Throwing Tools at Ranges

Tina Ulbrich

2023

Throwing Tools at Ranges

Tina Ulbrich – ROSEN Technology and Research Center

C++ on Sea 2023

Outline

- Setup
- Code example
- Google Benchmark
- Cachegrind
- Visual Studio Profiler
- Optimizing
- Conclusion

Setup

Setup

Laptop

Processor	Intel Core i3-6157U CPU @ 2,40GHz
Cores	2
Logical Processors	4
L1-Cache	128 KB
L2-Cache	512 KB
L3-Cache	3 MB
RAM	6 GB
OS	Windows 10

PC

Processor	13th Gen Intel Core i7-13700KF @ 3,4GHz
Cores	16
Logical Processors	24
L1-Cache	1,4 MB
L2-Cache	24 MB
L3-Cache	30 MB
RAM	64 GB
OS	Windows 11

Code

GitHub

Function 1, C++23

```
auto random_23_1(const std::vector<double>& rng)
{
    return ranges::views::cartesian_product(
        rng,
        ranges::views::iota(1, 5),
        rng | ranges::views::transform([](const auto e) { return e * e + 2; }))
        | ranges::views::transform([](const auto t)
        {
            const auto& [a, b, c] = t;
            return a * b + 2 * c;
        });
}
```

Function 1, C++17

```cpp
auto random_17_1(const std::vector<double>& rng)
{
    auto out = std::vector<double>();
    out.reserve(rng.size() * rng.size() * 4);
    for (const auto e1 : rng)
    {
        for (const auto e2 : std::array{ 1, 2, 3, 4 })
        {
            for (const auto e3 : rng)
            {
                const auto e = e1 * e2 + 2 * (e3 * e3 + 2);
                out.push_back(e);
            }
        }
    }

    return out;
}
```

Function 1, C – style C++

```
double* random_c_style_1(double* rng, int n)
{
    double* out = new double[n * n * 4];
    int out_idx = 0;
    for (int i = 0; i < n; ++i)
    {
        for (int j = 1; j < 5; ++j)
        {
            for (int k = 0; k < n; ++k)
            {
                out[out_idx] = rng[i] * j + 2 * (rng[k] * rng[k] + 2);
                ++out_idx;
            }
        }
    }

    return out;
}
```

Function 2, C++23

```
auto random_23_2(auto& rng1)
{
    return ranges::views::zip_with(
        [](const auto a, const auto b) { return a / static_cast<double>(b); },
        rng1,
        ranges::views::iota(10)
    );
}
```

Function 2, C++17

```
auto random_17_2(const std::vector<double>& rng)
{
    auto out = std::vector<double>(rng.size());
    int i = 10;
    std::transform(rng.begin(), rng.end(), out.begin(), [&](const auto e)
    {
        return e / static_cast<double>(i++);
    });

    return out;
}
```

Function 2, C – style C++

```
double* random_c_style_2(double* rng, int n)
{
    double* out = new double[n];
    for (int i = 0; i < n; ++i)
    {
        out[i] = rng[i] / double(i + 10);
    }

    return out;
}
```

Function 3, C++23

```
auto random_23_3(auto& rng1, auto& rng2)
{
    auto tmp1 = rng1 | ranges::views::partial_sum;
    auto tmp2 = rng2
        | ranges::views::sliding(2)
        | ranges::views::transform([](const auto subrng)
        {
            return 3.0 * subrng[1] - 2.0 * subrng[0];
        });

    return ranges::inner_product(tmp1 | ranges::views::drop(1), tmp2, 0.0);
}
```

Function 3, C++17

```
auto random_17_3(const std::vector<double>& rng1, const std::vector<double>& rng2)
{
    auto tmp1 = std::vector<double>(rng1.size());
    std::partial_sum(rng1.begin(), rng1.end(), tmp1.begin());

    auto tmp2 = std::vector<double>(rng2.size());
    std::adjacent_difference(rng2.begin(), rng2.end(), tmp2.begin(),
                             [](const auto left, const auto right)
    {
        return 3.0 * left - 2.0 * right;
    });

    return std::inner_product(tmp1.begin() +1, tmp1.end(), tmp2.begin() +1, 0.0);
}
```

Function 3, C – style C++

```
double random_c_style_3(double* rng1, double* rng2, int n)
{
    double* tmp1 = new double[n];
    tmp1[0] = rng1[0];
    for (int i = 1; i < n; ++i)
    {
        tmp1[i] = tmp1[i - 1] + rng1[i];
    }

    double* tmp2 = new double[n - 1];
    for (int i = 0; i < n - 1; ++i)
    {
        tmp2[i] = 3.0 * rng2[i + 1] - 2.0 * rng2[i];
    }
...
```

Function 3, C – style C++

```
...
    double sum = 0.0;
    for (int i = 0; i < n - 1; ++i)
    {
        sum += tmp1[i + 1] * tmp2[i];
    }

    delete[] tmp1;
    delete[] tmp2;
    return sum;
}
```

Function 4, C++23

```
double random_23(const std::vector<double>& rng)
{
    auto rng1 = random_23_1(rng);
    auto rng2 = random_23_2(rng1);

    return random_23_3(rng1, rng2);
}
```

Function 4, C++17

```
double random_17(const std::vector<double>& rng)
{
    const auto rng1 = random_17_1(rng);
    const auto rng2 = random_17_2(rng1);

    return random_17_3(rng1, rng2);
}
```

Function 4, C – style C++

```
double random_c_style(double* rng, int n)
{
    auto tmp1 = random_c_style_1(rng, n);
    auto tmp2 = random_c_style_2(tmp1, n * n * 4);
    auto out = random_c_style_3(tmp1, tmp2, n * n * 4);

    delete[] tmp1;
    delete[] tmp2;

    return out;
}
```

Google Benchmark

Google Benchmark

```cpp
static void cpp_17_benchmark_optimized(benchmark::State& state)
{
    const auto rng = std::vector(data.begin(), data.end());
    for (auto _ : state)
    {
         benchmark::DoNotOptimize(random_17_optimized(rng));
    }
}

BENCHMARK(cpp_17_benchmark)->Unit(benchmark::kMillisecond);

BENCHMARK_MAIN();
```

Google Benchmark

```
Run on (24 X 3450.41 MHz CPU s)
CPU Caches:
  L1 Data 48 KiB (x12)
  L1 Instruction 32 KiB (x12)
  L2 Unified 2048 KiB (x12)
  L3 Unified 30720 KiB (x1)
----------------------------------------------------------------------
Benchmark                            Time             CPU   Iterations
----------------------------------------------------------------------
c_style_benchmark                 25.3 ms         19.8 ms           34
cpp_17_benchmark                  35.4 ms         23.8 ms           50
cpp_23_benchmark                 317.0 ms        223.0 ms            4
```

Cachegrind

Cachegrind, C – style C++

```
I   refs:       92,364,114
I1  misses:          2,025
LLi misses:          1,956
I1  miss rate:        0.00%
LLi miss rate:        0.00%

D   refs:       26,758,586  ( 16,556,648 rd   +  10,201,938 wr)
D1  misses:      4,515,098  (  2,512,546 rd   +   2,002,552 wr)
LLd misses:      4,509,814  (  2,508,034 rd   +   2,001,780 wr)
D1  miss rate:        16.9% (       15.2%     +        19.6%  )
LLd miss rate:        16.9% (       15.1%     +        19.6%  )

LL refs:         4,517,123  (  2,514,571 rd   +   2,002,552 wr)
LL misses:       4,511,770  (  2,509,990 rd   +   2,001,780 wr)
LL miss rate:          3.8% (        2.3%     +        19.6%  )

Branches:       11,334,889  ( 11,329,459 cond +       5,430 ind)
Mispredicts:        18,731  (     18,163 cond +         568 ind)
Mispred rate:          0.2% (        0.2%     +        10.5%   )
```

Cachegrind, C++17

```
I   refs:      252,376,633
I1  misses:          2,044
LLi misses:          1,976
I1  miss rate:        0.00%
LLi miss rate:        0.00%

D   refs:      144,802,262  ( 26,594,447 rd   + 118,207,815 wr)
D1  misses:      6,015,098  (  2,512,549 rd   +   3,502,549 wr)
LLd misses:      5,979,557  (  2,508,074 rd   +   3,471,483 wr)
D1  miss rate:         4.2% (        9.4%     +         3.0%  )
LLd miss rate:         4.1% (        9.4%     +         2.9%  )

LL refs:         6,017,142  (  2,514,593 rd   +   3,502,549 wr)
LL misses:       5,981,533  (  2,510,050 rd   +   3,471,483 wr)
LL miss rate:          1.5% (        0.9%     +         2.9%  )

Branches:      115,330,825  (115,325,399 cond +       5,426 ind)
Mispredicts:        18,725  (     18,152 cond +         573 ind)
Mispred rate:          0.0% (        0.0%     +        10.6%   )
```

Cachegrind, C++23

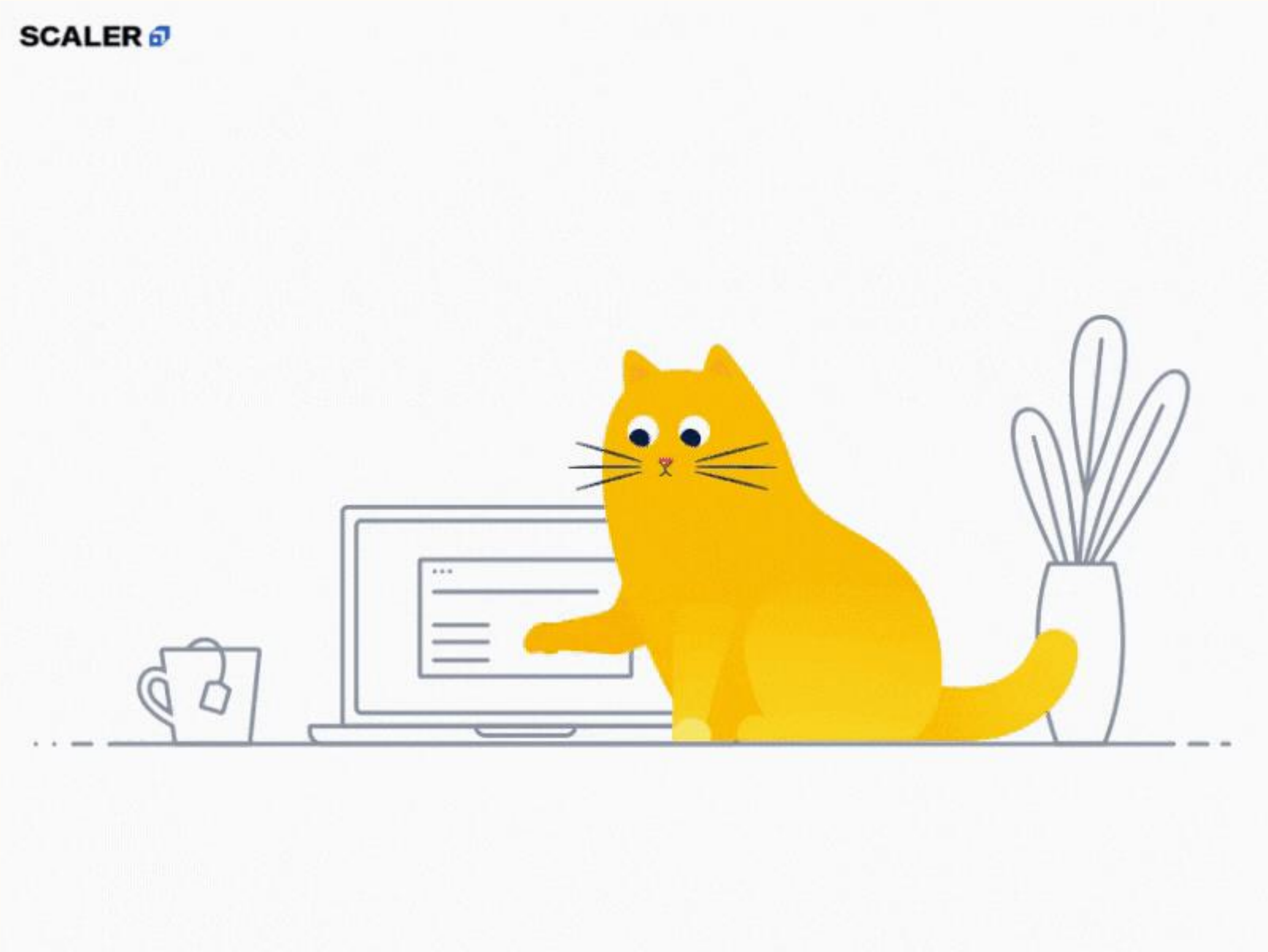
```
I   refs:      542,456,846
I1  misses:          2,036
LLi misses:          1,936
I1  miss rate:        0.00%
LLi miss rate:        0.00%

D   refs:      232,780,702  (112,575,063 rd   + 120,205,639 wr)
D1  misses:         14,845  (     12,349 rd   +       2,496 wr)
LLd misses:          9,425  (      7,708 rd   +       1,717 wr)
D1  miss rate:         0.0% (        0.0%     +         0.0%  )
LLd miss rate:         0.0% (        0.0%     +         0.0%  )

LL refs:            16,881  (     14,385 rd   +       2,496 wr)
LL misses:          11,361  (      9,644 rd   +       1,717 wr)
LL miss rate:          0.0% (        0.0%     +         0.0%  )

Branches:       36,340,555  ( 36,335,152 cond +       5,403 ind)
Mispredicts:        19,671  (     19,108 cond +         563 ind)
Mispred rate:          0.1% (        0.1%     +        10.4%   )
```

Cachegrind compare

	C-Style	C++17	C++23
I refs:	92,364,114	252,376,633	542,456,846
I1 misses:	2,025	2,044	2,036
LLi misses:	1,956	1,976	1,936
I1 miss rate:	0.00%	0.00%	0.00%
LLi miss rate:	0.00%	0.00%	0.00%
D refs:	26,758,586	144,802,262	232,780,702
D1 misses:	4,515,098	6,015,098	14,845
LLd misses:	4,509,814	5,979,557	9,425
D1 miss rate:	16.9%	4.2%	0.0%
LLd miss rate:	16.9%	4.1%	0.0%
LL refs:	4,517,123	6,017,142	16,881
LL misses:	4,511,770	5,981,533	11,361
LL miss rate:	3.8%	1.5%	0.0%
Branches:	11,334,889	115,330,825	36,340,555
Mispredicts:	18,731	18,725	19,671
Mispred rate:	0.2%	0.0%	0.1%

Profiling and Optimizing

Visual Studio Profiler



Function 3, C – style C++

```
double random_c_style_3(double* rng1, double* rng2, int n) {
    double* tmp1 = new double[n];
    tmp1[0] = rng1[0];
    for (int i = 1; i < n; ++i) {
        tmp1[i] = tmp1[i - 1] + rng1[i];
    }

    double* tmp2 = new double[n - 1];
    for (int i = 0; i < n - 1; ++i) {
        tmp2[i] = 3.0 * rng2[i + 1] - 2.0 * rng2[i];
    }

    double sum = 0.0;
    for (int i = 0; i < n - 1; ++i) {
        sum += tmp1[i + 1] * tmp2[i];
    }

    delete[] tmp1;
    delete[] tmp2;
    return sum;
}
```

Optimizing, C – style C++

```
double random_c_style_3(double* rng1, double* rng2, int n)
{
    double partial_sum = rng1[0];
    double sum = 0.0;
    for (int i = 1; i < n; ++i)
    {
        auto partial_sum += rng1[i];
        sum += partial_sum * (3.0 * rng2[i] - 2.0 * rng2[i - 1]);
    }

    return sum;
}
```

Function 3, C – style C++

```
double random_c_style_3(double* rng1, double* rng2, int n) {
    double* tmp1 = new double[n];
    tmp1[0] = rng1[0];
    for (int i = 1; i < n; ++i) {
        tmp1[i] = tmp1[i - 1] + rng1[i];
    }

    double* tmp2 = new double[n - 1];
    for (int i = 0; i < n - 1; ++i) {
        tmp2[i] = 3.0 * rng2[i + 1] - 2.0 * rng2[i];
    }

    double sum = 0.0;
    for (int i = 0; i < n - 1; ++i) {
        sum += tmp1[i + 1] * tmp2[i];
    }

    delete[] tmp1;
    delete[] tmp2;
    return sum;
}
```

Optimizing, C – style C++

```
double random_c_style_3(double* rng1, double* rng2, int n)
{
    double partial_sum = rng1[0];
    double sum = 0.0;
    for (int i = 1; i < n; ++i)
    {
        auto partial_sum += rng1[i];
        sum += partial_sum * (3.0 * rng2[i] - 2.0 * rng2[i - 1]);
    }

    return sum;
}
```

Function 3, C – style C++

```
double random_c_style_3(double* rng1, double* rng2, int n) {
    double* tmp1 = new double[n];
    tmp1[0] = rng1[0];
    for (int i = 1; i < n; ++i) {
        tmp1[i] = tmp1[i - 1] + rng1[i];
    }

    double* tmp2 = new double[n - 1];
    for (int i = 0; i < n - 1; ++i) {
        tmp2[i] = 3.0 * rng2[i + 1] - 2.0 * rng2[i];
    }

    double sum = 0.0;
    for (int i = 0; i < n - 1; ++i) {
        sum += tmp1[i + 1] * tmp2[i];
    }

    delete[] tmp1;
    delete[] tmp2;
    return sum;
}
```

Optimizing, C – style C++

```
double random_c_style_3(double* rng1, double* rng2, int n)
{
    double partial_sum = rng1[0];
    double sum = 0.0;
    for (int i = 1; i < n; ++i)
    {
        auto partial_sum += rng1[i];
        sum += partial_sum * (3.0 * rng2[i] - 2.0 * rng2[i - 1]);
    }

    return sum;
}
```

Function 3, C++17

```
auto random_17_3(const std::vector<double>& rng1, const std::vector<double>& rng2)
{
    auto tmp1 = std::vector<double>(rng1.size());
    std::partial_sum(rng1.begin(), rng1.end(), tmp1.begin());

    auto tmp2 = std::vector<double>(rng2.size());
    std::adjacent_difference(rng2.begin(), rng2.end(), tmp2.begin(),
                             [](const auto left, const auto right)
    {
        return 3.0 * left - 2.0 * right;
    });

    return std::inner_product(tmp1.begin() +1, tmp1.end(), tmp2.begin() +1, 0.0);
}
```

Optimizing, C++17

```
auto random_17_3(const std::vector<double>& rng1,
                 const std::vector<double>& rng2)
{
    double partial_sum = rng1[0];
    size_t i = 0;
    return std::accumulate(rng1.begin() + 1, rng1.end(), 0.0,
        [&](const auto sum, const auto cur)
        {
            partial_sum += cur;
            ++i;
            return sum + prev * (3.0 * rng2[i] - 2.0 * rng2[i - 1]);
        });
}
```

Optimizing, C++23

```cpp
double random_23(const std::vector<double>& rng)
{
    auto rng1 = random_23_1(rng);
    auto rng2 = random_23_2(rng1);
    return random_23_3(rng1, rng2);
}
```

```cpp
auto random_23_1(const std::vector<double>& rng)
{
    return ranges::views::cartesian_product(
        rng,
        ranges::views::iota(1, 5),
        rng | ranges::views::transform([](const auto e) { return e * e + 2; }))
        | ranges::views::transform([](const auto t)
        {
            const auto& [a, b, c] = t;
            return a * b + 2 * c;
        });
}
```

Optimizing, C++23

```
double random_23(const std::vector<double>& rng)
{
    auto rng1 = random_23_1(rng);
    auto rng2 = random_23_2(rng1);
    return random_23_3(rng1, rng2);
}
```

```
auto random_23_1(const std::vector<double>& rng)
{
    return ranges::views::cartesian_product(
        rng,
        ranges::views::iota(1, 5),
        rng | ranges::views::transform([](const auto e) { return e * e + 2; }))
        | ranges::views::transform([](const auto t)
        {
            const auto& [a, b, c] = t;
            return a * b + 2 * c;
        }) | ranges::to<std::vector>;
}
```

Google Benchmark

Optimized Benchmarks

```
Run on (24 X 3450.41 MHz CPU s)
CPU Caches:
  L1 Data 48 KiB (x12)
  L1 Instruction 32 KiB (x12)
  L2 Unified 2048 KiB (x12)
  L3 Unified 30720 KiB (x1)
----------------------------------------------------------------------
Benchmark                                Time             CPU   Iterations
----------------------------------------------------------------------
c_style_benchmark                     25.3 ms         19.8 ms           34
cpp_17_benchmark                      35.4 ms         23.8 ms           50
cpp_23_benchmark                     317.0 ms        223.0 ms            4

c_style_benchmark_optimized           13.2 ms         9.58 ms           75
cpp_17_benchmark_optimized            18.2 ms         15.5 ms           90
cpp_23_benchmark_optimized            12.0 ms         6.20 ms          204
```

Cachegrind

Cachegrind, optimized C – style C++

```
I   refs:      76,367,324
I1  misses:         2,025
LLi misses:         1,956
I1  miss rate:       0.00%
LLi miss rate:       0.00%

D   refs:      16,758,297  (12,556,469 rd   + 4,201,828 wr)
D1  misses:     2,515,039  ( 1,512,505 rd   + 1,002,534 wr)
LLd misses:     2,509,755  ( 1,507,993 rd   + 1,001,762 wr)
D1  miss rate:       15.0% (      12.0%     +      23.9%  )
LLd miss rate:       15.0% (      12.0%     +      23.8%  )

LL refs:        2,517,064  ( 1,514,530 rd   + 1,002,534 wr)
LL misses:      2,511,711  ( 1,509,949 rd   + 1,001,762 wr)
LL miss rate:         2.7% (       1.7%     +      23.8%  )

Branches:       5,334,767  ( 5,329,349 cond +     5,418 ind)
Mispredicts:       18,704  (    18,136 cond +       568 ind)
Mispred rate:         0.4% (       0.3%     +      10.5%   )
```

Cachegrind, optimized C – style C++

```
I   refs:            92,364,114          76,367,324
I1  misses:               2,025               2,025
LLi misses:               1,956               1,956
I1  miss rate:             0.00%               0.00%
LLi miss rate:             0.00%               0.00%

D   refs:            26,758,586          16,758,297
D1  misses:           4,515,098           2,515,039
LLd misses:           4,509,814           2,509,755
D1  miss rate:             16.9%               15.0%
LLd miss rate:             16.9%               15.0%

LL refs:              4,517,123           2,517,064
LL misses:            4,511,770           2,511,711
LL miss rate:               3.8%                2.7%

Branches:            11,334,889           5,334,767
Mispredicts:             18,731              18,704
Mispred rate:               0.2%                0.4%
```

Cachegrind, optimized C++17

```
I   refs:      164,384,131
I1  misses:          2,033
LLi misses:          1,965
I1  miss rate:        0.00%
LLi miss rate:        0.00%

D   refs:       84,759,055  (14,557,331 rd   + 70,201,724 wr)
D1  misses:      3,515,169  ( 1,512,636 rd   +  2,002,533 wr)
LLd misses:      3,131,856  ( 1,508,160 rd   +  1,623,696 wr)
D1  miss rate:         4.1% (      10.4%     +        2.9%  )
LLd miss rate:         3.7% (      10.4%     +        2.3%  )

LL refs:         3,517,202  ( 1,514,669 rd   +  2,002,533 wr)
LL misses:       3,133,821  ( 1,510,125 rd   +  1,623,696 wr)
LL miss rate:          1.3% (       0.8%     +        2.3%  )

Branches:       72,338,768  (72,333,349 cond +      5,419 ind)
Mispredicts:        18,732  (    18,162 cond +        570 ind)
Mispred rate:          0.0% (       0.0%     +       10.5%   )
```

Cachegrind, optimized C++17

```
I   refs:           252,376,633         164,384,131
I1  misses:               2,044               2,033
LLi misses:               1,976               1,965
I1  miss rate:             0.00%               0.00%
LLi miss rate:             0.00%               0.00%

D   refs:           144,802,262          84,759,055
D1  misses:           6,015,098           3,515,169
LLd misses:           5,979,557           3,131,856
D1  miss rate:              4.2%                4.1%
LLd miss rate:              4.1%                3.7%

LL refs:              6,017,142           3,517,202
LL misses:            5,981,533           3,133,821
LL miss rate:               1.5%                1.3%

Branches:           115,330,825          72,338,768
Mispredicts:             18,725              18,732
Mispred rate:               0.0%                0.0%
```

Cachegrind, optimized C++23

```
I   refs:      150,312,446
I1  misses:          2,045
LLi misses:          1,977
I1  miss rate:        0.00%
LLi miss rate:        0.00%

D   refs:       24,756,927  (20,555,184 rd   + 4,201,743 wr)
D1  misses:      1,015,022  (   512,478 rd   +   502,544 wr)
LLd misses:        645,605  (   143,833 rd   +   501,772 wr)
D1  miss rate:         4.1% (       2.5%     +      12.0%  )
LLd miss rate:         2.6% (       0.7%     +      11.9%  )

LL refs:         1,017,067  (   514,523 rd   +   502,544 wr)
LL misses:         647,582  (   145,810 rd   +   501,772 wr)
LL miss rate:          0.4% (       0.1%     +      11.9%  )

Branches:       16,325,629  (16,320,221 cond +      5,408 ind)
Mispredicts:        18,668  (    18,102 cond +        566 ind)
Mispred rate:          0.1% (       0.1%     +      10.5%   )
```

Cachegrind, optimized C++23

```
I   refs:           542,456,846         150,312,446
I1  misses:               2,036               2,045
LLi misses:               1,936               1,977
I1  miss rate:             0.00%               0.00%
LLi miss rate:             0.00%               0.00%

D   refs:           232,780,702          24,756,927
D1  misses:              14,845           1,015,022
LLd misses:               9,425             645,605
D1  miss rate:              0.0%                4.1%
LLd miss rate:              0.0%                2.6%

LL refs:                 16,881           1,017,067
LL misses:               11,361             647,582
LL miss rate:               0.0%                0.4%

Branches:            36,340,555          16,325,629
Mispredicts:             19,671              18,668
Mispred rate:               0.1%                0.1%
```

Cachegrind compare optimized

	C-Style	C++17	C++23
I refs:	76,367,324	164,384,131	150,312,446
I1 misses:	2,025	2,033	2,045
LLi misses:	1,956	1,965	1,977
I1 miss rate:	0.00%	0.00%	0.00%
LLi miss rate:	0.00%	0.00%	0.00%
D refs:	16,758,297	84,759,055	24,756,927
D1 misses:	2,515,039	3,515,169	1,015,022
LLd misses:	2,509,755	3,131,856	645,605
D1 miss rate:	15.0%	4.1%	4.1%
LLd miss rate:	15.0%	3.7%	2.6%
LL refs:	2,517,064	3,517,202	1,017,067
LL misses:	2,511,711	3,133,821	647,582
LL miss rate:	2.7%	1.3%	0.4%
Branches:	5,334,767	72,338,768	16,325,629
Mispredicts:	18,704	18,732	18,668
Mispred rate:	0.4%	0.0%	0.1%

Next?

Conclusion

Ranges

Questions?

Tina Ulbrich

ROSEN Technology and Research Center

@_yulivee_

✉ tina_ulbrich@gmx.de

#include <C++> tinaul