Scan

Scan operations are useful when working with prefix sums.

`exclusive_scan`: computes the exclusive prefix sum using a binary operation

- Behaves similar to std::reduce, but provides a range of all prefix sums
- excludes the last element in each iteration

```
OutIt exclusive_scan(InpIt first, InpIt last, OutIt first, T init)
FwdIt2 exclusive_scan(ExePol pol, FwdIt first, FWdIt last, FwdIt2 first2, T init)

OutIt exclusive_scan(InpIt first, InpIt last, OutIt first, T init, BiFun fun)
FwdIt2 exclusive_scan(ExePol pol, FwdIt first, FwdIt last, FwdIt2 first2, T init, BiFun fun)
```

`inclusive_scan`: computes the inclusive prefix sum using a binary operation

- Behaves similar to std::reduce, but provides a range of all prefix sums
- includes the last element in each iteration

```
OutIt inclusive_scan(InpIt first, InpIt last, OutIt first2)
FwdIt2 inclusive_scan(ExePol pol, FwdIt first, FwdIt last, FwdIt2 first2)

OutIt inclusive_scan(InpIt first, InpIt last, OutIt first, BiFun fun)
FwdIt2 inclusive_scan(ExePol pol, FwdIt first, FwdIt last, FwdIt2 first2, BiFun fun)

OutIt inclusive_scan(InpIt first, InpIt last, OutIt firs2t, BiFun fun, T init)
FwdIt2 inclusive_scan(ExePol pol, FwdIt first, FwdIt last, FwdIt2 first2, BiFun fun, T init)
```

`transform_exclusive_scan`: first transforms each element and then computes the exclusive prefix sums

```
OutIt transform_exclusive_scan(InpIt first, InpIt last, OutIt first2, T init, BiFun fun, UnFu
FwdIt2 transform_exclusive_scan(ExePol pol, FwdIt first, FwdIt last, FwdIt2 first2, T init, B
```

`transform_inclusive_scan`: first transforms each element of the input range and then computes the inclusive prefix sums

```
OutIt transform_inclusive_scan(InpIt first, InpIt last, OutIt first2, BiFun fun, UnFun fun2)
FwdIt2 transform_inclusive_scan(ExePol pol, FwdIt first, FwdIt last, FwdIt first2, BiFun fun,

OutIt transform_inclusive_scan(InpIt first, InpIt last, OutIt first2, BiFun fun, UnFun fun2,
FwdIt2 transform_inclusive_scan(ExePol pol, FwdIt first, FwdIt last, FwdIt first2, BiFun fun,
```

The following example illustrates the usage of the six algorithms using the parallel execution policy.

```
#include <iostream>
#include <numeric>
#include <string>
#include <vector>


int main(){

  std::cout << std::endl;

  // for_each_n

  std::vector<int> intVec{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  std::for_each_n(std::execution::par,
                  intVec.begin(), 5, [](int& arg){ arg *= arg; });

  std::cout << "for_each_n: ";
  for (auto v: intVec) std::cout << v << " ";
  std::cout << "\n\n";

  // exclusive_scan and inclusive_scan
  std::vector<int> resVec{1, 2, 3, 4, 5, 6, 7, 8, 9};
  std::exclusive_scan(std::execution::par,
                      resVec.begin(), resVec.end(), resVec.begin(), 1,
                      [](int fir, int sec){ return fir * sec; });

  std::cout << "exclusive_scan: ";
  for (auto v: resVec) std::cout << v << " ";
  std::cout << std::endl;

  std::vector<int> resVec2{1, 2, 3, 4, 5, 6, 7, 8, 9};

  std::inclusive_scan(std::execution::par,
                      resVec2.begin(), resVec2.end(), resVec2.begin(),
                      [](int fir, int sec){ return fir * sec; }, 1);

  std::cout << "inclusive_scan: ";
  for (auto v: resVec2) std::cout << v << " ";
  std::cout << "\n\n";

  // transform_exclusive_scan and transform_inclusive_scan
  std::vector<int> resVec3{1, 2, 3, 4, 5, 6, 7, 8, 9};
  std::vector<int> resVec4(resVec3.size());
  std::transform_exclusive_scan(std::execution::par,
                                resVec3.begin(), resVec3.end(),
                                resVec4.begin(), 0,
                                [](int fir, int sec){ return fir + sec; },
                                [](int arg){ return arg *= arg; });
```

```
  std::cout << "transform_exclusive_scan: ";
  for (auto v: resVec4) std::cout << v << " ";
  std::cout << std::endl;


  std::vector<std::string> strVec{"Only", "for", "testing", "purpose"};
  std::vector<int> resVec5(strVec.size());

  std::transform_inclusive_scan(std::execution::par,
                                strVec.begin(), strVec.end(),
                                resVec5.begin(), 0,
                                [](auto fir, auto sec){ return fir + sec; },
                                [](auto s){ return s.length(); });

  std::cout << "transform_inclusive_scan: ";
  for (auto v: resVec5) std::cout << v << " ";
  std::cout << "\n\n";

  // reduce and transform_reduce
  std::vector<std::string> strVec2{"Only", "for", "testing", "purpose"};

  std::string res = std::reduce(std::execution::par,
                    strVec2.begin() + 1, strVec2.end(), strVec2[0],
                    [](auto fir, auto sec){ return fir + ":" + sec; });

  std::cout << "reduce: " << res << std::endl;


  std::size_t res7 = std::parallel::transform_reduce(std::execution::par,
                      strVec2.begin(), strVec2.end(), 0,
                      [](std::size_t a, std::size_t b){ return a + b; },
                      [](std::string s){ return s.length(); });

  std::cout << "transform_reduce: " << res7 << std::endl;

  std::cout << std::endl;

}
```

The new algorithms