0/1 KNAPSACK PROBLEM

CSE 633 – Parallel Algorithms

Prof. Russ Miller

Student Name: Lakshya Rawal

UB Person Number: 50459636

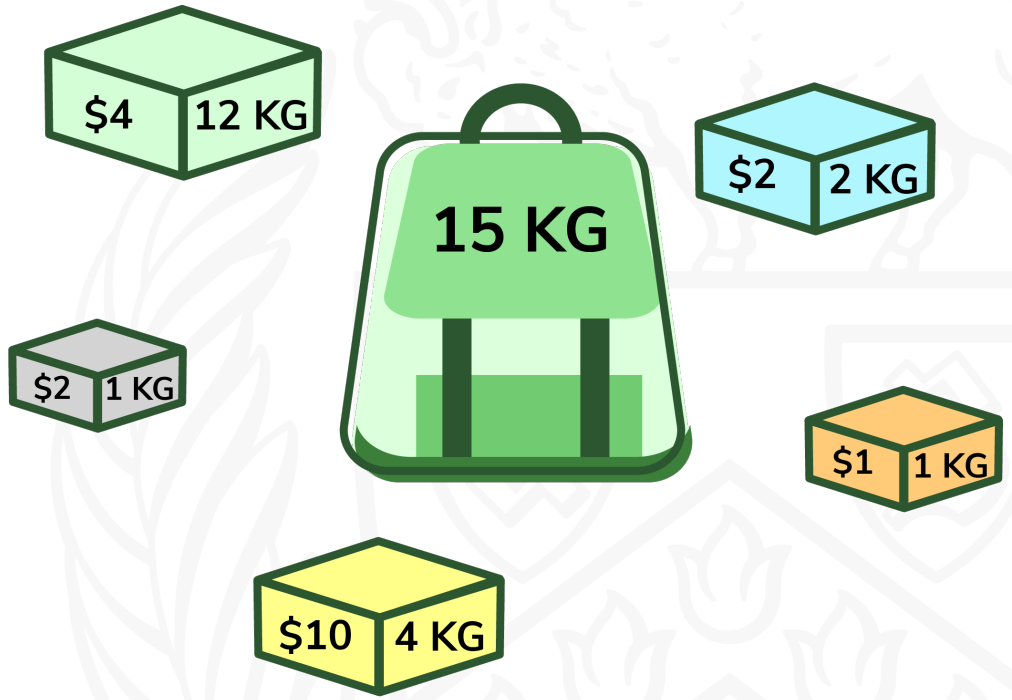
University at Buffalo The State University of New York

Outline

- Problem Statement
- Applications
- Brute Force Solution
- Sequential Algorithm
- Parallel Approach
- Results
- To Do
- References

Problem Statement

We are given **N** items where each item has some weight and value associated with it. We are also given a bag with capacity **W**, [i.e., the bag can hold at most W weight in it]. The target is to put the items into the bag such that the sum of values associated with them is the maximum possible.



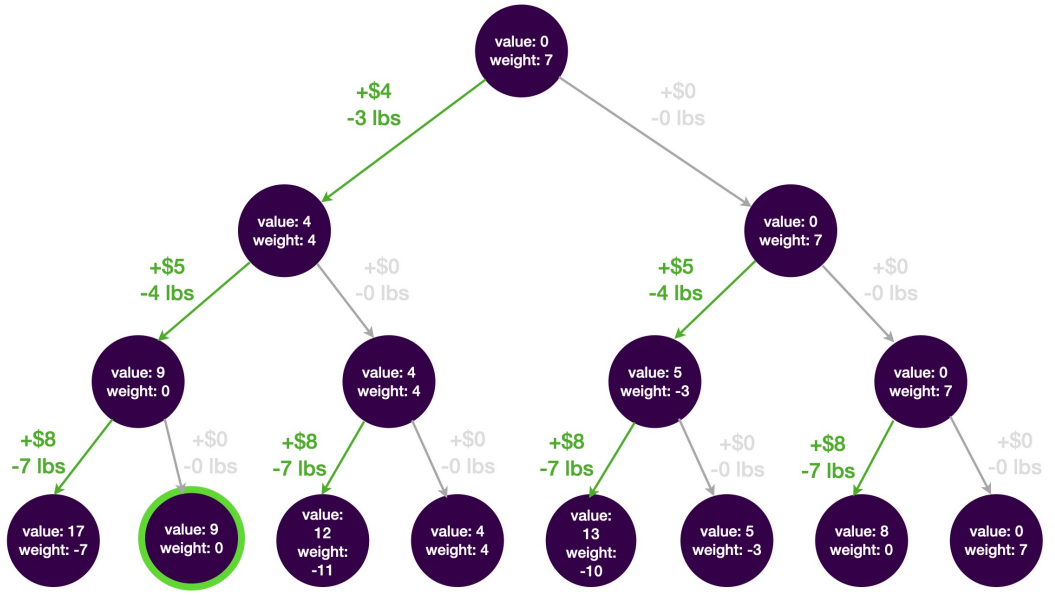
Applications

- Finding the least wasteful way to cut raw materials
- Selection of investments and portfolios, selection of assets for asset-backed securitization
- Generating keys for the Merkle–Hellman and other knapsack cryptosystems.

Brute Force Solution

This solution is brute-force because it evaluates the total weight and value of all possible subsets, then selects the subset with the highest value that is still under the weight limit.

- Time complexity: $O(2^n)$, due to the double the number of calls at each level
- Auxiliary space: $O(1)$, no additional storage is needed.



Sequential Algorithm: Dynamic Programming

Dynamic programming generates solution to knapsack problem in $O(n^2)$ time, a time controlled by the number of items and the maximum capacity in the problem.

```
int[][] DP = new int[N + 1][capacity + 1];

for (int i = 1; i <= N; i++) {
  // v -> value of the item and  w-> weight of the item
  int w = W[i - 1], v = V[i - 1];

      for (int sz = 1; sz <= capacity; sz++) {

        // Not including the item
        DP[i][sz] = DP[i - 1][sz];

        // Including the item if condition satisfies
        if (sz >= w && DP[i - 1][sz - w] + v > DP[i][sz]) DP[i][sz] = DP[i - 1][sz - w] + v;
      }
    }
 }
    // Return the maximum profit
    return DP[N][capacity];
}
```

Sequential Algorithm

- Time complexity: $O(N * W)$, where we create a matrix of number of items and each capacity
- Auxiliary space: $O(N * W)$, storage is needed to store the previous results.

V/W	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
2/3	0	0	0	2	2	2	2	2
2/1	0	2	2	2	4	4	4	4
4/3	0	2	2	4	6	6	6	8
5/4	0	2	2	4	6	7	7	9
3/2	0	2	3	5	6	7	9	10

Max Value: 10

Items Included: 3/2, 5/4, 2/1

Parallel Algorithm

Approach: Dividing columns between processors

- P Processors are assigned W/P columns which they compute independently
- P1 starts for the first column and fills up every nth column. If the condition is satisfied that it needs the value from another column it will call for the value using MPI_RECV.
- The Processor then uses MPI_ISEND to send all its calculated values to other processors in the columns they might need it.
- This value is then updated with the current value which is the value of the current item it is processing.

V/W	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
2/3	0	0	0	2	2	2	2	2
2/1	0	2	2	2	4	4	4	4
4/3	0	2	2	4	6	6	6	8
5/4	0	2	2	4	6	7	7	9
3/2	0	2	3	5	6	7	9	10
	P1	P2	P3	P1	P2	P3	P1	P2 …

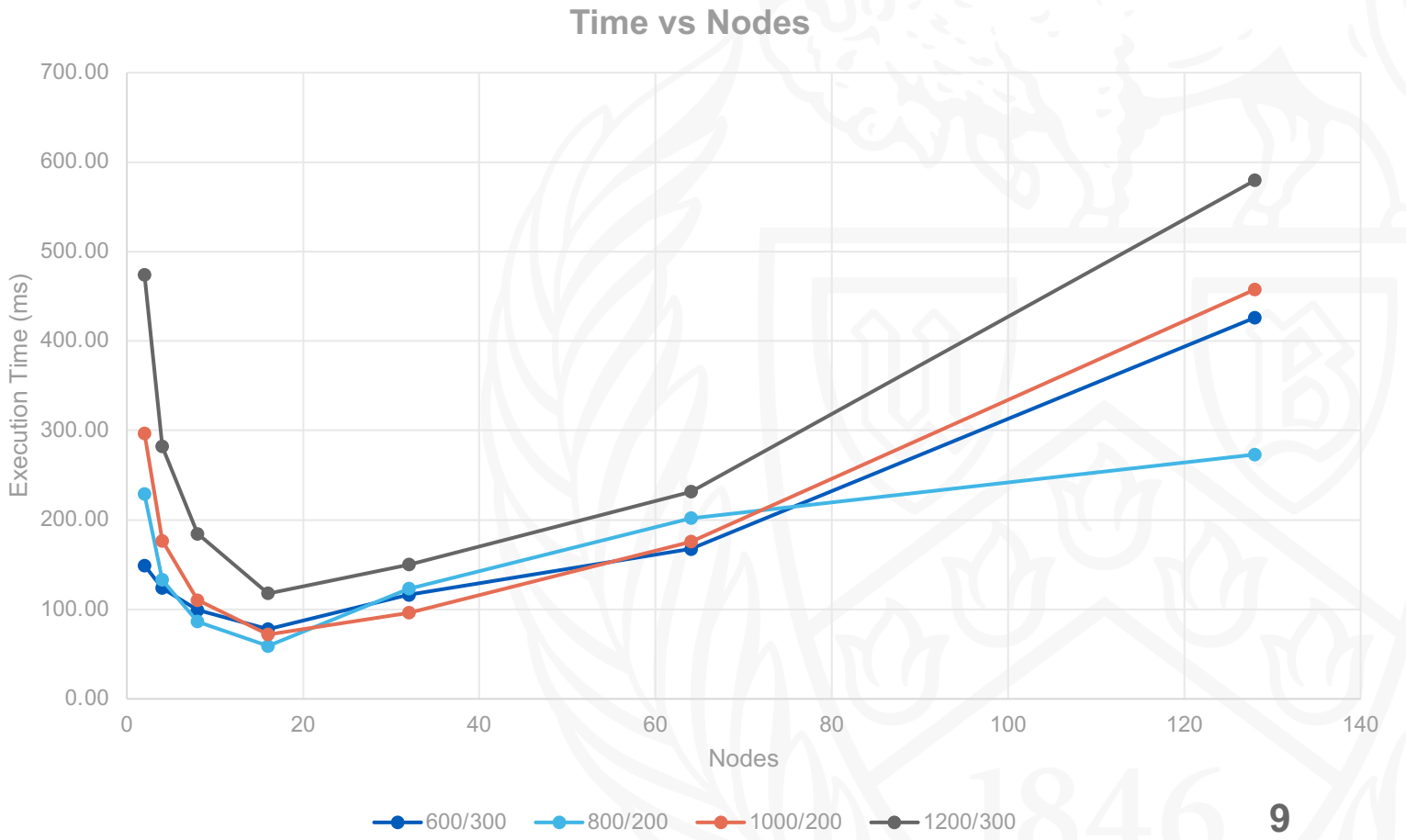
Experiment Results

This table depicts the run time of parallel algorithm in milliseconds with the increase in maximum weight and number of items in the knapsack problem

Nodes	600/300	800/200	1000/200	1200/300
2	149.21	228.83	296.89	474.13
4	123.84	133.16	176.52	282.23
8	99.22	86.30	110.21	184.62
16	77.93	59.19	71.95	117.76
32	116.35	123.16	96.09	150.10
64	167.51	201.80	175.83	231.43
128	426.10	273.16	457.60	579.56

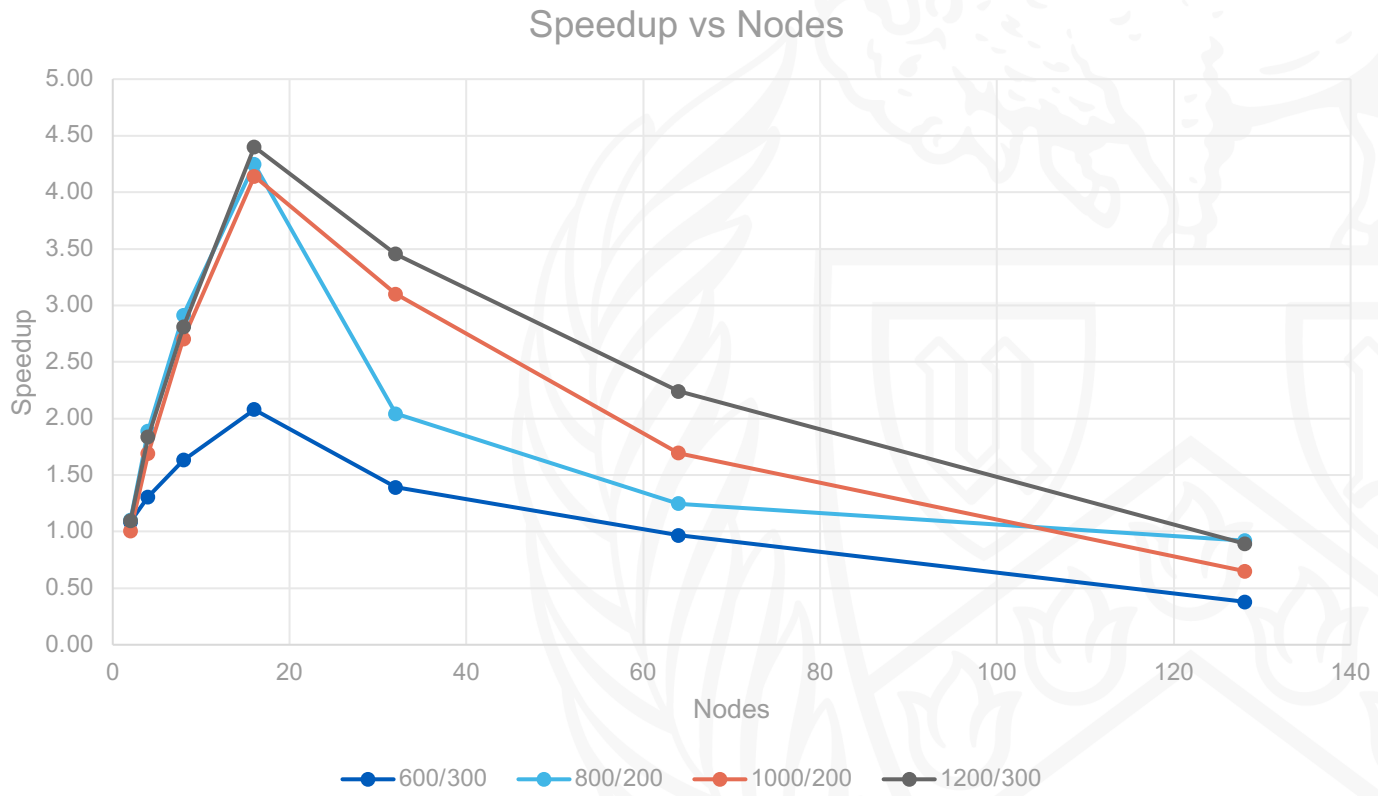
Graph and Analysis

We see a gradual decrease in run time of this algorithm as we increase the number of nodes in this approach as more processors are responsible for fewer columns.



Speedup vs Nodes

- When comparing this parallel approach to a sequential algorithm we were able speedup for decent input sizes whereas there are signs of diminishing returns after 64 processors.



Note: Speedup was calculated using the formula: Tseq / Tparallel

To Do

- Identify opportunity areas using MPI + OpenMP hybrid approach which is more popular for solving np hard problems
- Understand the use of CUDA in solving np complete problems considering the Integration of GPU Based Parallel Computing in this problem

References

- https://ubccr.freshdesk.com/support/solutions/articles/13000026245-tutorials-workshops-and-training-documents
- https://docs.ccr.buffalo.edu/en/latest/
- https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/
- https://www.educative.io/blog/0-1-knapsack-problem-dynamic-solution
- https://en.wikipedia.org/wiki/Knapsack_problem