Parallel STL – summary

Execution policies

- `sequenced_policy` (`std::execution::seq`): execution may not be parallelized
- `parallel_policy` (`std::execution::par`): execution may be parallelized
- `parallel_unsequenced_policy` (`std::execution::par_unseq`): execution may be parallelized, element access functions may be invoked in any order, the execution may be vectorized
- `unsequenced_policy` (`std::execution::unseq`, since C++20): execution may be vectorized

Performance summary

The operation used for this example is a `std::transform` with a relatively complex task executed on a `std::vector` of integers. Results show the number of iterations per second.

Data size	x86 `seq`	x86 `par`	arm64 `seq`	arm64 `par`
32	354k (100%)	74.0k (**21%**)	1.81M (100%)	242k (**13%**)
128	66.3k (100%)	33.2k (**50%**)	436k (100%)	86.9k (**20%**)
512	18.7k (100%)	16.7k (**89%**)	96.5k (100%)	45.1k (**47%**)
2048	4.70k (100%)	8.12k (**173%**)	17.7k (100%)	32.9k (**186%**)
8192	1.10k (100%)	2.92k (**265%**)	4.20k (100%)	14.7k (**350%**)
32768	277 (100%)	835 (**301%**)	993 (100%)	4.53k (**456%**)
131072	65.5 (100%)	222 (**339%**)	242 (100%)	1.15k (**475%**)

Compilers and HW used for testing: x86-64 – MSVC 2019 (`/O2`) on Intel i7-8650U (4 cores × 2 threads), ARM64 – GCC 10.2 (`-O3`) on Apple M1 (4 low power + 4 high power cores).

Basic observations

- Data size and platform should be taken into consideration when deciding on an execution policy. Spawning threads itself has a cost, which may be different between platforms.
- Manually setting the `unseq` policy seemed to provide very little (if any) profit. This is due to the compiler itself optimizng code, when `-O0` option is used to disable optimizations, the gain from this policy is noticeably higher.