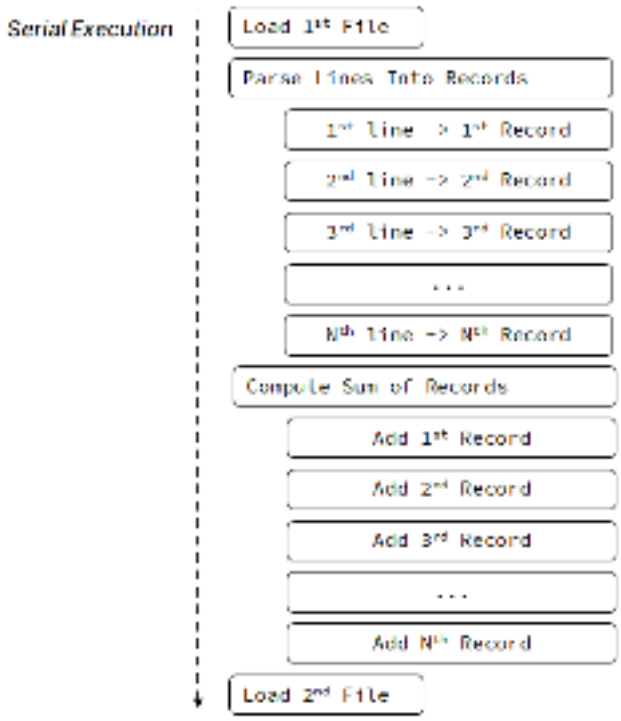
Using Parallel Algorithms

This lesson highlights the advantages of using parallel algorithms to implement the CSV reader application we just made.

Previously the code was executed in a sequenced way. We can illustrate it in the following diagram:

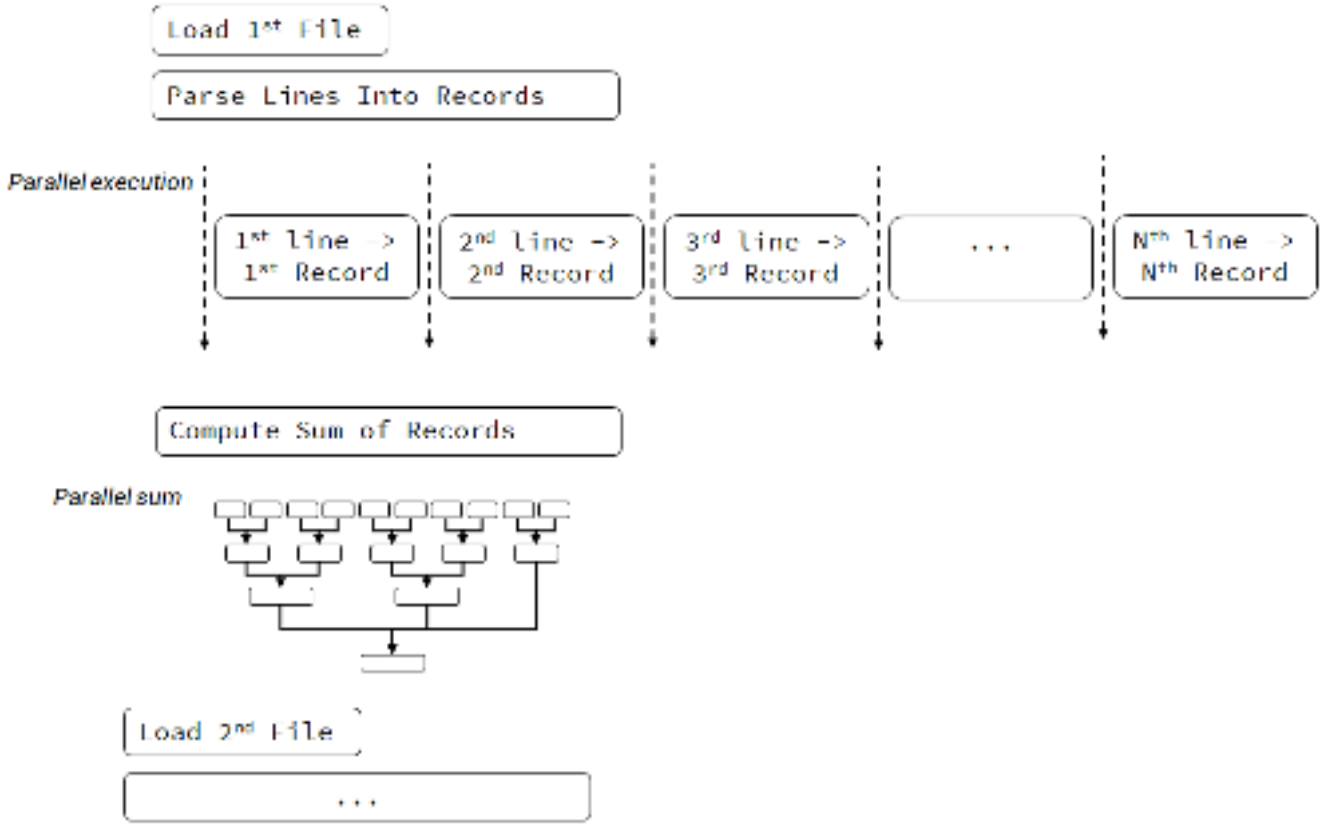


We open each file, process it, calculate, then we go to another file. All this happens on a single thread.

However, there are several places we can consider using parallel algorithms:

- Each file can be processed separately
- Each line of a file can be converted independently into the Record Data
- Calculations can be enhanced with parallel execution

If we focus on the second and the third option, we can move into the following execution model:



In the above diagram, we can see that we're still processing file one by one, but we use parallel execution while parsing the strings and making the calculations.

When doing the conversion, we have to remember that exceptions won't be re-thrown from our code. Only `std::terminate` will be called.

> 💡 As of December 2018, only the MSVC compiler supports parallel execution in the Standard Library. The parallel version of the example does not work with GCC or Clang. It's possible to use a third party library like Intel Parallel STL or HPX. You can, however, compile the serial version (`csv_reader.cpp`) and play with the code. It was tested on GCC 8.2. Unfortunately, it fails to compile with Clang. The serial version of this program is present in The Serial Version lesson. You can use the sample CSV files present there.

Let's look at how to get the best performance with parallel algorithms, in the next lesson.