag-j,j}$ = MascMax($sumV_{diag-j,j}$,
   $sumV_{diag-j-1,j}$ …, $sumV_{(diag-j)/2,j}$)
4.5) max_i = 0
4.6) for j = 1 to W do
   if (max_i < $sumH_{i,diag-1-j}$)
   max_i = $sumH_{i,diag-1-j}$
4.7) for j = 1 to W do
   $F_{i,diag-i}$ = max_i
4.8) diag = diag + 1 }}

Figure 4: Single IS MASC implementation of the 2D knapsack algorithm.

The parallelization given in Figure 4 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 $j \le b$ and those in column b and row i with $i \le a$. This data dependency prevents the parallel computation of a row or column of F values, 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, $diag/2 \le i+j \le diag$, 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 MASC 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, for each row the maximum of $sumH_{i,\,diag-i}$ through $sumH_{i,\,(diag-i)/2}$ is computed, saving the result in $F_{i,\,diag-i}$.

4.1 Analysis

By examining the steps in Figure 4, we calculate the running time of the algorithm. 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 MASC in step 4.5 completes in constant time (or in a factor $\log_k L$ where k is based on hardware considerations). 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 MASC algorithm is $O(n)+O(nW)+O(W(L+W))$ which reduces to $O(W(n+L+W))$.

The benefit of the associative version is that the number of processors is cost-optimal in relation to the sequential dynamic programming. 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.

5 Multiple instruction stream algorithm

Figure 5 shows the algorithm to solve the cutting-stock problem when we have available $max\{L,W\}$ ISs.

// MULTIPE IS PARALLEL ALGORITHM

1) if (L < W) {
   swap(L,W)
   for k = 1 to n swap(l_i,w_i) }
2) for all processors $P_{i,j}$ do in parallel listening to IS_0 {
2.1) $F_{i,\,j}=0$; $V_{i,\,j}=0$; $H_{i,\,j}=0$
2.2) for k=1 to n
   if ($i \ge l_k$ and $j \ge w_k$)
   $F_{i,\,j}$ = max{ $F_{i,\,j}$, p_k }
3) diag = 2
4) while (diag $\le$ L +W) do {
4.1) if (diag $\le$ i + j and i > 0 and j > 0) {
   $V_{i,\,j} = V_{i-1,\,j}$
   $H_{i,\,j} = H_{i,\,j-1}$ }
4.2) if (i * 2 + j = diag) $V_{i,\,j} = F_{i,\,j}$
   if (j * 2 + i = diag) $H_{i,\,j} = F_{i,\,j}$
4.3) if (diag/2 $\le$ i + j $\le$ diag) {
   $sumV_{i,\,j} = F_{i,\,j} + V_{i,\,j}$
   $sumH_{i,\,j} = F_{i,\,j} + H_{i,\,j}$ }
4.4) Set proc $P_{i,j}$ to listen to IS_j ***,For each IS_j do***
4.5) $F_{diag-j,\,j}$ =MascMax($sumV_{diag-j,\,j}$,
   $sumV_{diag-j-1,\,j}$,…,
   $sumV_{(diag-j)/2,\,j}$)
4.6) Set proc $P_{i,j}$ to listen to IS_i ***,For each IS_i do***
4.7) $F_{i,\,diag-i}$ =MascMax($sumH_{i,\,diag-i}$,
   $sumH_{i,\,diag-i-1}$,…,
   $sumH_{i,\,(diag-i)/2}$)
4.8) Set proc $P_{i,j}$ to listen to IS_0
4.9) diag = diag + 1 }}

Figure 5: Multiple IS 2D Knapsack Algorithm

Here we have one IS for every row and column. Here we describe the differences between the single IS version and the multiple IS method. Before the *MascMax* function

is called in steps 4.5 and 4.7, we switch from all PEs listening to IS_0, to each $PE_{i,j}$ listening to IS_j in step 4.4 and then IS_i in step 4.6. In other words, we have IS_j control column j, and IS_i control column i. Since an IS has the ability to find the maximum of all its controlled PEs in $O(1)$ (or logarithmic time), and we have sufficient ISs, then steps 4.5 and 4.7 can be completed in $O(1)$. For all of the sub-steps in step 4, the while loop from 2 to $L+W$ is the only sequential cost. Counting the non-parallel portions of the algorithm, step 1 is completed in $O(n)$, step 2.2 also takes $O(n)$, and step 4 finishes in $O(L+W)$ time. Therefore the entire algorithm requires $O(n+L+W)$ time if there are $max\{L,W\}$ ISs available.

5.1 With fewer ISs

The running time when we apply fewer ISs, $s < max\{L,W\}$, to the problem in Figure 3 is an acceptable ratio. A method spanning sections 4.4 to 4.8 that loops $\lfloor W/s \rfloor$ times for each column and $\lfloor L/s \rfloor$ times for each row is possible. Thus, the resulting run-time looping through *MascMax* is $O(n+(L+W)^2/s)$. If $s=O(max\{L,W\})$ then we again have the fully multiple IS algorithm in Figure 5. Figure 6 contains the modifications to make inreference to the Figure 5 algorithm from lines 4.4 to 4.8.

4.4) for k=0 to $\lfloor W/s \rfloor - 1$
  if (PE is in group k)
    Set proc $P_{i,j}$ to listen to $IS_{j \bmod s}$ ***,For each IS_j do*** {
4.5) $F_{diag-j,\,j}$ =MascMax($sumV_{diag-j,\,j}$,
  $sumV_{diag-j-1,\,j}$,…,
  $sumV_{(diag-j)/2,\,j}$) }
4.6) for k=0 to $\lfloor L/s \rfloor - 1$
  if (PE is in group k)
    Set proc $P_{i,j}$ to listen to $IS_{i \bmod s}$ ***,For each IS_i do*** {
4.7) $F_{i,\,diag-i}$ =MascMax($sumH_{i,\,diag-i}$,
  $sumH_{i,\,diag-i-1}$,…,
  $sumH_{i,\,(diag-i)/2}$) }
4.8) Set proc $P_{i,j}$ to listen to IS_0

Figure 6: MASC(LW, s) Implementation of 2D Knapsack Algorithm (fewer ISs)

6 Summary

It has been shown that both the single and the multiple IS MASC model has good computational characteristics. By maintaining a parallel variable for shifting data and computing on the diagonal, we have solved the data dependencies required. Table 1 shows comparative results for this algorithm and previous implementations. We have shown that using multiple instruction streams is a possible solution to solving complex problems. It is estimated that on a cluster even with a fast network, that communication would dominate, since there are on the order of L *(or W)* shifts being performed each iteration in the main loop of step 4. Each of these shifts on a 2D grid represent an order of L *(or W)* communications per shift or $O(L^2+W^2)$ shifts for step 4. When the LW sized table is divided up between a relatively small number of processors in a cluster, the number of messages sent will be on the order of the running time of the sequential algorithm. Therefore, the times presented for MASC with this particularly irregular algorithm indicate a result for MASC in comparison to theoretical models such as PRAM employing as much parallelism as possible. Future work that shows an exact mapping of MASC to a variant of PRAM will demonstrate that MASC is an efficient as well as a practical model.

Model	Time	Processors
Sequential	$O(LW(n+L+W))$	*1*
CRCW PRAM	$O(n+L+W)$	*LW*
2D Mesh	$O(max(L,W)(n+L+W))$	*LW*
Hypercube	$O(log(max(L,W))(n+L+W))$	*LW*
MMB	$O(n+L+W)$	*LW*
1 IS MASC +1D mesh	$O(min\{L,W\}(n+L+W))$	*max{L,W} PEs +1 IS*
Multi IS MASC +2D mesh	$O(n+L+W)$	*(LW) PEs + max{L,W} ISs*
Multi IS MASC +2D Mesh	$O(n+\frac{(L+W)^2}{s})$	*(LW) PEs + s ISs where* $s \le max(L,W)$

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

References

[1] R. Andonov and 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 comptational procedure for two-dimensional stock cutting, IBM J. Res. Develop., 16:462-469, 1967.

[4] P.Y. Wang and 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, pages168-171, 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, MASC: 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.
[10] Lisa Nicklas, Robert Atkins, Sanjeev Setia and Pearl Wang, Design and Implementation of a Parallel Solution to the Cutting Stock Problem', Concurrency: Practice & Experience, October 1998.
[11] S. Hillis, Data parallel algorithms, Communications of the ACM, 29(12):1170--1183, December 1986.
[12] K. F. Hioe, Asprol (associative programming language), Master's project, Kent State University, Math and Computer Science (MSB), August 1986.
[13] J. Potter, Associative Computing - A Programnming Paradigm for Massively Parallel Computers, Plenum Publishing, N.Y., 1992.
[14] Potter-Baker-Scott-Bansal-Leangsuksun-Asthagiri, MASC: An associative computing paradigm, IEEE Computer, pages 19-25, November 1994.