CSE 708 Seminar

Subset Sum Count (Using OpenMP in C)

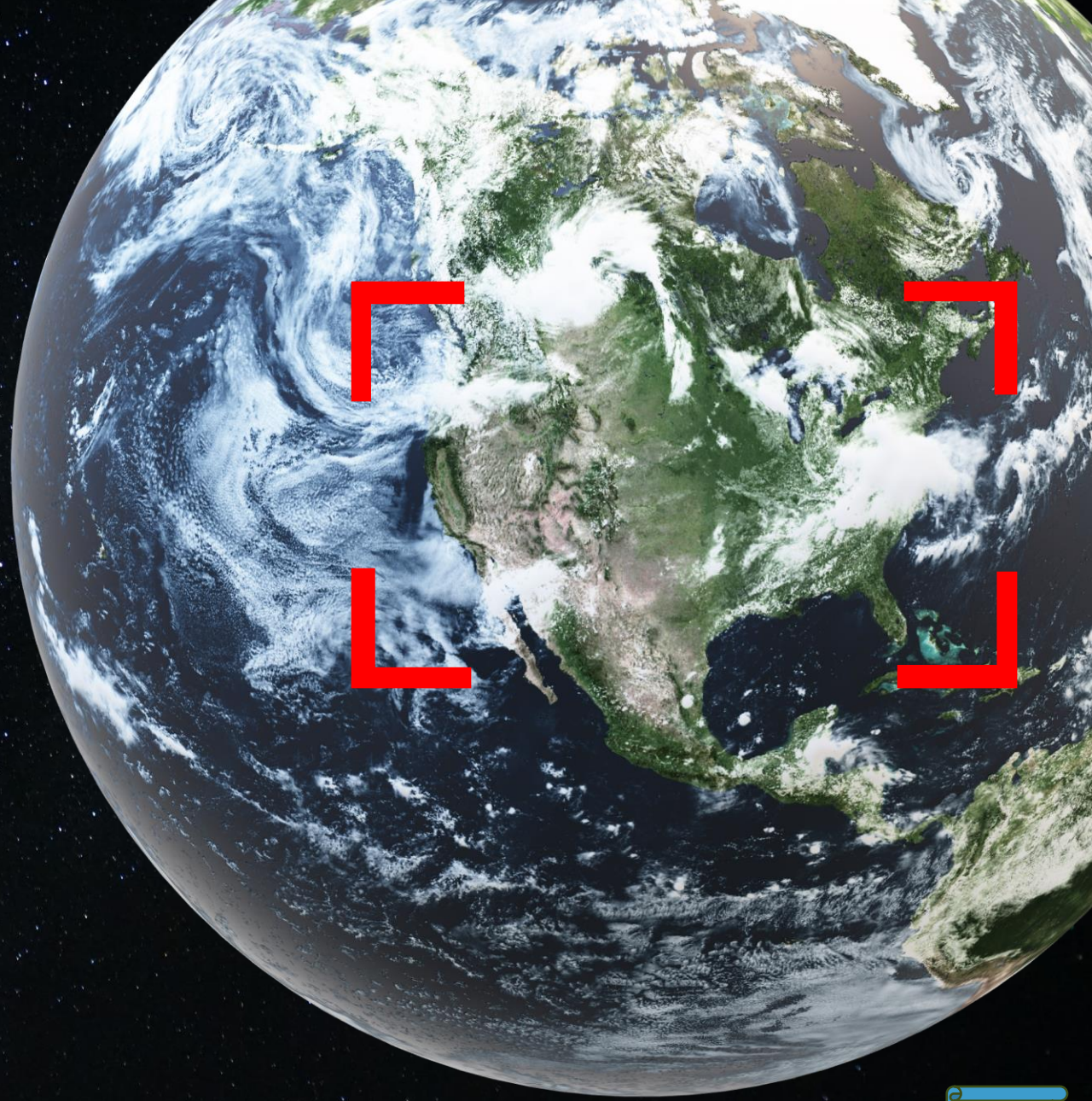
Name: Kunal Chand

UB Email: kchand@buffalo.edu

Person Number: 50465175

Instructor: Professor Russ Miller

Content



1) Problem Statement

Determine the count of the subsets within an array whose sum equals a given target sum.

array[ ] = { 3, 1, 2, 4, 5} Target Sum = 6

subsets = {3, 1, 2}, {4, 2}, {1,5}

Subset Sum Count = 3

Constraint: All array elements are whole numbers.

2) Sequential Solution

		0	1	2	3	4	5	6
-	0	1	0	0	0	0	0	0
3	1	1	0	0	1	0	0	0
1	2	1	1	0	1	1	0	0
4	3	1	1	0	1	2	-	-
2	4	-	-	-	-	-	-	-
5	5	-	-	-	-	-	-	-

if (array[i] > j): DP[i][j] = DP[i-1][j]

else: DP[i][j] = DP[i-1][j] + DP[i-1][j-array[i]]

2) Sequential Solution

		0	1	2	3	4	5	6
-	0	1	0	0	0	0	0	0
3	1	1	0	0	1	0	0	0
1	2	1	1	0	1	1	0	0
4	3	1	1	0	1	2	1	0
2	4	1	1	1	2	2	2	2
5	5	1	1	1	2	2	3	3

```java
static int subsetSum(int array[], int array_size, int sum){
    // Declaring and Initializing the DP matrix
    int DP[][] = new int[array_size + 1][sum + 1];
    DP[0][0] = 1;
    for(int i = 1; i <= sum; i++)
      DP[0][i] = 0;

    for(int i = 1; i <= array_size; i++){
        for(int j = 0; j <= sum; j++){
            // DP Formula
            if (array[i-1] > j)
              DP[i][j] = DP[i-1][j];
            else
              DP[i][j] = DP[i-1][j] + DP[i-1][j-array[i-1]];
        }
    }

    return DP[array_size][sum];
}
```

if (array[i] > j): DP[i][j] = DP[i-1][j]

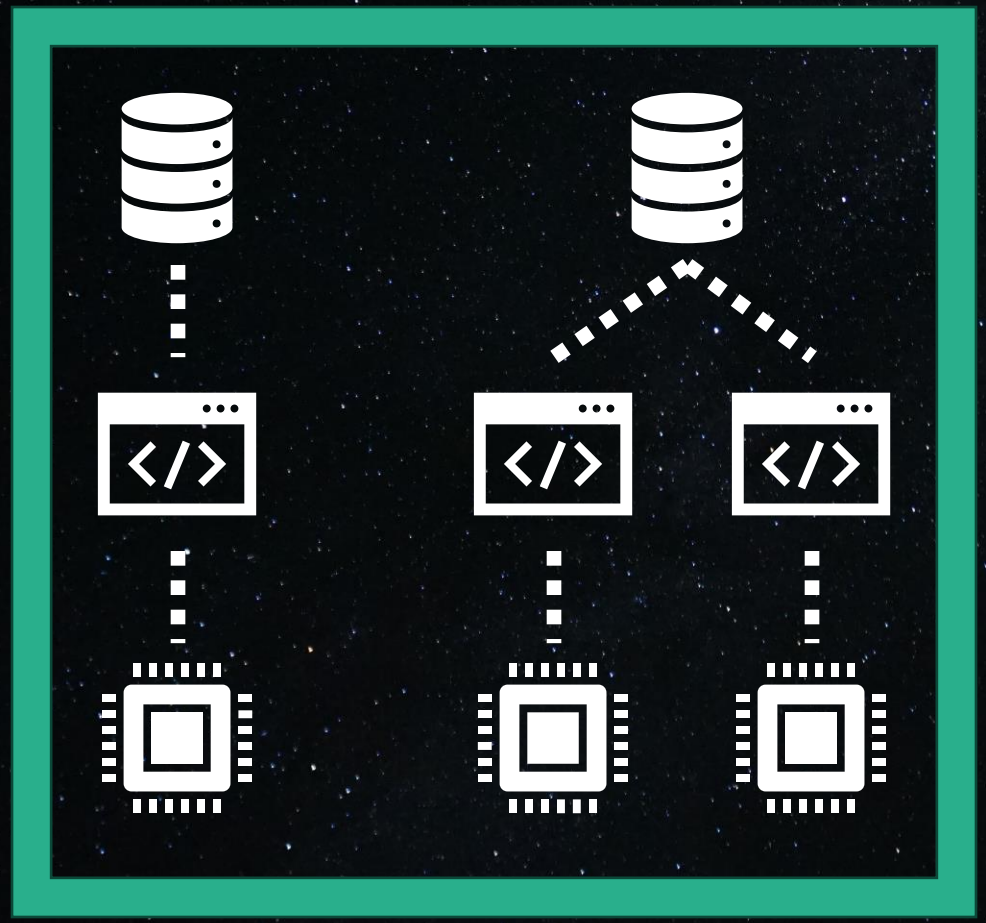
else: DP[i][j] = DP[i-1][j] + DP[i-1][j-array[i]]

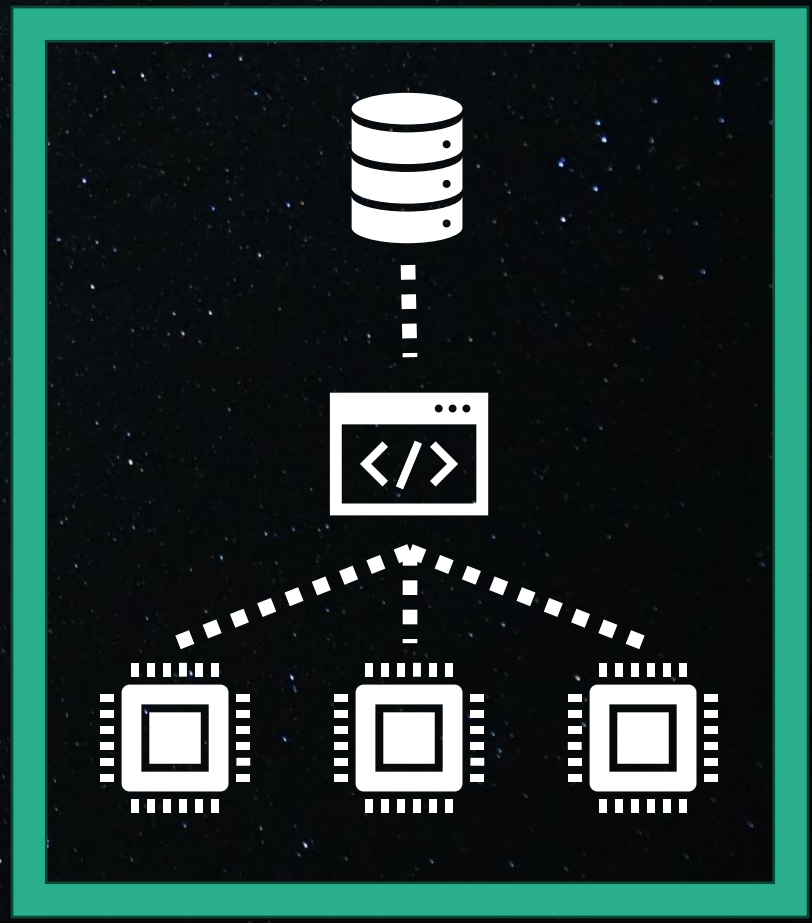
3) MPI vs OpenMP

Nodes

Tasks or Processes

Threads or Cores





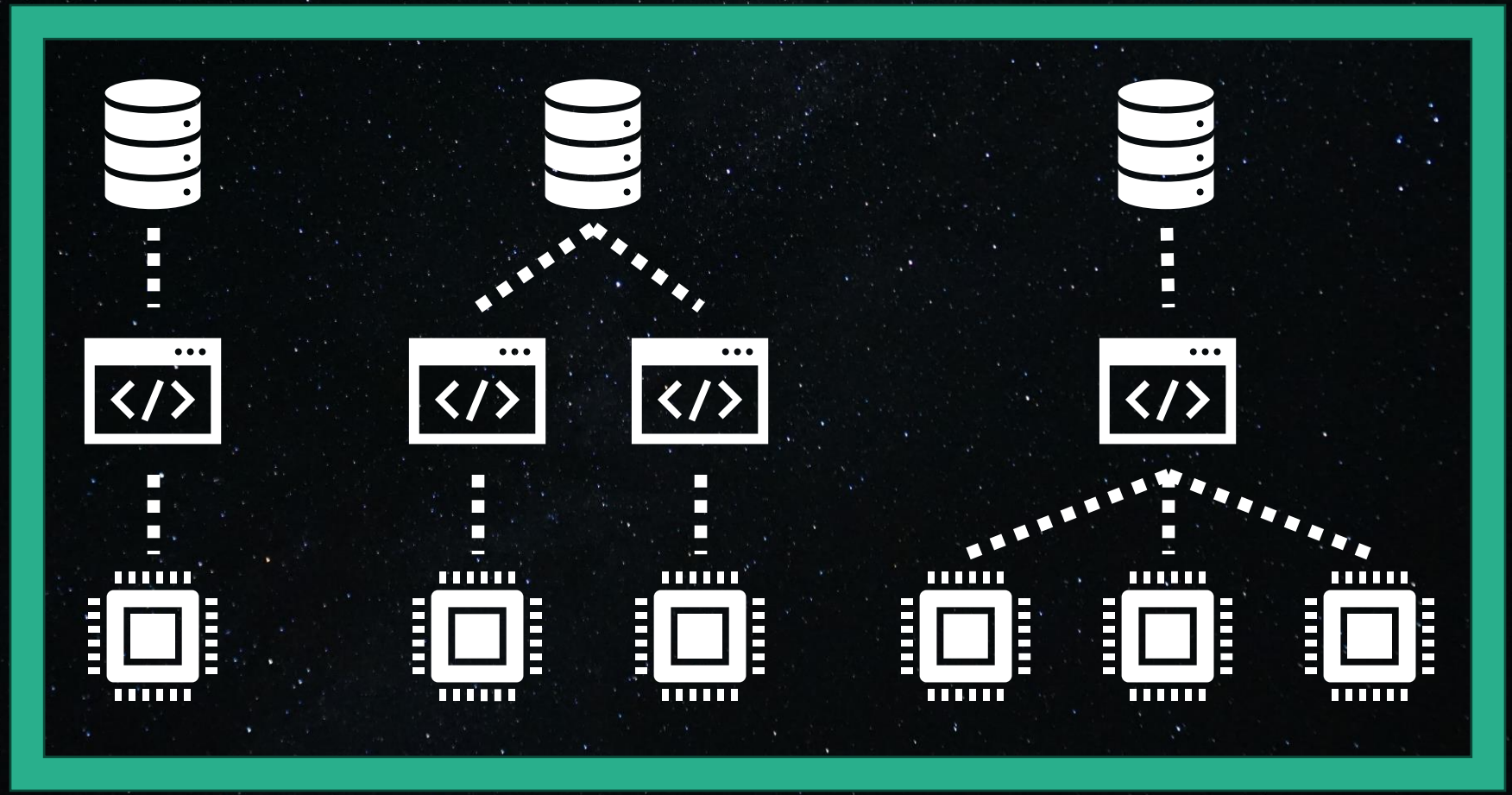
3) MPI vs OpenMP

Hybrid = MPI + OpenMP

Nodes

Tasks or Processes

Threads or Cores



3) MPI vs OpenMP

MPI

```
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=1
```

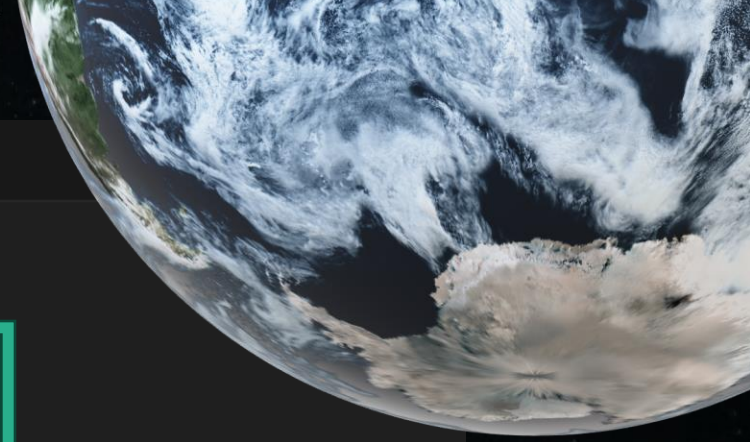
OpenMP

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=3
```

Hybrid

```
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=3
```

4) Slurm Script



OpenMP Configuration

Reserve all nodes & cores (Pizza Box) for exclusive use

Setting # of threads equal to # of CPU cores requested

Enables support for OpenMP pragmas and directives

slurm.sh

708 > open_mp_c_execution > slurm.sh

```
1   #!/bin/bash
2
3   #SBATCH --nodes=1
4   #SBATCH --ntasks-per-node=1
5   #SBATCH --cpus-per-task=6
6
7   #SBATCH --constraint=IB|OPA
8   #SBATCH --time=00:10:00
9   #SBATCH --partition=general-compute
10  #SBATCH --qos=general-compute
11
12  #SBATCH --job-name="ssc_6_thread"
13  #SBATCH --output=output_openmp.txt
14
15  #SBATCH --exclusive
16
17  module load intel
18
19  export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
20
21  gcc  -fopenmp  -o compiled_executive ssc_openmp.c
22
23  ./compiled_executive
24
```

5) Parallel Solution

Sequential Code Snippet

```
115  for (int i = 1; i <= n; i++){
116      for (int j = 0; j <= sum; j++){
117          if (a[i - 1] > j){
118              dp[i][j] = dp[i - 1][j];
119          }
120          else{
121              dp[i][j] = dp[i - 1][j] + dp[i - 1][j - a[i - 1]];
122          }
123      }
124  }
```

Parallel OpenMP Code Snippet

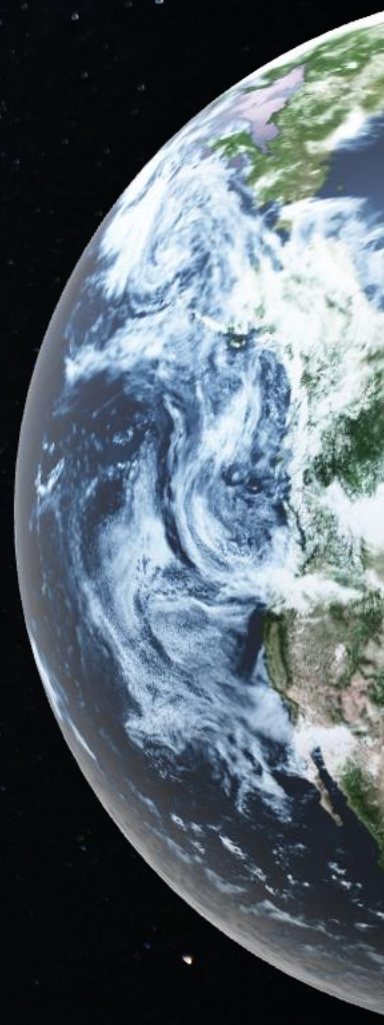
```
126  for (int i = 1; i <= n; i++) {
127      #pragma omp parallel for
128      for (int j = 0; j <= sum; j++) {
129          if (a[i - 1] > j){
130              #pragma omp atomic write
131              dp[i][j] = dp[i - 1][j];
132          }
133          else {
134              #pragma omp atomic write
135              dp[i][j] = dp[i - 1][j] + dp[i - 1][j - a[i - 1]];
136          }
137      }
138  }
```

5) Parallel Solution

```
140   // Row Wise Iteration
141   for (int i = 1; i <= n; i++) {
142       #pragma omp parallel for
143       for (int j = 0; j <= sum; j++) {
144           // Parallel column execution for a given row i
145       }
146       // Resumes sequential execution
147   }
```

```
126   for (int i = 1; i <= n; i++) {
127       #pragma omp parallel for
128       for (int j = 0; j <= sum; j++) {
129
130           int thread_id = omp_get_thread_num();
131           thread_table[i][j] = thread_id;
132
133           if (a[i - 1] > j){
134               #pragma omp atomic write
135               dp[i][j] = dp[i - 1][j];
136           }
137           else {
138               #pragma omp atomic write
139               dp[i][j] = dp[i - 1][j] + dp[i - 1][j - a[i - 1]];
140           }
141       }
142   }
```

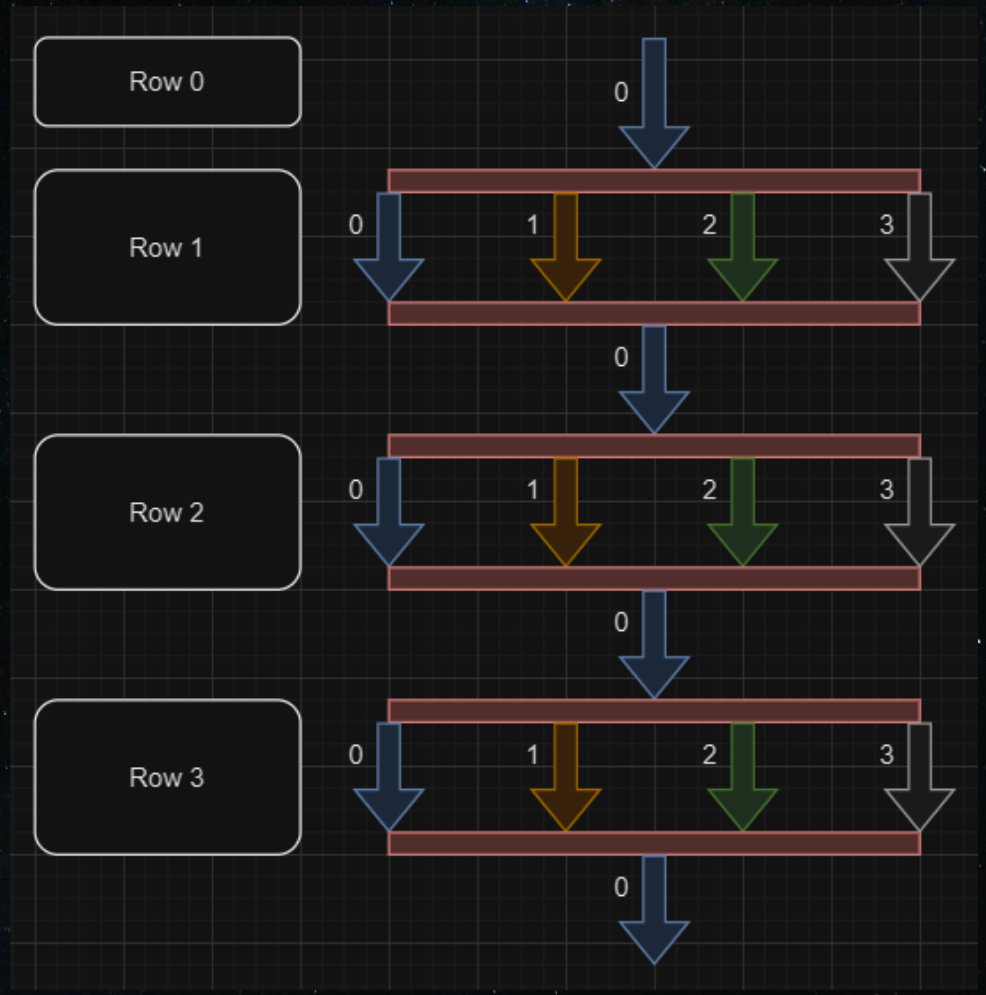
```
1   Max Threads = 4
2
3   Input Array = [3, 1, 2, 4, 5]
4   Target Sum = 6
5
6   Thread Table:
7   Row 0: 0 0 0 0 0 0 0
8   Row 1: 0 0 1 1 2 2 3
9   Row 2: 0 0 1 1 2 2 3
10  Row 3: 0 0 1 1 2 2 3
11  Row 4: 0 0 1 1 2 2 3
12  Row 5: 0 0 1 1 2 2 3
13
14  DP Table:
15  Row 0: 1 0 0 0 0 0 0
16  Row 1: 1 0 0 1 0 0 0
17  Row 2: 1 1 0 1 1 0 0
18  Row 3: 1 1 1 2 1 1 1
19  Row 4: 1 1 1 2 2 2 2
20  Row 5: 1 1 1 2 2 3 3
21
22  Subset Sum Count = 3
```



5) Parallel Solution

```
Max Threads = 4

Input Array = [3, 1, 2, 4, 5]
Target Sum = 6

Thread Table:
Row 0: 0 0 0 0 0 0 0
Row 1: 0 0 1 1 2 2 3
Row 2: 0 0 1 1 2 2 3
Row 3: 0 0 1 1 2 2 3
Row 4: 0 0 1 1 2 2 3
Row 5: 0 0 1 1 2 2 3

DP Table:
Row 0: 1 0 0 0 0 0 0
Row 1: 1 0 0 1 0 0 0
Row 2: 1 1 0 1 1 0 0
Row 3: 1 1 1 2 1 1 1
Row 4: 1 1 1 2 2 2 2
Row 5: 1 1 1 2 2 3 3

Subset Sum Count = 3
```

```
// Row Wise Iteration
for (int i = 1; i <= n; i++) {
    #pragma omp parallel for
    for (int j = 0; j <= sum; j++) {
        // Parallel column execution for a given row i
    }
    // Resumes sequential execution
}
```



6) Execution Result

extracted_values.log

708 > open_mp_c_execution > all_execution

```
1-core: 0.000057 seconds
2-core: 0.000291 seconds
3-core: 0.000289 seconds
4-core: 0.000401 seconds
5-core: 0.000339 seconds
6-core: 0.000485 seconds
7-core: 0.000532 seconds
8-core: 0.000604 seconds
9-core: 0.000676 seconds
10-core: 0.000680 seconds
11-core: 0.000628 seconds
12-core: 0.000863 seconds
13-core: 0.000910 seconds
14-core: 0.003784 seconds
15-core: 0.001814 seconds
16-core: 0.008463 seconds
```

7) Future Scope

1) Gather result on much input size and plot graph
2) Analyze Standard Speedup (Amdhal's Law) and Scaled Speedup (Gustafson's Law)
3) Access more cores
4) Explore Hybrid approach

8) Reference

1) GFG: https://www.geeksforgeeks.org/count-of-subsets-with-sum-equal-to-x/
2) Princeton University BootCamp: https://princetonuniversity.github.io/PUbootcamp/sessions/parallel-programming/Intro_PP_bootcamp_2018.pdf
3) Dr. Jones Lectures on OpenMP