Report on Compute Assignment

Trivial solution

We can generate every possible subset having houses serially and among those (in total n^2), there will be at least one subset with a summation that matches/ smaller closest of *max*.

Insights

1. Time complexity is O(n^2).
2. There are repeated summations calculations among the different sets of all possible serial subsets.

Idea

- Memoization

  We can store the summation we are calculating now for future use to calculate the summations of other subsets.

Solution

- Prefix sum

```
void seqPrefixSumFunc(vector<int> &home, int homes, vector<int>
&prefixSum)
{
    prefixSum[0] = home[0];
    for(int i = 1; i < homes; i++)
    {
        prefixSum[i] = prefixSum[i - 1] + home[i];
    }
}
```

Finding the range of houses

We will use prefix sum to populate the 2-D matrix (as symmetric, so only one upper triangular matrix is populated).

Idea

The prefix sum tells the summation result from the beginning to a particular index. To figure out the prefix sum between two specific indices, we can subtract the prefix sum from a bigger to a smaller indexed prefix sum. `table[row][col] = table[0][col] - table[0][row - 1];`

To find the range of the houses, we want either to match with the *max* or at least get the smaller closest to it. Here is the implementation.

```
void seqFindRange(vector<int> &home, vector<int> &prefixSum, int
&startHome, int &endHome, int &candies, int max, int homes)
{
    vector<vector<int>> table(homes, vector<int>(homes));
    startHome = -1;
    endHome = -1;
    int diff = numeric_limits<int>::max();

    for(int row = 0; row < homes; row++)
    {
```

```
        for(int col = row; col < homes; col++)
        {
            if (row == 0)
                table[row][col] = prefixSum[col];
            else if (row == col)
                table[row][col] = home[col];
            else
                table[row][col] = table[0][col] - table[0][row -
1];

            if (table[row][col] <= max && diff > max - table[row]
[col])
            {
                diff = max - table[row][col];
                startHome = row + 1;
                endHome = col + 1;
                candies = table[row][col];

            }
            if (diff == 0)
                break;

        }
    }
}
```

Idea

The operations to find the candies for varied ranges (i,j) are independent operations, coming from the prefix sum. It can be parallelized!

Parallel implementation using OpenMP

Challenges

- Parallelize the prefix sum function.
- Parallelize the find range function.

To parallelize the prefix sum, we could have used the regular binary tree approach by partitioning. However, for each level we visit, more and more threads will kept idle and the load balance will not be good. As a consequence, more waiting, underutilization, and poor performance.

Idea

Instead of partitioning the data and calculating the final prefix sum on each subset of data, we have taken a different approach. We jumped over the data in such a way, that there will partial prefix sum calculated traversing the whole data in parallel. It takes ceil(Log2(n)) rounds. After finishing all the rounds, the final partial sum can be found.

Interesting fact

- The reading of data must be done for all threads before writing. To trick openMP, we used two different parallel for loops to take advantage of implicit barrier functionality.

```
        #pragma omp parallel for
        for (int i = stepSize; i < homes; i++)
        {
            hold[i] = prefixSum[i] + prefixSum[i - stepSize];
        }
        #pragma omp parallel for
        for (int i = stepSize; i < homes; i++)
        {
            prefixSum[i] = hold[i];
        }
```

- The load balance is done automatically and all the threads are equally busy in each round improving the utilization of hardware and lowering the waiting time.

The next challenge is to parallelize the findRange() function. As we mentioned, the operations are independent of calculating the number of candies, we can assign each thread to calculate the candies count independently by placing the `\#pragma omp parallel for reduction(closestMax:grabber)` in front of the outer for loop.

Challenge

- In sequential code, we have taken the notes of candies count and start and ending number of the range based on the diff (difference) between the max and current candies count (must be lower). New lower overrides the older higher values of diff. Each time it lowers the value, we take the notes of candies and start and ending index.

```
if (table[row][col] <= max && diff > max - table[row][col])
            {
                diff = max - table[row][col];
                startHome = row + 1;
                endHome = col + 1;
                candies = table[row][col];

            }
```

In parallel setting, we are assigning threads to work independently, this diff, startHome, endHome, candies variables, stored in main memory, can create race conditions, thus engendering erroneous results.

Solution

One way to handle by using a critical region, which will serialize the code and create a longer waiting time as only one thread can enter into the region to maintain consistency.

Better solution

Usage of custom reduction.

```
struct rangeGrabber{
    int candies;
    int start;
    int end;
    rangeGrabber() : candies(-1), start(-1), end(-1){}
};

#pragma omp declare reduction(closestMax : rangeGrabber : omp_out =
omp_in.candies > omp_out.candies? omp_in : omp_out)
```

We want to take notes of candies and start and ending indices for the maximum candies while not greater than *max*. We are getting independent of using main memory shared space, rather we are providing a custom data structure specifying the behavior. All we want, indices, and candies count using reduction on maximum candies <= *max*.

```
vector<vector<int>> table(homes, vector<int>(homes));
    startHome = -1;
    endHome = -1;
    int diff = numeric_limits<int>::max();
```

```
    #pragma omp parallel
    for(int col = 0; col < homes; col++)
    {
        table[0][col] = prefixSum[col];
    }

    rangeGrabber grabber;
    #pragma omp parallel for reduction(closestMax:grabber)
    for(int row = 0; row < homes; row++)
    {
        for(int col = row; col < homes; col++)
        {
            if (row != 0)
                table[row][col] = prefixSum[col] - prefixSum[row -
1];

            if ( table[row][col] <= max)
            {
                grabber.candies = table[row][col];
                grabber.start = row + 1;
                grabber.end = col + 1;
            }
        }
    }
```

Can we do better?

So far sequential code has quadratic time complexity and parallel code is sped up by the factor of the number of processors. Is there a linear (O(n)) exist? - **No!**

Proof

We need to know the highest candies count <= max. For any unsorted houses based on candy count, each of the houses is the starting candidate and has the potential to have the highest candy count. Any algorithm needs to traverse each of the items. Any algorithm will have the complexity at least O(n). However, for each item in traversal, is there a constant time optimal substructure of dynamic programming that exists for this problem? It exists only when there is an exact match with *max*. However, the question says there can be inequality of candies quantity <= max. So, we don't know where the best starting index of the range is after the current house inclusion in dynamic programming when quantity is strictly < max. So, we need to iterate over all past values (in the worst case) to figure out the best starting position for the current inclusion. Thus, this problem can never be solved in linear time.

Performance gain

For a randomly generated relatively large input.txt file has been fed into the code to get the idea of speed up.

```
const int max_homes = 10000;
    const int max_pieces = 1000;
    int homes = rand() % max_homes + 1;

    int max = rand() % max_pieces + 1;
    outfile << max << std::endl;

    for (int i = 0; i < homes; ++i) {
        outfile << (rand() % max_pieces + 1) << std::endl;
    }
```

The thread count is set 32 and the speed up is **4.875**.

```cpp
int main()
{
    static vector<int> home;
    static int homes;
    static int max;
    string fileName = "input1.txt";
    readFromFile(home, homes, max, fileName);
    vector<int> prefixSum(homes);
    int startHome, endHome, candies;

    int timeSeq = serialCode(home, prefixSum, startHome, endHome, candies, max, homes);

    int timePar = parallelCode(home, prefixSum, startHome, endHome, candies, max, homes);

    cout<< "Serial: "<< timeSeq << endl <<"Parallel: "<<timePar << endl;
    return 0;
}
```

PROBLEMS 58 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(base) sskg8@compute-17-5:~/code/compute-assignment$ g++ -fopenmp code.cpp -o code && ./code
Start at home 1488 and go to home 1492 getting 1413 pieces of candy
Start at home 2497 and go to home 2499 getting 1413 pieces of candy
Serial: 78
Parallel: 16
(base) sskg8@compute-17-5:~/code/compute-assignment$
```