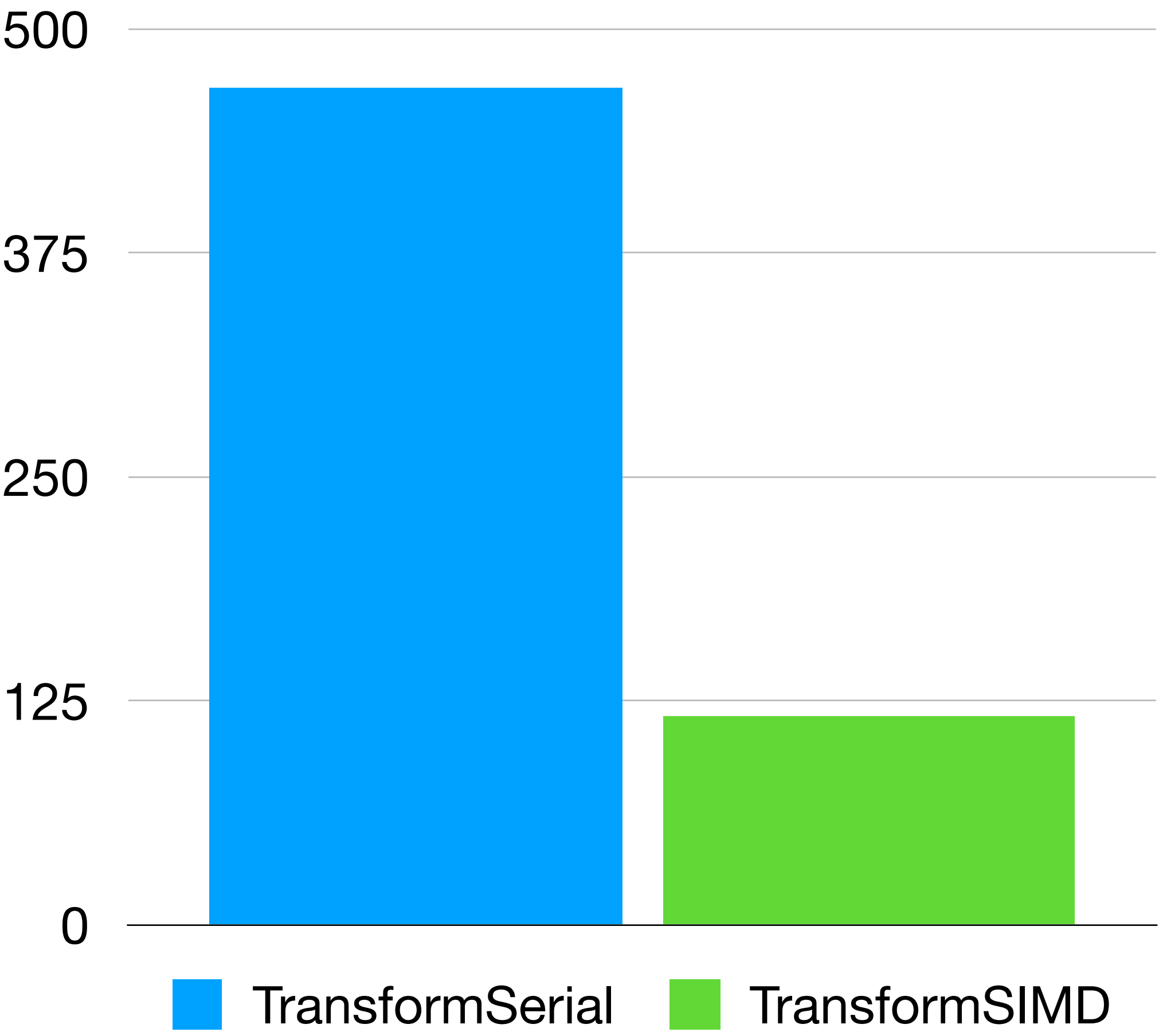



```
// Return 256 random floats +/-1.0f
static const std::vector<float>& getBufferData();

static void TransformXXX(benchmark::State& state)
{
    const auto& v = getBufferData();
    auto copy = v;

    for (auto _ : state)
    {
        std::transform (v.begin(), v.end(), copy.begin(), copy.end(),
                        [] (auto v1, auto v2) { return v1 + v2; });
    }
}

BENCHMARK(TransformXXX);
```



4x Faster

https://quick-bench.com/q/p0J2y_b8qUaJ560iDSBL_Sgjl

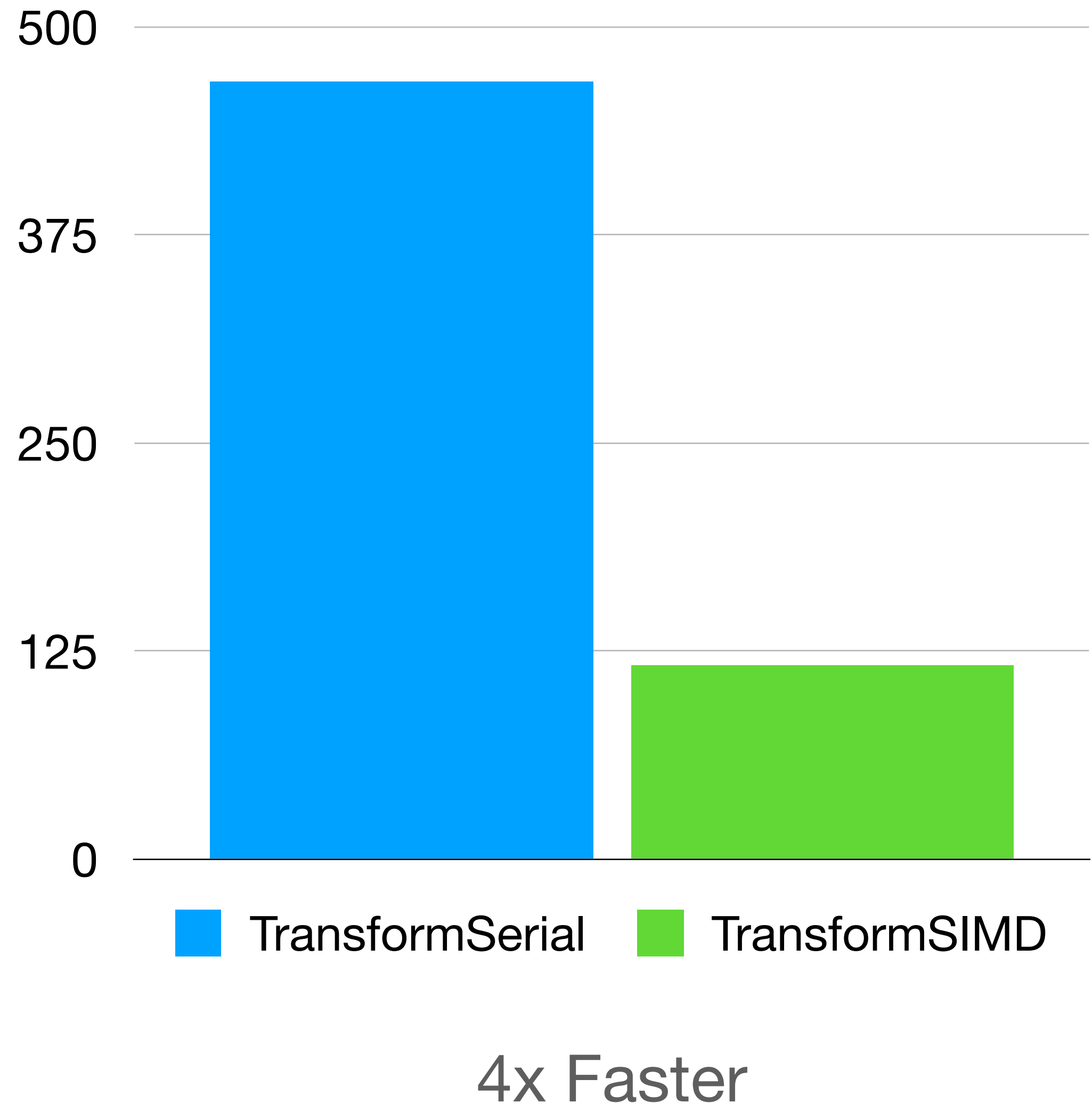
3

5

```
// Return 256 random floats +/-1.0f
static const std::vector<float>& getBufferData();

static void TransformXXX(benchmark::State& state)
{
    const auto& v = getBufferData();
    auto copy = v;

    for (auto _ : state)
    {
        std::transform (v.begin(), v.end(), copy.begin(), copy.end(),
                        [] (auto v1, auto v2) { return v1 + v2; });
    }
}
BENCHMARK(TransformXXX);
```




```

#include <algorithm>

const auto v = { 566, ... };

static void MinAndMaxElement(benchmark::State& state)
{
    for (auto _ : state)
    {
        const auto min = std::min_element (begin(v), end(v));
        const auto max = std::max_element (begin(v), end(v));

        // Make sure the variable is not optimized away by compiler
        benchmark::DoNotOptimize(min);
        benchmark::DoNotOptimize(max);
    }
}
// Register the function as a benchmark
BENCHMARK(MinAndMaxElement);

static void MinMaxElement(benchmark::State& state)
{
    for (auto _ : state)
    {
        const auto [min, max] = std::minmax_element (begin(v), end(v));

        // Make sure the variable is not optimized away by compiler
        benchmark::DoNotOptimize(min);
        benchmark::DoNotOptimize(max);
    }
}
// Register the function as a benchmark
BENCHMARK(MinMaxElement);

```