Project 1: Parallel implementation of vector accumulation

Provided file: **Chem281P1.cpp**:

In this project you will have an opportunity to apply the parallel programming skills you developed during our live discussions. You are asked to use OpenMP to parallelize the often-used accumulation of the entries in a vector. Mathematically, the problem is defined as

$$x_i = \sum_{k=0}^{i} y_k$$

This problem can be implemented in C++ using the well-known recurrence relation

```
#define N <num>
int y[N];
for (unsigned int i=1; i<N; i++)
  y[i] += y[i-1];
```

The computation of any entry of $\mathbf{y}$ depends on the computation of all the previous elements of the array. This data dependency prevents parallelizing this algorithm.

One possible solution is to use a divide-and-conquer approach. We will *logically* split this array into as many disconnected chunks as threads we are using. We can compute the beginning of each chunk (partition) using the thread number of the executing thread. Now, we can can apply the serial code independently on each chunk. However, the computation is not finished yet. Now, we have to add to each chunk the accumulated value of the last entry in the preceeding chunk.

This requirement adds another serialization dependency. However, we can circumvent this dependecy by using an auxiliar array of size equal to $number\_of\_threads+1$. Each thread will store the maximum value of the accumulated sum in its corresponding array in the entry $thread\_number+1$ of the auxiliar array. The 0-th entry in this auxiliar array has previously been set to 0. Now, we need to choose one thread to accumulate all the entries in this array. But this array is very small, its length is $number\_of\_threads+1$. However, before performing this computation we must ensure that all the

threads have finished the previous task. We can use an barrier for this purpose.

When the single thread finished updating the small array, now every thread can read their corresponding values from this array and add it to all the entries in their own partition. Again, we need a barrier before this task to ensure that the single thread updating the small array has completed its task.

Your assignment:

- Implement the parallel version of the accumulation algorithm.
- Run the algorithm using only one thread. Check the execution time. Compare this time to the time taken by the serial version. Explain the difference
- Run your code using different number of threads. Explain what you see
- Make sure your code computes the right result regardless of the number of threads used.

After you successfully completed your programming assignment, address the following questions

- What is the overhead incurred by parallelization?
- How does your code scale as you add more threads?
- Is this algorithm compute bound or memory-bandwidth bound?

The file **Chem281P1.cpp** provides a number of auxiliary functions to help you implement this project. It handles the timing, argument parsing, result comparison, data generation, etc. It also provide suggestions to help you implement your code.