Algorithm Update

This section lists all the algorithms that can handle execution policy parameter. It further looks at the usage of the Run and Measure function.

WE'LL COVER THE FOLLOWING

- Execution Policy Parameter
- New Algorithms

Execution Policy Parameter

The execution policy parameter was added to most of the existing algorithms.

Here's the list:

`adjacent_difference`	`inplace_merge`	`replace_copy`
`adjacent_find`	`is_heap`	`replace_copy_if`
`all_of`	`is_heap_until`	`replace_if`
`any_of`	`is_partitioned`	`reverse`
`copy`	`is_sorted`	`reverse_copy`
`copy_if`	`is_sorted`	`rotate`
`copy_n`	`is_sorted_until`	`rotate_copy`
`count`	`lexicographical_compare`	`search`

- `count_if`
- `equal`
- `exclusive_scan`
- `fill`
- `fill_n`
- `find`
- `find_end`
- `find_first_of`
- `find_if`
- `find_if_not`
- `for_each`
- `for_each_n`
- `generate`
- `generate_n`
- `includes`
- `inclusive_scan`
- `max_element`
- `merge`
- `min_element`
- `minmax_element`
- `mismatch`
- `move`
- `none_of`
- `nth_element`
- `partial_sort`
- `partial_sort_copy`
- `partition`
- `partition_copy`
- `remove`
- `remove_copy`
- `remove_copy_if`
- `remove_if`
- `search_n`
- `set_difference`
- `set_intersection`
- `set_symmetric_difference`
- `set_union`
- `sort`
- `stable_partition`
- `stable_sort`
- `swap_ranges`
- `transform`
- `transform_exclusive_scan`
- `transform_inclusive_scan`
- `transform_reduce`
- `uninitialized_copy`
- `uninitialized_copy_n`
- `uninitialized_fill`

`inner_product`	`replace`	`uninitialized_fill_n`
	`unique`	`unique_copy`

New Algorithms

To fully support new parallel execution patterns The Standard Library was also equipped with a set of new algorithms:

Algorithm	Description
`for_each`	similar to `for_each` except returns `void`
`for_each_n`	applies a function object to the first `n` elements of a sequence
`reduce`	similar to `accumulate`, except out of order execution to allow parallelism
`transform_reduce`	transforms the input elements using a unary operation, then reduces the output out of order
`exclusive_scan`	parallel version of `partial_sum`, excludes the i-th input element from the i-th sum, out of order execution to allow parallelism
`inclusive_scan`	parallel version of `partial_sum`, includes the i-th input element in the i-th sum, out of order execution to allow parallelism
`transform_exclusive_scan`	applies a functor, then calculates

`transform_exclusive_scan`	exclusive scan
`transform_inclusive_scan`	applies a functor, then calculates inclusive scan

The new algorithms form three groups: `for_each`, `reduce` and then `scan`, plus their alternatives.

With `reduce` and `scan` you also get “fused” versions like `transform_reduce`. These compositions should give you much better performance than using two separate steps - because the cost of parallel execution setup is smaller and also you have less one loop traversals.

The new algorithms also provide overloads without the execution policy parameter so that you can use them in a standard serial version.

In the next lesson, you’ll find a description of each group.