Subset Sum
Team Cthulu

Tushar Iyer & Aziel Shaw

Rochester Institute of Technology

September, 2018
Outline

1. Introduction
   - Overview

2. Computational Problem
   - Topic - Subset Sum

3. Algorithms
   - Sequential Algorithm
   - Parallel Algorithm

4. Conclusion
   - Final Remarks
   - Questions
NP-Completeness & Dynamic Programming

- NP Completeness is a computational problem which belongs to both NP and NP-Hard problems
- Until P=NP is solved, we don’t know if there is an algorithm to solve NP-Complete problems in Polynomial time
- Dynamic programming allows us to speed up the computation of certain NP-Complete problems (If there are overlapping sub-problems) through use of Optimal Substructures.
Subset Sum

- Is an NP-Complete problem.
- Given a set of numbers and a target sum, find a set that totals that target sum.
- Dynamic Programming algorithm achieves best results both for performance and parallelization.
Our thought process...

- For subset sum, we had to use the dynamic programming version of SubsetSum as that has for loops which we are able to parallelize.

Algorithm Input and Output:

- **INPUT**: A set of numbers, a target sum, length of the set.
- **OUTPUT**: A set of numbers that satisfy the sum, or an empty set is none are found.
Sequential Algorithm

Algorithm 1 SubsetSumSeq

Ensure: A subset which adds up to a given sum.

1: procedure SubsetSeq(Set, Sum)
2:   for index 0 to n do
3:     refTable[index][0] = TRUE
4:   for index 1 to n do
5:     currPos = set[index]
6:     for index2 0 to setlen - 1 do
7:       refTable[index][index2] = refTable[index - 1][index2]
8:     for index2 0 to sum do
9:       newPos = currPos + index2
10:      if bitAt(refTable[index][index2]) == 1 AND newPos <= sum then
11:         changeValue(refTable[index - 1], newPos)
Algorithm 2 SubsetSumSmp

Ensure: A subset which adds up to a given sum.

1: procedure SubsetSeq(Set, Sum)
2:   parallelFor index 0 to n do
3:     refTable[[index][0]] = TRUE
4:   end
5:   for index 1 to n do
6:     currPos = set[index]
7:     parallelFor index2 0 to setlen do
8:       refTable[index][index2] = refTable[index - 1][index2]
9:     end
10:    parallelFor index2 0 to sum do
11:      newPos = currPos + index2
12:      if bitAt(refTable[index][index2]) == 1 AND newPos <= sum then
13:        changeValue(refTable[index - 1], newPos)
14:    end
Final Remarks

Our hypothesis...

- We think that the time will increase proportionate to the size of the total set.
- Parallelization will help, but only to an extent
- Beyond a threshold, we will be subject to diminishing returns
Any Questions?