Counting Elements

This lesson discusses in detail, an algorithm that counts the number of elements in a container.

WE'LL COVER THE FOLLOWING

- Introduction
- Diagram
- Implementation
  - Try It Out
- More Examples

Introduction

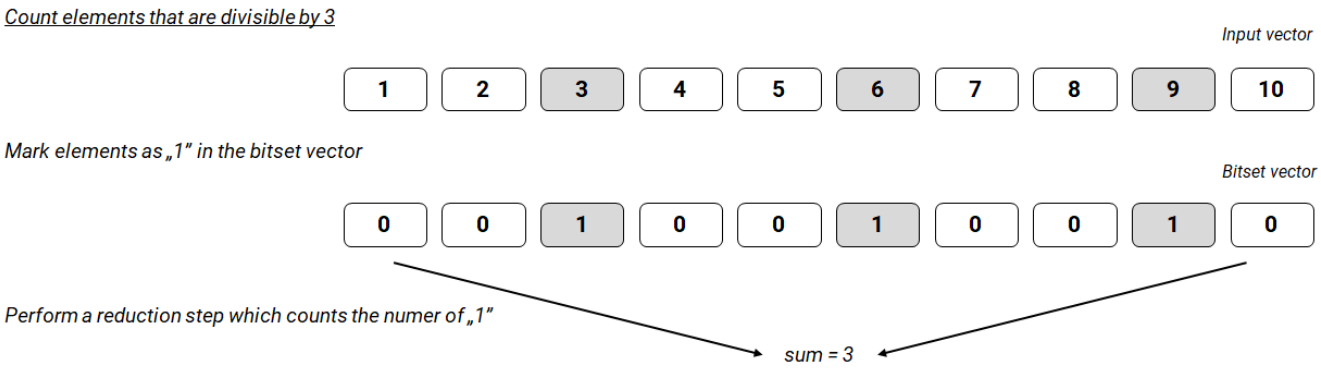
To gain some practice, let's build an algorithm that counts the number of elements in a container. Our algorithm will be a version of another standard algorithm `count_if`.

The main idea is to use `transform_reduce` - a new "fused" algorithm. It first applies some unary function over an element and then performs a reduce operation.

To get the count of elements that satisfy some predicate, we can firstly filter each element (transform). We return 1 if the element passes the filter and 0 otherwise. Then, in the reduction step, we count how many elements returned 1.

Diagram

Here's a diagram that illustrates the algorithm for a simple case:



- The first step is to perform the `transform` step in `transform_reduce` algorithm. We return `1` for matching elements and `0` otherwise.
- Then the `reduce` step is used to compute the sum of all `1`. We have three values that satisfy the condition, so the output is `3`.

Implementation

And here's the code:

```
template <typename Policy, typename Iter, typename Func>
std::size_t CountIf(Policy policy, Iter first, Iter last, Func predicate) {
    return std::transform_reduce(policy,
        first,
        last,
        std::size_t(0),
        std::plus<std::size_t>{},
        [&predicate](const Iter::value_type& v) {
            return predicate(v) ? 1 : 0;
        }
    );
}
```

We can run it on the following test containers:

```
int main()
{
  std::vector<int> v(100);
  std::iota(v.begin(), v.end(), 0);
  auto NumEven = CountIf(std::execution::par, v.begin(), v.end(),
                         [](int i) {
                           return i % 2 == 0;
                         }
                        );
  std::cout << NumEven << '\n';
}
```

To get number of spaces in a string:

```
int main()
{
  std::string_view sv = "Hello   Programming   World";
  auto NumSpaces = CountIf(std::execution::seq, sv.begin(), sv.end(),
                           [](char ch) {
                             return ch == ' ';
                           });
  std::cout << NumSpaces << '\n';
}
```

Or even on a map:

```
int main()
{
  std::map<std::string, int> CityAndPopulation{
    {"Cracow", 765000},
    {"Warsaw", 1745000},
    {"London", 10313307},
    {"New York", 18593220},
    {"San Diego", 3107034}
  };
  auto NumCitiesLargerThanMillion = CountIf(std::execution::seq,
                                            CityAndPopulation.begin(), CityAndPopulation.end(
                                            [](const std::pair<const std::string, int>& p) {
                                              return p.second > 1000000;
                                            });
  std::cout << CitiesLargerThanMillion << '\n';
}
```

The example uses simple test data and to have good performance over the sequential version the size of data would have to be significantly increased. For example, the cities and their population could be loaded from a database.

Try It Out

Here is the complete code for you to execute:

input.cpp

simpleperf.h

```
#include <algorithm>
#include <execution>
#include <iostream>
#include <map>
#include <numeric>
#include <string_view>
#include <vector>
```

```
#include "simpleperf.h"

template <typename Policy, typename Iter, typename Func>
std::size_t CountIf(Policy policy, Iter first, Iter last, Func predicate) {
        return std::transform_reduce(policy,
                first,
                last,
                std::size_t(0),
                std::plus<std::size_t>{},
                [&predicate](const auto& v) {
                    return predicate(v) ? 1 : 0;
            }
    );
}

int main(int argc, const char** argv) {
        const size_t vecSize = argc > 1 ? atoi(argv[1]) : 6000000;
        std::cout << vecSize << '\n';
        std::vector<int> vec(vecSize);

        std::iota(vec.begin(), vec.end(), 0);

        RunAndMeasure("CountIf seq", [&vec] {
                return CountIf(std::execution::seq, vec.begin(), vec.end(), [](int i) {retur
        });

        RunAndMeasure("CountIf par", [&vec] {
                return CountIf(std::execution::par, vec.begin(), vec.end(), [](int i) {retur
        });

        RunAndMeasure("std::count_if par", [&vec] {
                return std::count_if(std::execution::par, vec.begin(), vec.end(), [](int i)
        });

        ////////////////////////////////////
        // vector
        std::vector<int> v(100);
        std::iota(v.begin(), v.end(), 0);
        auto NumEven = CountIf(std::execution::par, v.begin(), v.end(),
                [](int i) { return i % 2 == 0; }
        );
        std::cout << NumEven << '\n';

        ////////////////////////////////////
        // string_view
        std::string_view sv = "Hello   Programming   Word";
        auto NumSpaces = CountIf(std::execution::par, sv.begin(), sv.end(),
                [](char ch) { return ch == ' '; }
        );
        std::cout << NumSpaces << '\n';

        ////////////////////////////////////
        // map
        std::map<std::string, int> CityAndPopulation {
                {"Cracow", 765000},
                {"Warsaw", 1745000},
                {"London", 10313307},
                {"New York", 18593220},
                {"San Diego", 3107034}
        };

        auto NumCitiesLargerThanMillion = CountIf(std::execution::par,
```

```
                CityAndPopulation.begin(), CityAndPopulation.end(),
                [](const std::pair<const std::string, int>& p) {
                    return p.second > 1000000;
        }
        );
        std::cout << NumCitiesLargerThanMillion << '\n';

    return 0;
}
```

More Examples

Here's a list of a few other ideas where parallel algorithms could be beneficial:

- statistics - calculating various maths properties for a set of data
- processing CSV records line by line in parallel
- parsing files in parallel - one file per thread, or chunks of a file per thread
- computing summed-area tables
- parallel matrix operations
- parallel dot product

You can find a few more examples in the following articles:

- The Amazing Performance of C++17 Parallel Algorithms, is it Possible?
- How to Boost Performance with Intel Parallel STL and C++17 Parallel Algorithms
- Examples of Parallel Algorithms From C++17
- Parallel STL And Filesystem: Files Word Count Example

Well now that you've reached the end of the chapter, the next lesson will provide you with a summary of all the major concepts.