Progress report submitted

in partial fulfillment of the requirement for

CENG 505 - Parallel Computing I

PARALLEL ALGORITHMS

for

2D Cutting Stock Problems

By

Derya AKBULUT 200972007

Fall 2010

TABLE OF CONTENTS

1. INTRODUCTION

This report is written to analyze the applicability of parallel computing on cutting stock problems, specifically for the two dimensional case. Firstly, brief information is given about two dimensional cutting stock problems. Then a study from literature is presented as an example for parallel computing applications on 2D cutting stock problems. Finally, the construction algorithm of my thesis (An edge matching method for 2D irregular shaped cutting stock problems) is analyzed to see the points where parallelization is possible.

2. 2D CUTTING STOCK PROBLEMS

In general case of 2D cutting stock problems, there exist a rectangular stock material and rectangular parts to be produced by cutting the stock material in pieces (see Figure 1). The problem is generating a cutting pattern for the pieces such that the scrap of the stock material is minimized (see Figure 2)

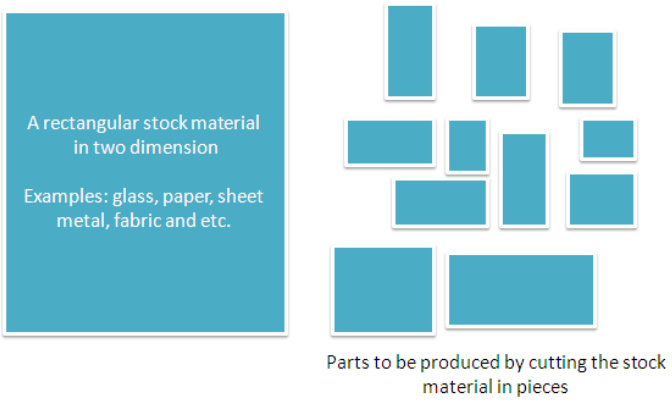

Figure 1 Stock material and the parts to be produced

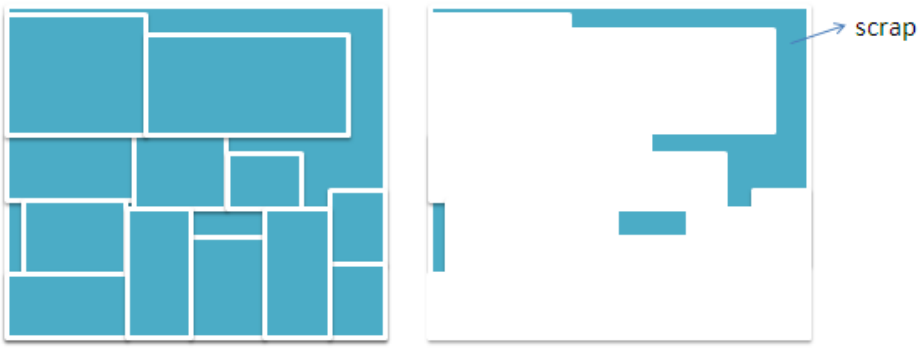

Figure 2 Illustration of a cutting pattern and related scrap

Cutting stock problems are discrete combinatorial optimization problems thus computational effort is required as the number of pieces increases. So parallelization may be advantageous.

3. AN EXAMPLE FROM LITERATURE

Since the problem requires high computational efforts due to its combinatorial nature, there are many parallel computing studies on the subject. In this section, "*Fine and coarse grained parallel algorithms"* of Armas et al. [1] is presented as an example application.

3.1 Problem Description

The problem, which is studied in the paper, is a regular 2D cutting stock problem with guillotine constraint. With guillotine constraint, the stock materials can be split into pieces using side by side cuts. Examples of non-guillotine and guillotine patterns are illustrated in Figure 3.

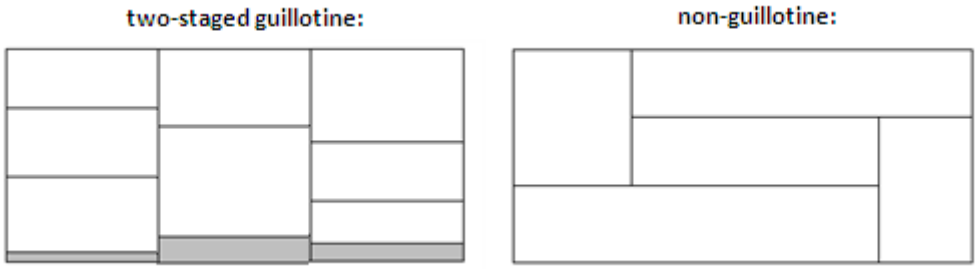

Figure 3 Guillotine and non-guillotine patterns

3.2 Solution Methodology

In related study [1], Viswanathan and Bagchi (VB) Algorithm is used to obtain cutting patterns. Then the sequential VB algorithm is modified in two different ways;

- Fine Grained Parallel Scheme (shared-memory scheme – OpenMP)
- Coarse Grained Algorithm (distributed scheme - MPI)

Before illustrating the parallel algorithms general structure of the original VB algorithm is presented in the following section.

3.2.1 VB Algorithm

The algorithm uses two lists: OPEN and CLIST. At each step best build of piece in OPEN is combined with the already found best meta-rectangles in CLIST to produce horizontal and vertical builds and the results are stored in the lists. Decision of 'best' is made using a simple function given in Equation 1. An illustration of meta-rectangles and the algorithm is given in Figure 4 and Figure 5 respectively.

$F`(\alpha) = g(\alpha) + h`(\alpha)$ *Equation 1.*

Where;

g(α): profit obtained from piece

h`(α):Profit can be obtained from remaining area

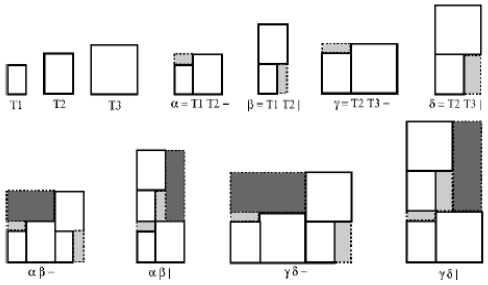

Figure 4 Examples of horizontal and vertical meta-rectangles. Shaded areas represent waste. [1]

1: OPEN := $\{T_1, T_2, \ldots, T_n\}$; CLIST := $\emptyset$;
2: **repeat**
3: Choose α meta-rectangle from OPEN with highest f' value;
4: **if** $(h'(\alpha) == 0)$ **then**
5: return(α);
6: **end if**
7: Insert α in CLIST;
8: **for** all β in CLIST **do**
9: $\gamma_H = \alpha\beta-$; /* *horizontal build* */
10: $l_{\gamma_H} = l_\alpha + l_\beta$; $w_{\gamma_H} = \max(w_\alpha, w_\beta)$;
11: $\gamma_V = \alpha\beta|$; /* *vertical build* */
12: $l_{\gamma_V} = \max(l_\alpha, l_\beta)$; $w_{\gamma_V} = w_\alpha + w_\beta$;
13: $g(\gamma_H) = g(\gamma_V) = g(\alpha) + g(\beta)$;
14: $x_i(\gamma_H) = x_i(\gamma_V) = x_i(\alpha) + x_i(\beta)\ \forall i \in [1, n]$;
15: **if** $((l_{\gamma_H} \leq L)$ and $(w_{\gamma_H} \leq W)$ and $(x_i(\gamma_H) \leq b_i\ \forall i))$ **then**
16: Insert γ_H in OPEN;
17: **end if**
18: **if** $((l_{\gamma_V} \leq L)$ and $(w_{\gamma_V} \leq W)$ and $(x_i(\gamma_V) \leq b_i\ \forall i))$ **then**
19: Insert γ_V in OPEN;
20: **end if**
21: **end for**
22: **until** forever

Figure 5 Viswanathan and Bagchi's Algorithm [1]

3.2.2 Fine Grained Parallel Scheme (shared-memory scheme – OpenMP)

First implementation is a shared memory scheme and uses OpenMP for the parallel execution of the horizontal and vertical combination loops. Each processor works on a section of CLIST structure but keeps a replicated copy of CLIST. OPEN is distributed among processors. The algorithm is given in Figure 6.

```
1: if (k == MASTER) then
2:    OPEN_k := {T_1, T_2, ..., T_n};
3:    sharedBestUpper[k] := StaticMap(element in OPEN_k with highest f');
4:    sharedBestUpId := k;
5: else
6:    OPEN_k := ∅;
7: end if
8: CLIST := ∅;
9: BestSol := LowerBound();    BestSolValue: = g(BestSol);
10: while (∃i/ OPEN_i ≠ ∅) do
11:    if (k == sharedBestUpId) then
12:       Remove α meta-rectangle from OPEN_k with highest f';
13:    else
14:       α := DynamicMap(sharedBestUpper[sharedBestUpId]);
15:    end if
16:    Insert α in CLIST;
17:    # pragma parallel for
18:    for (all β in CLIST / x_i(α) + x_i(β) ≤ b_i ∀i, l_β + l_α ≤ L) do
19:       γ_H = αβ−;      /* horizontal build */
20:       l_γH = l_α + l_β;  w_γH = max(w_α, w_β);  g(γ_H) = g(α) + g(β);
21:       x_i(γ_H) = x_i(α) + x_i(β) ∀i ∈ [1, n];
22:       if (g(γ_H) >BestSolValue) then
23:          Clear OPEN_k from BestSolValue to g(γ_H);
24:          BestSolValue = g(γ_H);    BestSol = γ_H;
25:       end if
26:       if (f'(γ_H) >BestSolValue) then
27:          Insert γ_H in OPEN_k at entry f'(γ_H);
28:       end if
29:    end for
30:    # pragma parallel for
31:    for (all β in CLIST / x_i(α) + x_i(β) ≤ b_i ∀i, w_β + w_α ≤ W) do
32:       γ_V = αβ|;      /* vertical build */
33:       l_γV = max(l_α, l_β);  w_γV = w_α + w_β;  g(γ_V) = g(α) + g(β);
34:       x_i(γ_V) = x_i(α) + x_i(β) ∀i ∈ [1, n];
35:       if (g(γ_V) >BestSolValue) then
36:          Clear OPEN_k from BestSolValue to g(γ_V);
37:          BestSolValue = g(γ_V);    BestSol = γ_V;
38:       end if
39:       if (f'(γ_V) >BestSolValue) then
40:          Insert γ_V in OPEN_k at entry f'(γ_V);
41:       end if
42:    end for
43:    sharedBestUpper[k] := StaticMap(element in OPEN_k with highest f');
44:    sharedBestSolV[k] := BestSolValue;
45:    if (k == MASTER) then
46:       sharedBestUpId:= i/ ∄j sharedBestUpper[j] > sharedBestUpper[i];
47:       sharedBestSolId := i/ ∄j sharedBestSolV[j] > sharedBestSolV[i];
48:    end if
49:    if (sharedBestSolValue[sharedBestSolId] != BestSolValue) then
50:       BestSolValue := sharedBestSolV[sharedBestSolId];
51:       BestSol := NULL;
52:    end if
53: end while
54: if (BestSol != NULL) then
55:    Return BestSol;
56: end if
```

Figure 6 Fine Grained Parallel Scheme [1]

3.2.3 Coarse Grained Algorithm (distributed scheme - MPI)

Second implementation is a distributed scheme and uses MPI for the parallel execution of the main search loop together with a flexible synchronization scheme and load balancing. Each processor works on a section of CLIST structure but keeps a replicated copy of CLIST. OPEN is distributed among processors. The algorithm is given in Figure 7.

```
1: OPEN_k := {T_{k+j*p} / k + j * p < n};    CLIST := ∅;
2: BestSol := LowerBound();    BestSolValue: = g(BestSol);
3: while (∃i/ OPEN_i ≠ ∅) do
4:    if (OPEN_k == ∅) or (time ≥ tPeriod) or (iters == iPeriod) then
5:       (λ, C, R) = AllToAll (BestSolValue, sizeof(OPEN_k), PC);
6:       if (max(λ) > BestSolValue) then
7:          Clear OPEN_k from BestSolValue to max(λ);
8:          BestSolValue = max(λ);    BestSol = NULL;
9:       end if
10:      CLIST = CLIST ∪ (R - PC);    PC = ∅;
11:      Π = {π_0, ..., π_{p-1}} partition of {S ⊗ T / S,T ∈ R; S ≠ T};
12:      Compute vertical and horizontal combinations of π_k;
13:      if (balanceRequired(C, minBalThresh, maxBalThresh)) then
14:         loadBalance(C, MaxBalanceLength);
15:      end if
16:      iters = time = 0;
17:   else
18:      Choose α meta-rectangle from OPEN_k with highest f';
19:      Insert α in CLIST and in PC;
20:      for (all β in CLIST / x_i(α) + x_i(β) ≤ b_i∀i, l_β + l_α ≤ L) do
21:         γ_H = αβ−;      /* horizontal build */
22:         l_{γH} = l_α+l_β; w_{γH} = max(w_α, w_β); g(γ_H) = g(α)+g(β);
23:         x_i(γ_H) = x_i(α) + x_i(β) ∀i ∈ [1, n];
24:         if (g(γ_H) >BestSolValue) then
25:            Clear OPEN_k from BestSolValue to g(γ_H);
26:            BestSolValue = g(γ_H);    BestSol = γ_H;
27:         end if
28:         if (f'(γ_H) >BestSolValue) then
29:            Insert γ_H in OPEN_k at entry f'(γ_H);
30:         end if
31:      end for
32:      for (all β in CLIST / x_i(α)+x_i(β) ≤ b_i∀i, w_β + w_α ≤ W) do
33:         γ_V = αβ|;      /* vertical build */
34:         l_{γV} = max(l_α, l_β); w_{γV} = w_α+w_β; g(γ_V) = g(α)+g(β);
35:         x_i(γ_V) = x_i(α) + x_i(β) ∀i ∈ [1, n];
36:         if (g(γ_V) >BestSolValue) then
37:            Clear OPEN_k from BestSolValue to g(γ_V);
38:            BestSolValue = g(γ_V);    BestSol = γ_V;
39:         end if
40:         if (f'(γ_V) >BestSolValue) then
41:            Insert γ_V in OPEN_k at entry f'(γ_V);
42:         end if
43:      end for
44:      iters = iters + 1;
45:   end if
46: end while
47: if (BestSol != NULL) then
48:    Return BestSol;
49: end if
```

Figure 7 Coarse grained parallel algorithm [1]

As it is seen in Figure 7, before each step the need for synchronization is checked (line 4). And also an all to all communication is existent (line 5) to be able to track the global best.

3.2.4 Computational Results

Computational results for 4 date sets are illustrated in Table 1, with the two parallel algorithms. For each algorithm there are two categories such as 'no dominances' and 'all dominances'. Here, dominances mean that the algorithms check the dominancy of a meta-rectangle so that if it is dominant no need to search for dominated figures, thus the search area is pruned. In the 'no dominances' category, the original algorithm is conserved.

Table 1 Computational Results [1]

Proc.	No Dominances		All Dominances	
	Fine-grained	Coarse-grained	Fine-grained	Coarse-grained
ATP33s				
1	4190.82	4420.71	19.3	19.53
2	3361.66	2617.68	9.29	10.82
4	2141.4	1410.84	5.43	5.88
8	1919.88	789.67	7.04	3.4
16	1678.36	394.66	5.61	1.99
ATP36s				
1	167.73	174.65	4.83	4.23
2	77.55	94.5	2.01	2.35
4	40.9	56.16	0.74	1.26
8	36.11	29.94	0.81	0.98
16	30.72	17.81	0.73	0.71
ATP37s				
1	1401.97	1648.75	15.78	17.44
2	640.15	860.2	8.63	9.72
4	692.27	435.83	4.9	5.24
8	1375.6	254.79	5.52	3.3
16	667.05	135.98	4.9	2.75
ATP39s				
1	70.64	84.28	13.67	14.68
2	27.62	43.15	6.54	8.02
4	8.79	23.47	3.89	4.33
8	3.83	14.51	2.97	2.55
16	2.5	7.48	3.29	1.52

As it is seen, parallelization improves the computation time for all cases. But speed up graphs may differ with respect to the data. For example, for ATP33s, ATP36s and ATP37s yields a better result for coarse grained case with no dominance and 16 processors but it is not true for

ATP39s. But the table is adequate to state that parallelization significantly speeds up the algorithm.

4. APPLICABILITY FOR MY THESIS

The title of my thesis is “An Edge Matching Approach for two dimensional irregular shaped cutting stock problems” Differently from general 2D cutting stock problems it considers irregular shaped parts to produce. An example for result pattern from the construction algorithm is illustrated in Figure 8.

Figure 8 Cutting pattern generated by the construction algorithm

The flow chart of the algorithm is illustrated in Figure 9. Algorithm uses the corner coordinates of each piece and the boundary. Then a smooth operation is done to reduce the number of the vertices within a tolerance limit. Then area calculations are done for each piece and the pieces are sort with respect to the areas. For the largest piece, align operations are handled. For align function. Corners of the pieces are matched with the corners of the boundary one by one. And for each corner edge matches are measured by rotating the piece left and right without any overlaps. The position that gives the best edge fit is selected and next largest piece is taken for align function. The algorithm returns when all pieces are inserted or the remaining area is not enough for a new piece. Due to this algorithm the tasks that are possible to parallelize can be stated as following:

- Smooth Function
- Sort and find largest:
    - Find areas for all shapes
    - Align function

- Interior angles for all shapes
- Rotation angles
- Coordinates after rotation

- Best fit to current boundary (A logarithmic speed up can be achived due to pruned search space by global best)

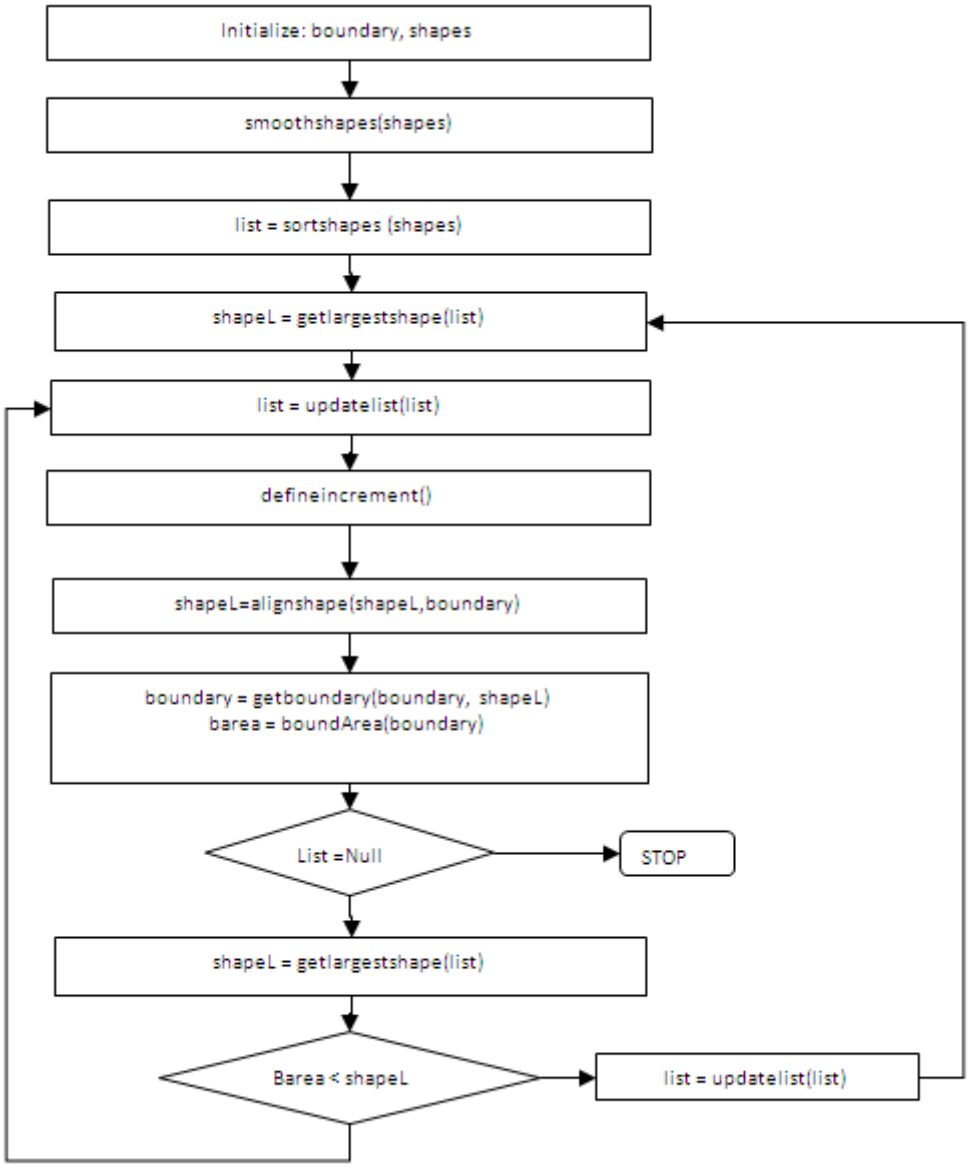

Figure 9 Construction algorithm

5. CONCLUSION

This report is written to analyze the applicability of parallel computing on cutting stock problems. The study of Armas et al. was an evidence of possibility to implement parallel applications for cutting stock problems. When the algorithm for the case in my thesis is considered, it is seen that a parallel algorithm can relax the computational limitations caused by the combinatorial nature of the problem. For future study, it is decided to parallelize the algorithm after its sequential structure is completed.

REFERENCES

[1] Armas J., Leon C., Miranda G., "*Fine and Coarse-grained Parallel Algorithms for the 2D Cutting Stock Problem*," pdp, pp.255-262, 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, 2010