Parallel Calculations

Now, it's time to refactor the CalcTotalOrder function using parallel algorithms.

WE'LL COVER THE FOLLOWING

Arithmetic Operations in Parallel Algorithms

Another place where we can us parallel algorithms is `CalcTotalOrder()`.

Instead of `std::accumulate` we can use `std::transform_reduce`.

As mentioned in the Parallel Algorithms chapter, the floating point sum operation is not associative. However, in our case, the results should be stable enough to give 2 decimal places of precision. If you need better accuracy and numerical stability, please consider using a different method.

```
double CalcTotalOrder(const std::vector<OrderRecord>& records,
                      const Date& startDate, const Date& endDate)
{
  return std::transform_reduce(
    std::execution::par,
    std::begin(records), std::end(records),
    0.0,
    std::plus<>(),
    [&startDate, &endDate](const OrderRecord& rec) {
      if (rec.CheckDate(startDate, endDate))
        return rec.CalcRecordPrice();

      return 0.0;
    }
  );
}
```

We use the `transform` step of `std::transform_reduce` to “extract” values to sum. We cannot easily use `std::reduce` as it would require us to write a reduction operation that works with two `OrderRecord` objects.

Let’s test.