

PARO 2022

Optymalizacje w C++

Prowadzący:

- Marek Jarycki
- Eryk Kubański

Współautorzy:

- Krzysztof Pawluch
- Adam Badura

Co optymalizować?

Co optymalizować?

- czas wykonania?

Co optymalizować?

- czas wykonania?
- ilość użytej pamięci?

Co optymalizować?

- czas wykonania?
- ilość użytej pamięci?
- czas budowania?

Co optymalizować?

- czas wykonania?
- ilość użytej pamięci?
- czas budowania?
- rozmiar wynikowej aplikacji?

Co optymalizować?

- czas wykonania?
- ilość użytej pamięci?
- czas budowania?
- rozmiar wynikowej aplikacji?
- szybkość wymiany danych (w sieci)?

Co optymalizować?

- czas wykonania?
- ilość użytej pamięci?
- czas budowania?
- rozmiar wynikowej aplikacji?
- szybkość wymiany danych (w sieci)?
- rozmiar kodu źródłowego?

Co optymalizować?

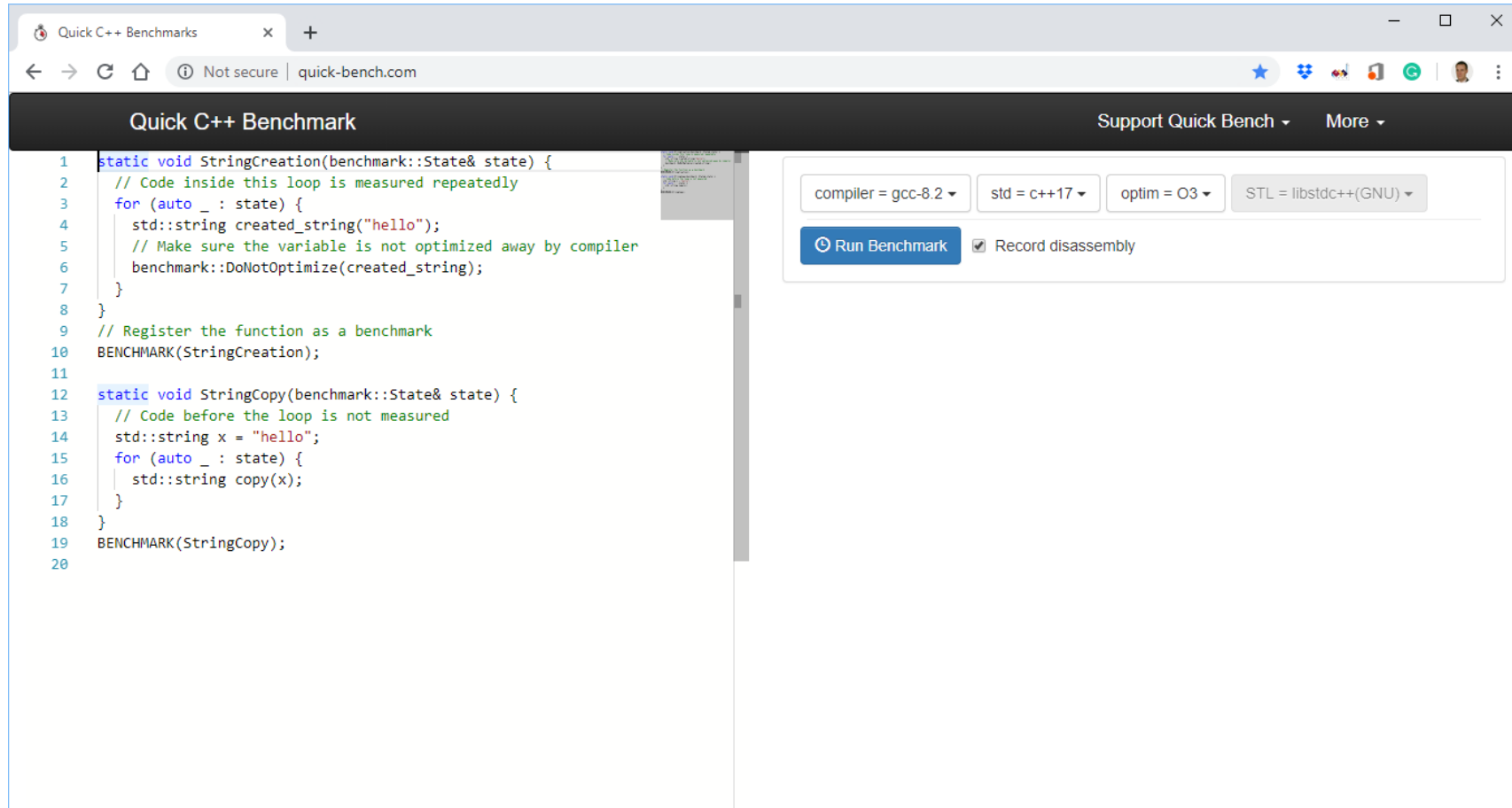
- czas wykonania?
- ilość użytej pamięci?
- czas budowania?
- rozmiar wynikowej aplikacji?
- szybkość wymiany danych (w sieci)?
- rozmiar kodu źródłowego?

→ zgodnie z wymaganiami

→ według potrzeb

→ według testów (bottlenecki)

Quick C++ Benchmarks



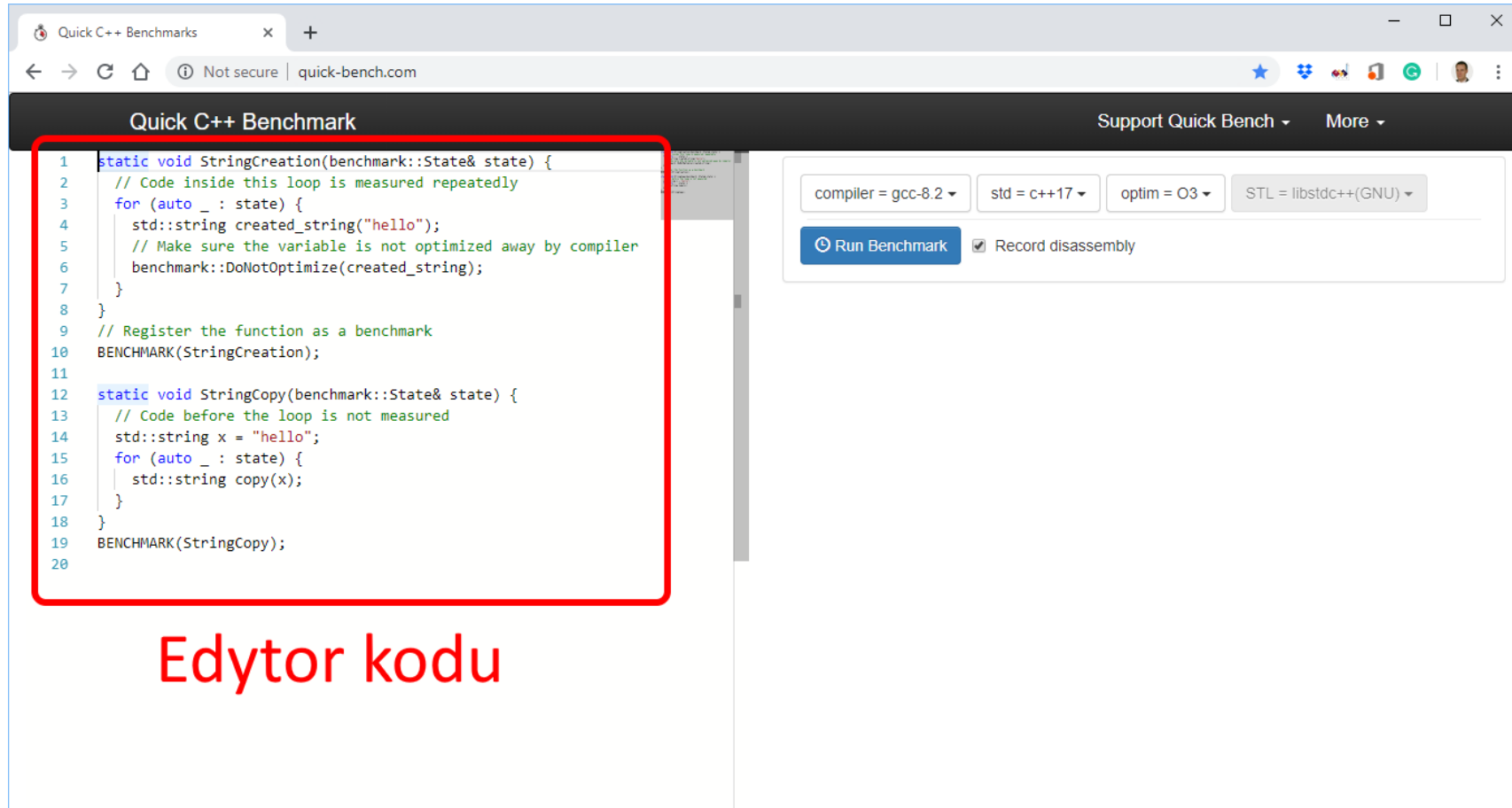
The screenshot shows the Quick C++ Benchmarks website in a web browser. The browser's address bar shows the URL `quick-bench.com`. The website has a dark header with the title "Quick C++ Benchmark" and links for "Support Quick Bench" and "More". The main content area is split into two panels. The left panel contains a C++ code editor with the following code:

```
1 static void StringCreation(benchmark::State& state) {
2     // Code inside this loop is measured repeatedly
3     for (auto _ : state) {
4         std::string created_string("hello");
5         // Make sure the variable is not optimized away by compiler
6         benchmark::DoNotOptimize(created_string);
7     }
8 }
9 // Register the function as a benchmark
10 BENCHMARK(StringCreation);
11
12 static void StringCopy(benchmark::State& state) {
13     // Code before the loop is not measured
14     std::string x = "hello";
15     for (auto _ : state) {
16         std::string copy(x);
17     }
18 }
19 BENCHMARK(StringCopy);
20
```

The right panel contains configuration options for the benchmark. It includes dropdown menus for "compiler = gcc-8.2", "std = c++17", "optim = O3", and "STL = libstdc++(GNU)". Below these is a blue "Run Benchmark" button and a checkbox labeled "Record disassembly" which is currently checked.

<http://quick-bench.com/>

Quick C++ Benchmarks



The screenshot shows the Quick C++ Benchmarks website interface. The browser address bar displays "quick-bench.com". The page title is "Quick C++ Benchmark". The code editor on the left contains the following C++ code:

```
1 static void StringCreation(benchmark::State& state) {  
2     // Code inside this loop is measured repeatedly  
3     for (auto _ : state) {  
4         std::string created_string("hello");  
5         // Make sure the variable is not optimized away by compiler  
6         benchmark::DoNotOptimize(created_string);  
7     }  
8 }  
9 // Register the function as a benchmark  
10 BENCHMARK(StringCreation);  
11  
12 static void StringCopy(benchmark::State& state) {  
13     // Code before the loop is not measured  
14     std::string x = "hello";  
15     for (auto _ : state) {  
16         std::string copy(x);  
17     }  
18 }  
19 BENCHMARK(StringCopy);  
20
```

The code is highlighted with a red box. To the right of the code editor, there are configuration options for the benchmark:

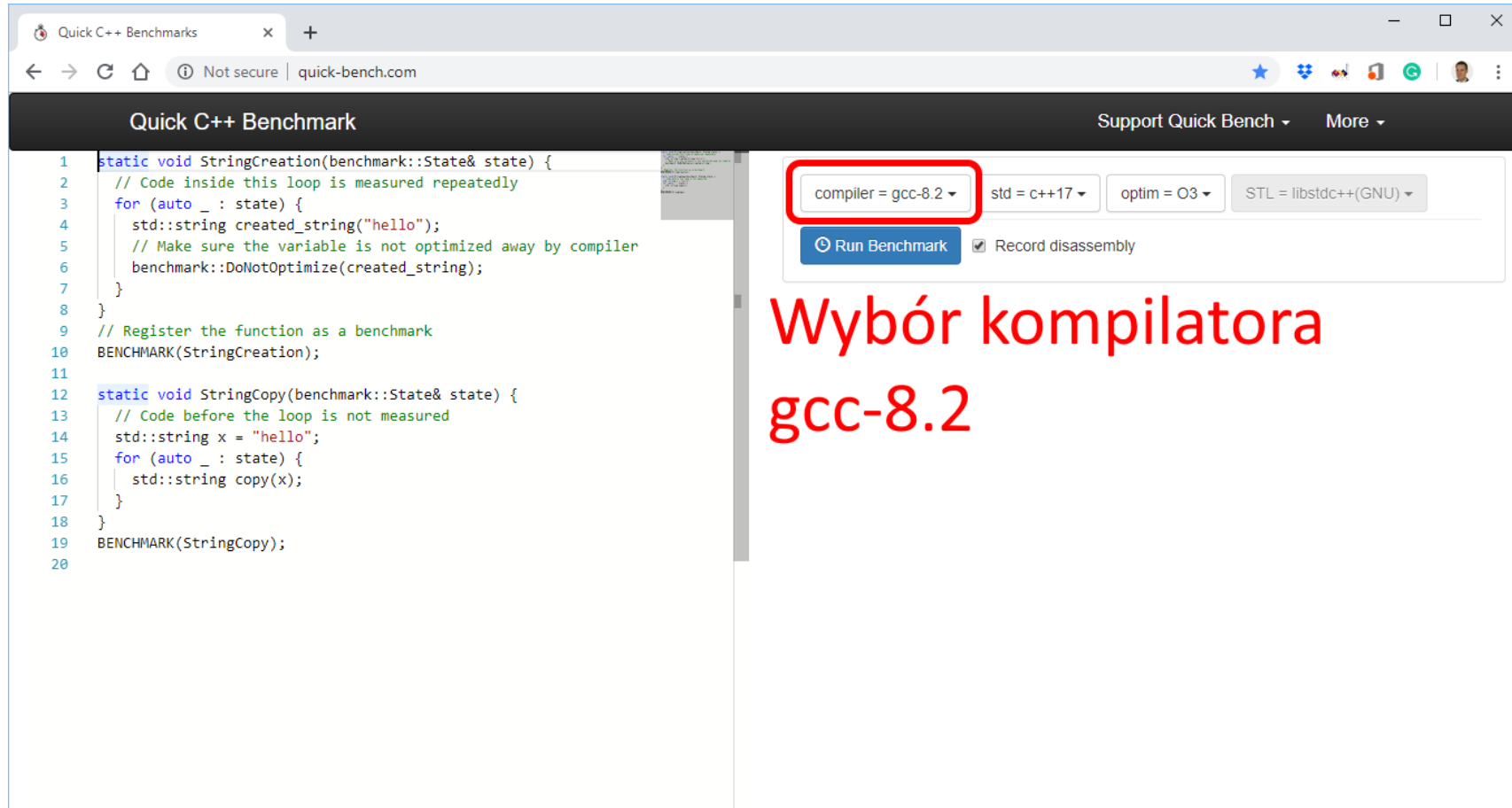
- compiler = gcc-8.2
- std = c++17
- optim = O3
- STL = libstdc++(GNU)

Below these options, there is a "Run Benchmark" button and a checkbox labeled "Record disassembly".

Edytor kodu

<http://quick-bench.com/>

Quick C++ Benchmarks



Quick C++ Benchmark

Support Quick Bench ▾ More ▾

```
1 static void StringCreation(benchmark::State& state) {  
2     // Code inside this loop is measured repeatedly  
3     for (auto _ : state) {  
4         std::string created_string("hello");  
5         // Make sure the variable is not optimized away by compiler  
6         benchmark::DoNotOptimize(created_string);  
7     }  
8 }  
9 // Register the function as a benchmark  
10 BENCHMARK(StringCreation);  
11  
12 static void StringCopy(benchmark::State& state) {  
13     // Code before the loop is not measured  
14     std::string x = "hello";  
15     for (auto _ : state) {  
16         std::string copy(x);  
17     }  
18 }  
19 BENCHMARK(StringCopy);  
20
```

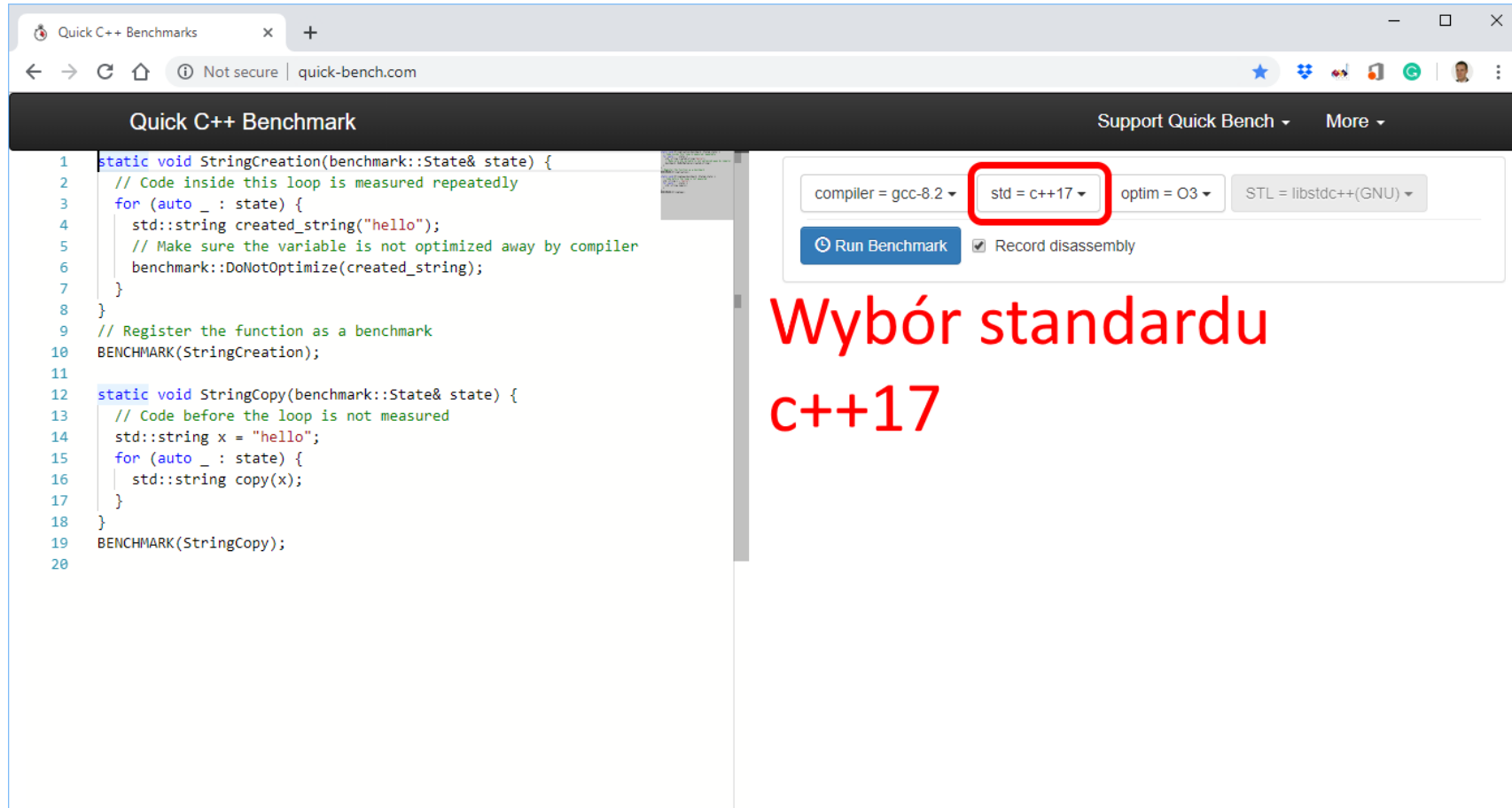
compiler = gcc-8.2 ▾ std = c++17 ▾ optim = O3 ▾ STL = libstdc++(GNU) ▾

Run Benchmark ☒ Record disassembly

Wybór kompilatora
gcc-8.2

<http://quick-bench.com/>

Quick C++ Benchmarks



Quick C++ Benchmark

Support Quick Bench ▾ More ▾

```
1 static void StringCreation(benchmark::State& state) {  
2     // Code inside this loop is measured repeatedly  
3     for (auto _ : state) {  
4         std::string created_string("hello");  
5         // Make sure the variable is not optimized away by compiler  
6         benchmark::DoNotOptimize(created_string);  
7     }  
8 }  
9 // Register the function as a benchmark  
10 BENCHMARK(StringCreation);  
11  
12 static void StringCopy(benchmark::State& state) {  
13     // Code before the loop is not measured  
14     std::string x = "hello";  
15     for (auto _ : state) {  
16         std::string copy(x);  
17     }  
18 }  
19 BENCHMARK(StringCopy);  
20
```

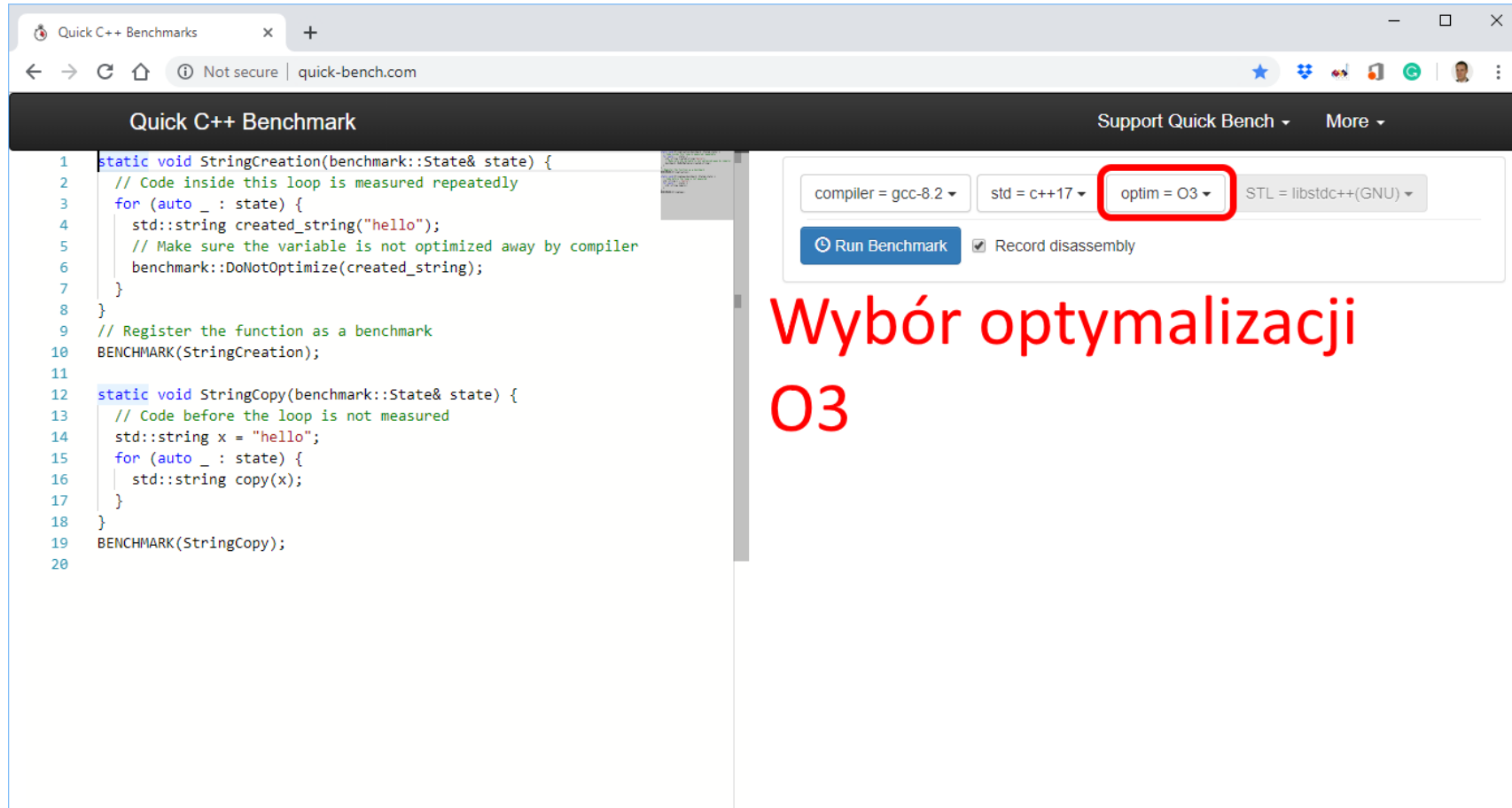
compiler = gcc-8.2 ▾ **std = c++17 ▾** optim = O3 ▾ STL = libstdc++(GNU) ▾

Run Benchmark ☒ Record disassembly

Wybór standardu
c++17

<http://quick-bench.com/>

Quick C++ Benchmarks



Quick C++ Benchmark

Support Quick Bench ▾ More ▾

```
1 static void StringCreation(benchmark::State& state) {  
2     // Code inside this loop is measured repeatedly  
3     for (auto _ : state) {  
4         std::string created_string("hello");  
5         // Make sure the variable is not optimized away by compiler  
6         benchmark::DoNotOptimize(created_string);  
7     }  
8 }  
9 // Register the function as a benchmark  
10 BENCHMARK(StringCreation);  
11  
12 static void StringCopy(benchmark::State& state) {  
13     // Code before the loop is not measured  
14     std::string x = "hello";  
15     for (auto _ : state) {  
16         std::string copy(x);  
17     }  
18 }  
19 BENCHMARK(StringCopy);  
20
```

compiler = gcc-8.2 ▾ std = c++17 ▾ **optim = O3 ▾** STL = libstdc++(GNU) ▾

Run Benchmark ☒ Record disassembly

Wybór optymalizacji
O3

<http://quick-bench.com/>

Quick C++ Benchmarks

Quick C++ Benchmark

Support Quick Bench ▾ More ▾

```
1 static void StringCreation(benchmark::State& state) {
2     // Code inside this loop is measured repeatedly
3     for (auto _ : state) {
4         std::string created_string("hello");
5         // Make sure the variable is not optimized away by compiler
6         benchmark::DoNotOptimize(created_string);
7     }
8 }
9 // Register the function as a benchmark
10 BENCHMARK(StringCreation);
11
12 static void StringCopy(benchmark::State& state) {
13     // Code before the loop is not measured
14     std::string x = "hello";
15     for (auto _ : state) {
16         std::string copy(x);
17     }
18 }
19 BENCHMARK(StringCopy);
20
```

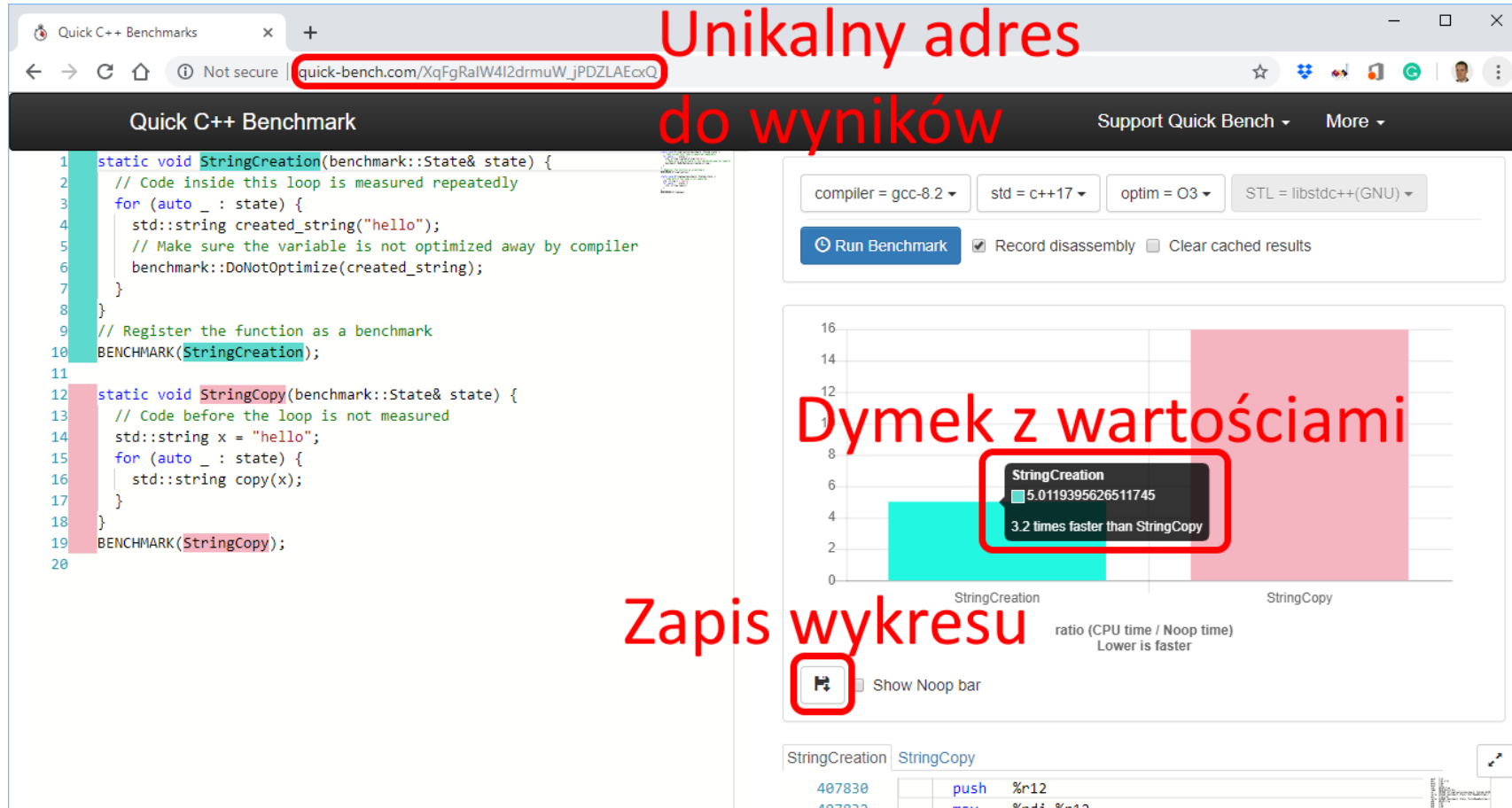
compiler = gcc-8.2 ▾ std = c++17 ▾ optim = O3 ▾ STL = libstdc++(GNU) ▾

Run Benchmark ☒ Record disassembly

Uruchomienie testów

<http://quick-bench.com/>

Quick C++ Benchmarks



<http://quick-bench.com/>

Złożoność algorytmiczna

Złożoność algorytmiczna

- Nie da się wygrać ze złożonością algorytmiczną.

Złożoność algorytmiczna

- Nie da się wygrać ze złożonością algorytmiczną.
- Implementację optymalizujemy, gdy mamy właściwe algorytmy.

Ćwiczenie 1 – Porównanie sortowań

Sortowanie bąbelkowe

średnio $O(n^2)$

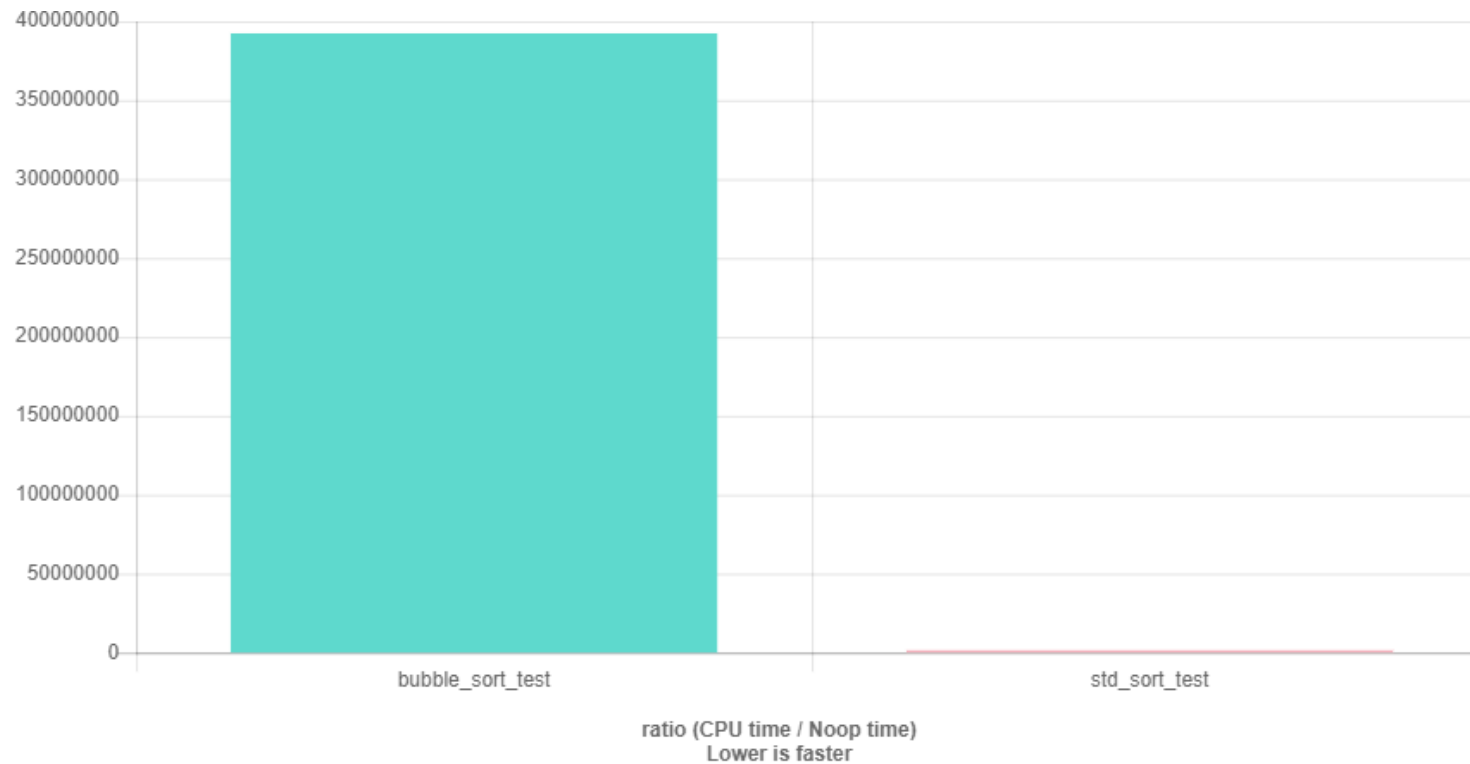
`std::sort`

średnio $O(n \log n)$

Ćwiczenie 1 – Porównanie sortowań

Sortowanie bąbelkowe
średnio $O(n^2)$

`std::sort`
średnio $O(n \log n)$



Złożoność algorytmiczna

- Nie da się wygrać ze złożonością algorytmiczną.
- Implementację optymalizujemy, gdy mamy właściwe algorytmy.
- Sama złożoność może być jednak myląca.

Ćwiczenie 2 – Porównanie sortowań

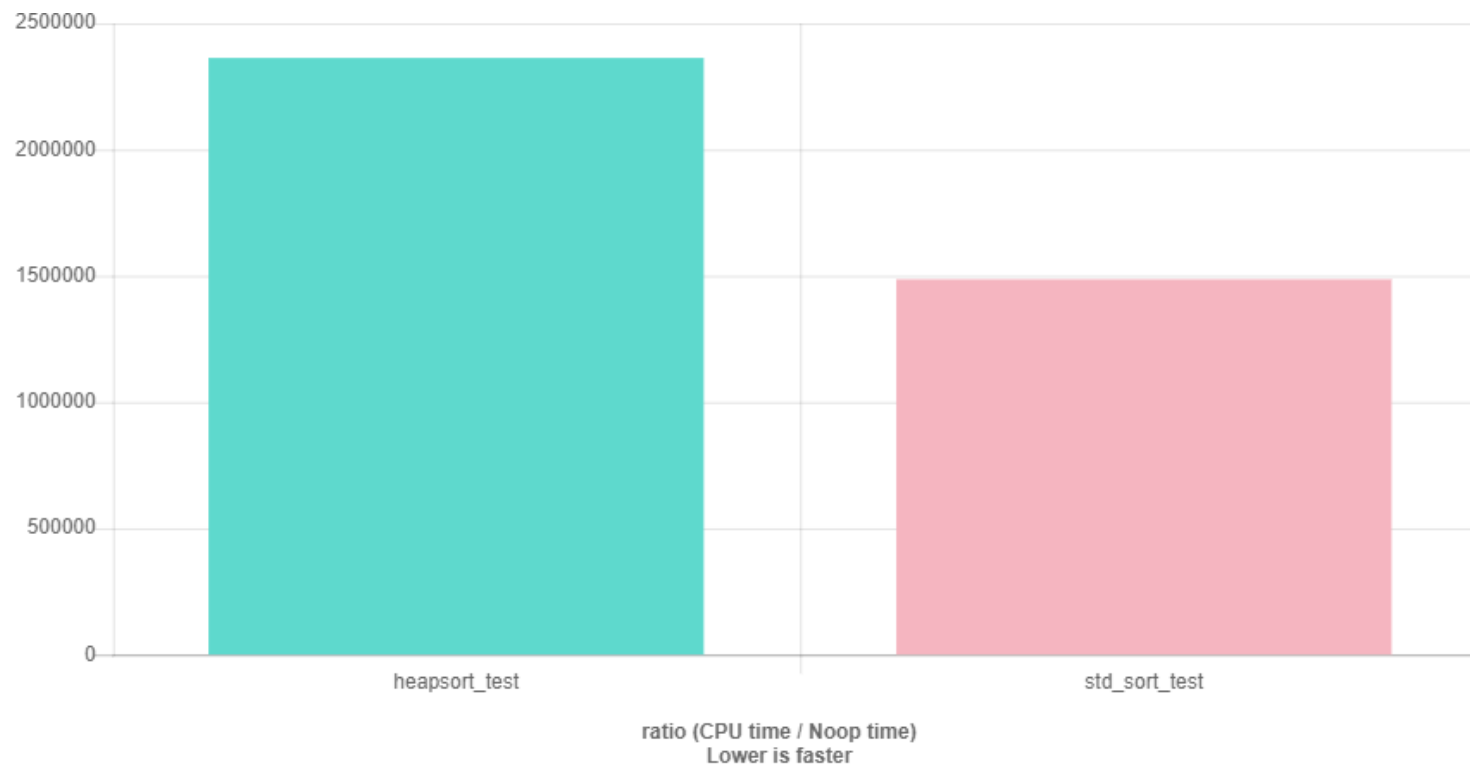
Sortowanie przez kopcowanie
pesymistycznie $O(n \log n)$

`std::sort`
średnio $O(n \log n)$

Ćwiczenie 2 – Porównanie sortowań

Sortowanie przez kopcowanie
pesymistycznie $O(n \log n)$

`std::sort`
średnio $O(n \log n)$



Złożoność algorytmiczna

- Nie da się wygrać ze złożonością algorytmiczną.
- Implementację optymalizujemy, gdy mamy właściwe algorytmy.
- Sama złożoność może być jednak myląca.
- Charakterystyka problemu pozwala dobrać dedykowany algorytm, często „out of the box”.

Ćwiczenie 3 – Porównanie sortowań

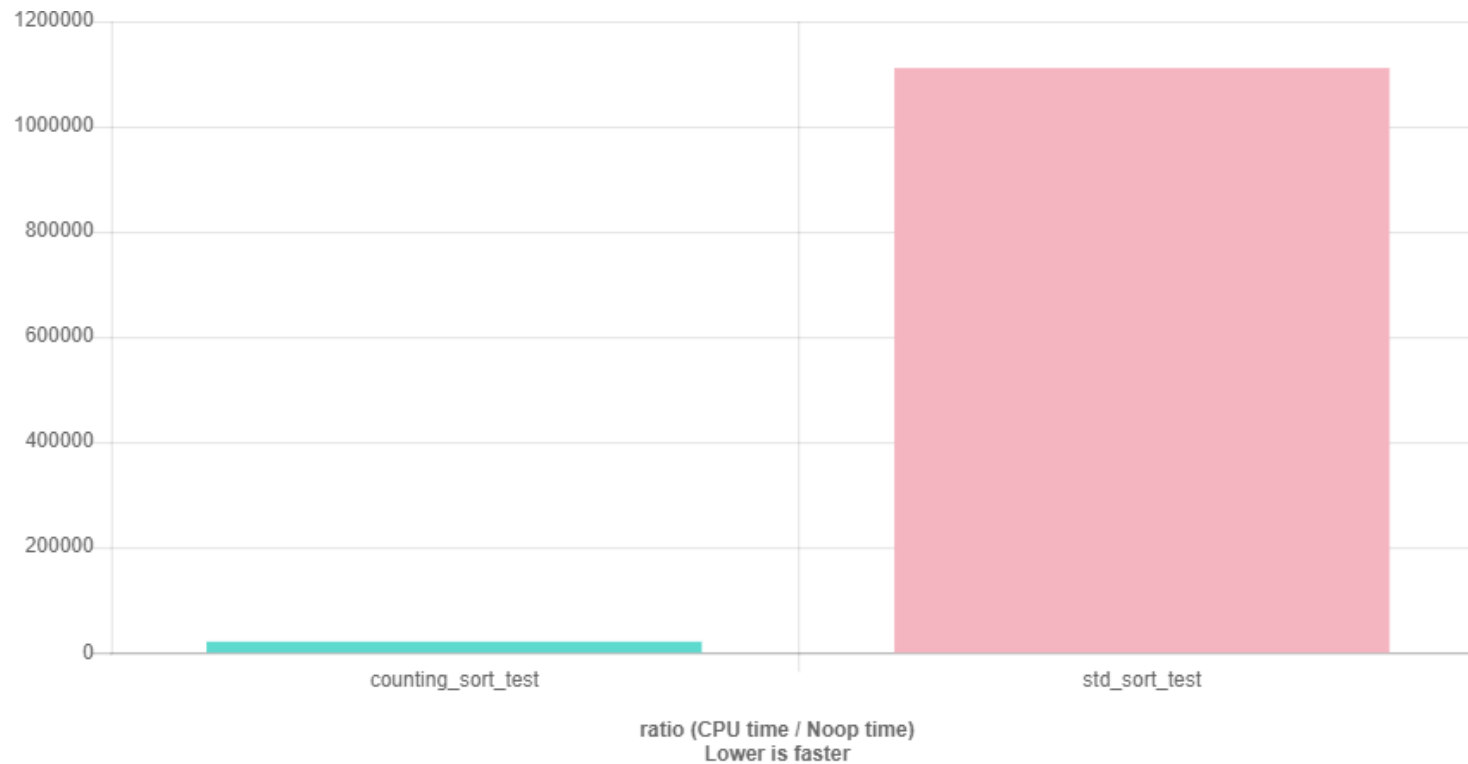
Sortowanie przez zliczanie
pesymistycznie $O(n)$

`std::sort`
średnio $O(n \log n)$

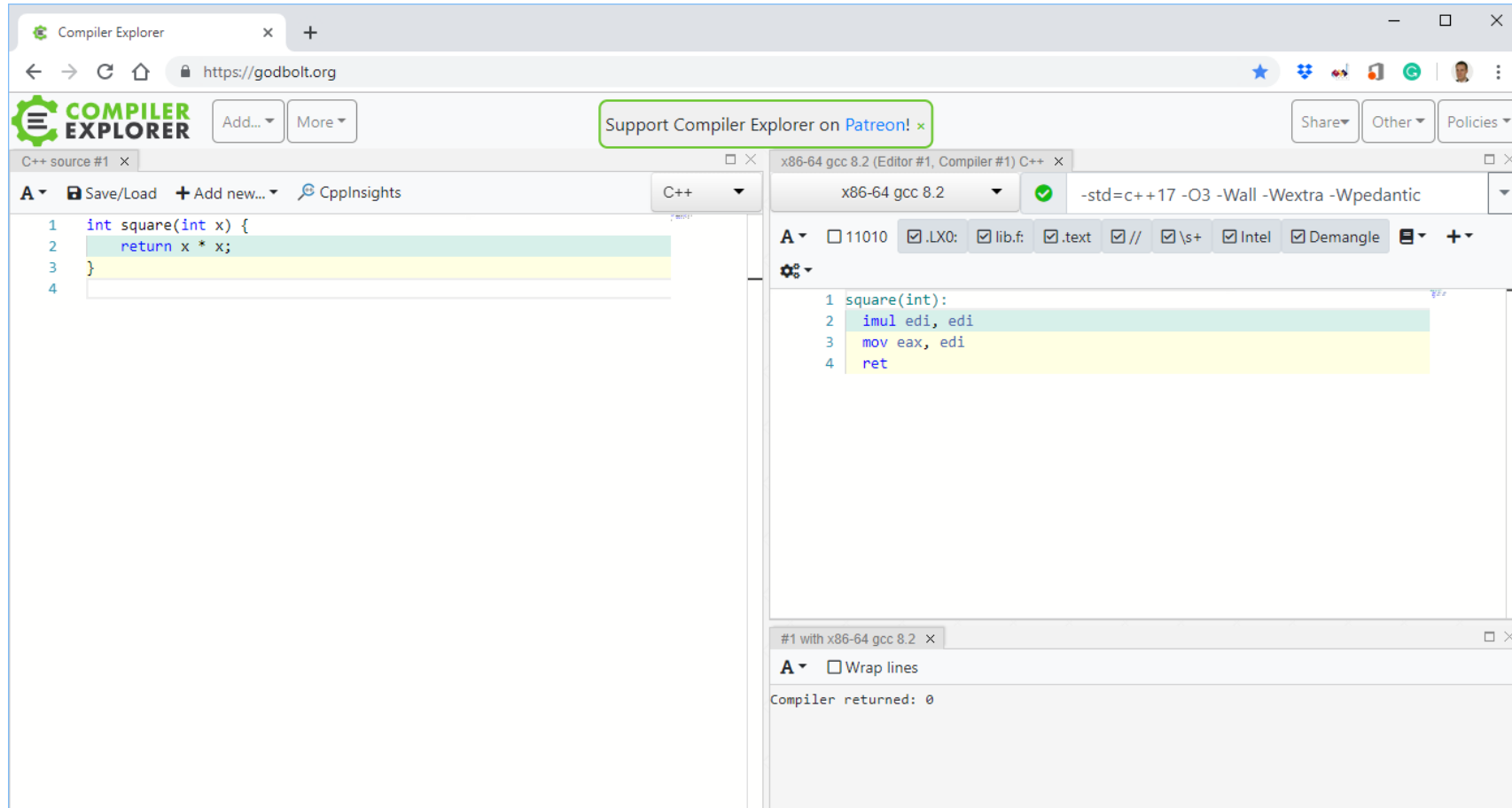
Ćwiczenie 3 – Porównanie sortowań

Sortowanie przez zliczanie
pesymistycznie $O(n)$

`std::sort`
średnio $O(n \log n)$

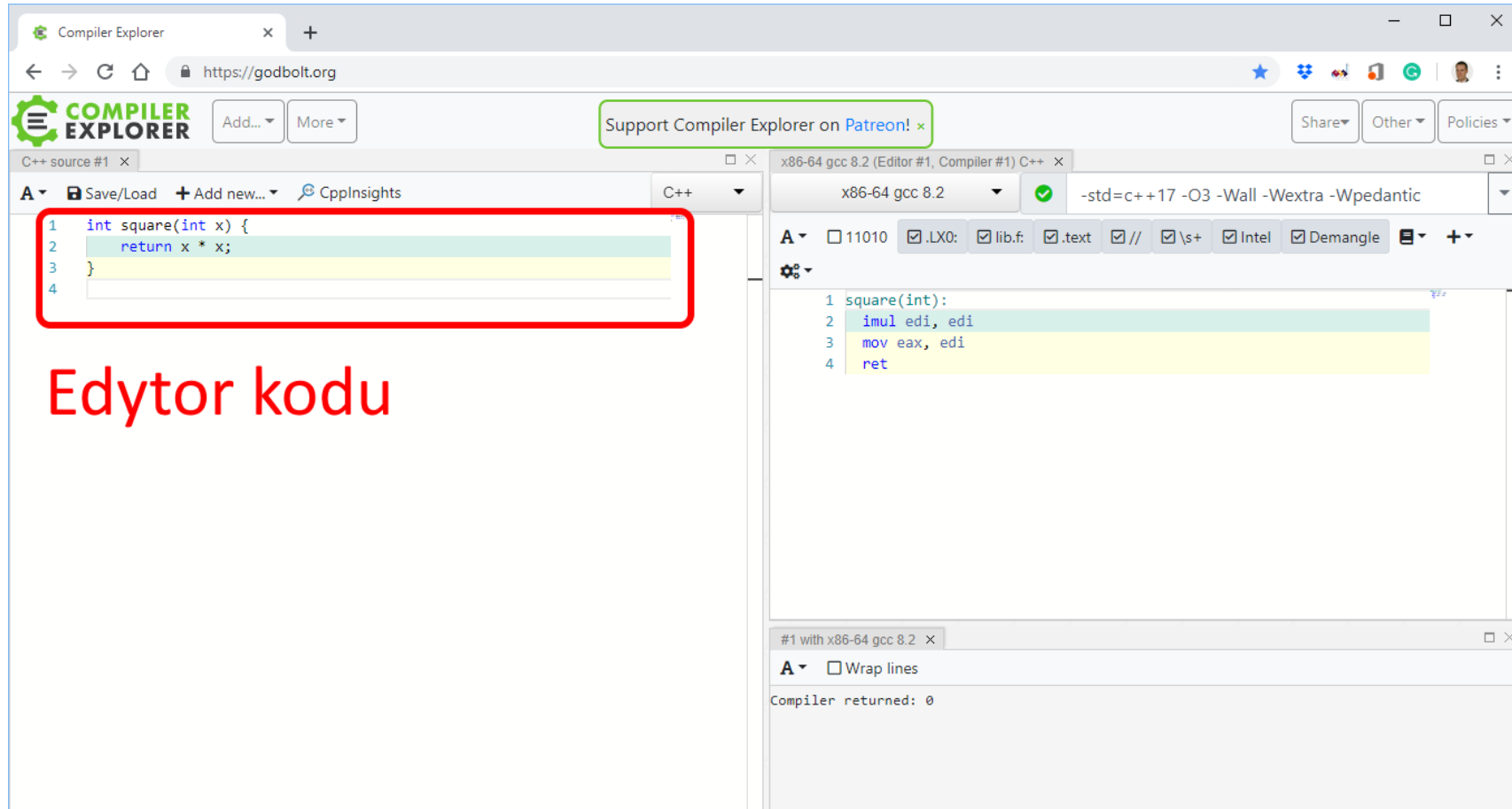


Compiler Explorer



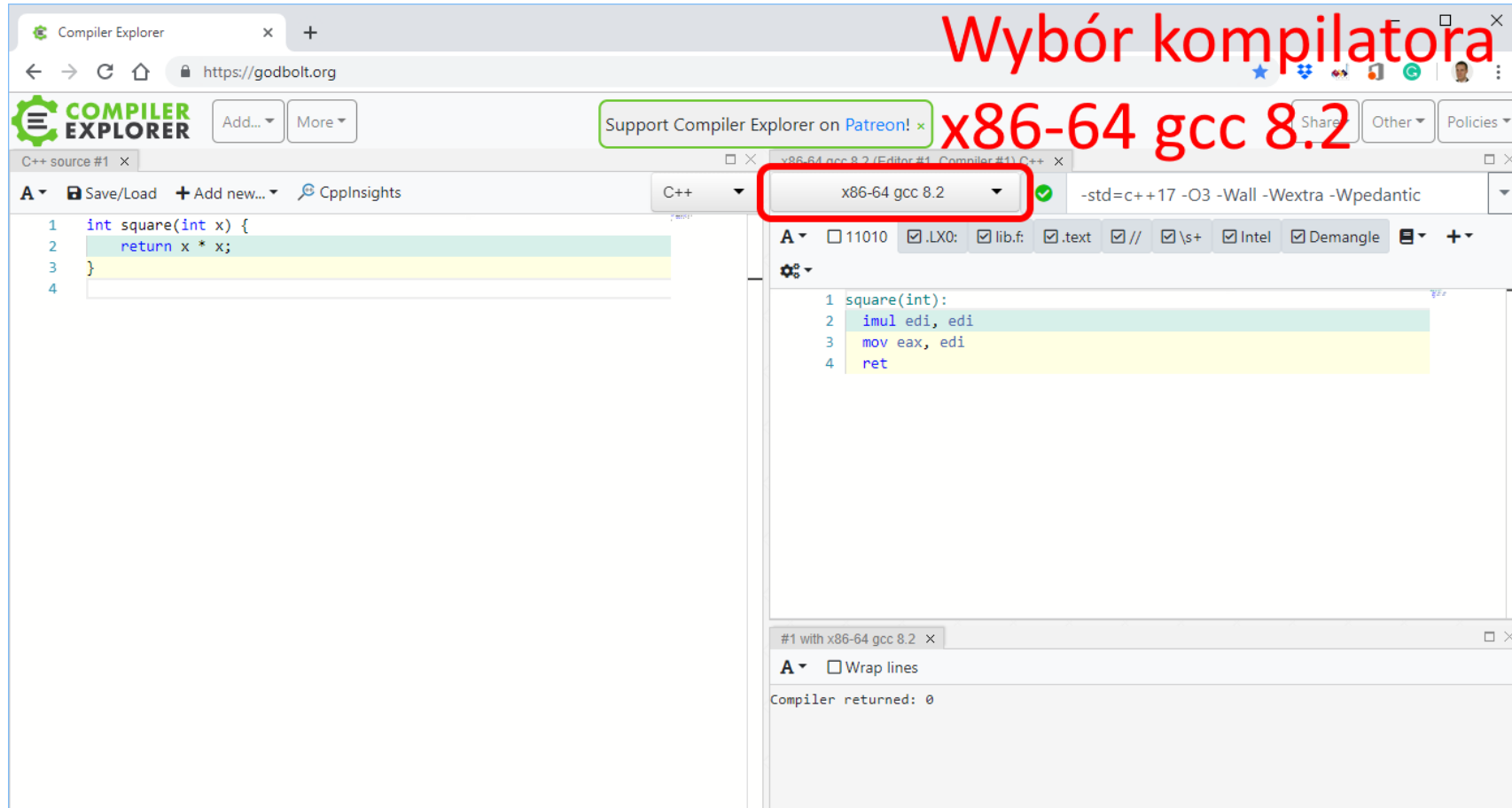
<https://godbolt.org/>

Compiler Explorer



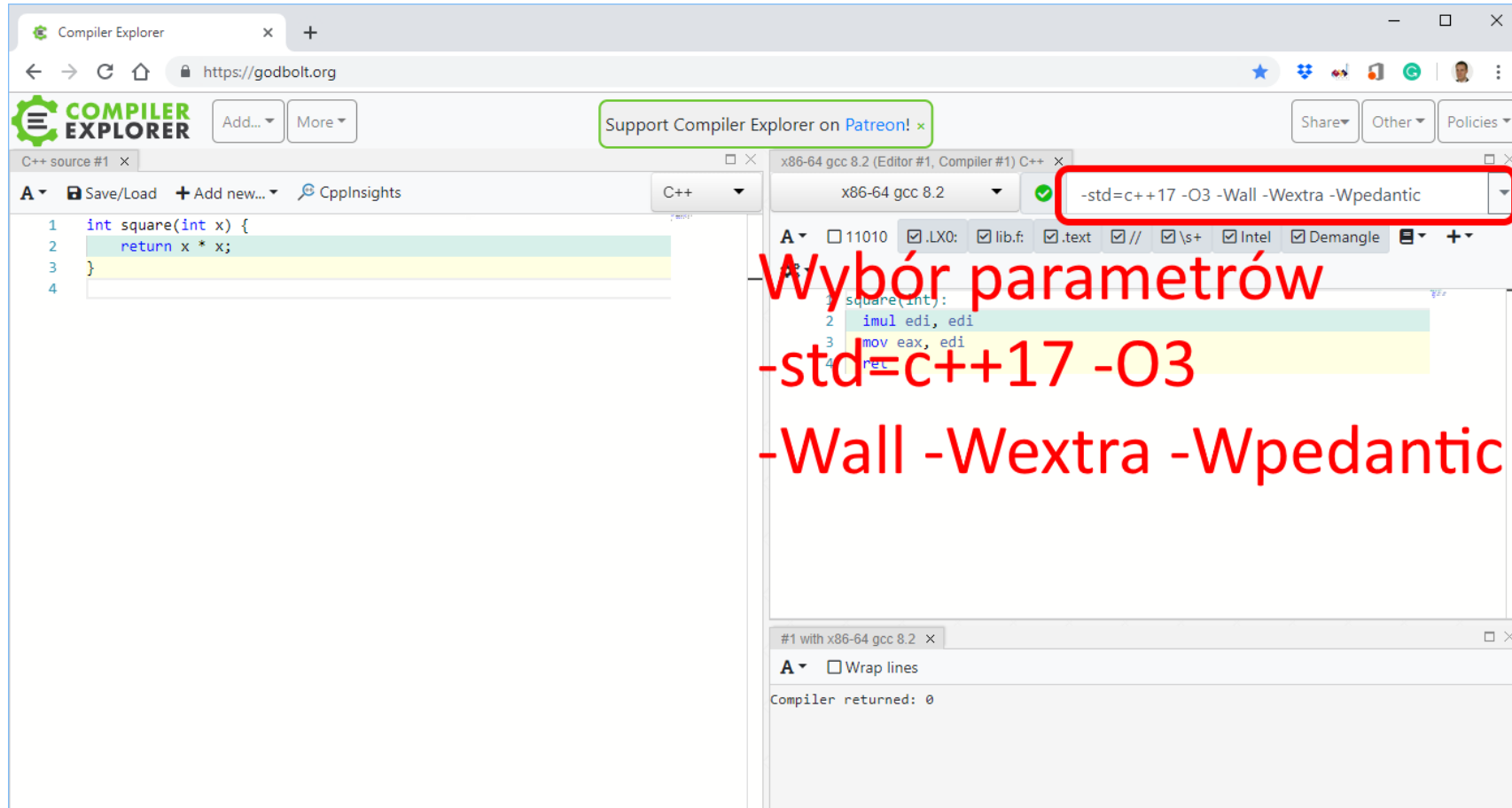
<https://godbolt.org/>

Compiler Explorer



<https://godbolt.org/>

Compiler Explorer



<https://godbolt.org/>

Compiler Explorer

The screenshot displays the Compiler Explorer interface at <https://godbolt.org>. The left pane shows the C++ source code for a function `square`. The right pane shows the assembly output for the same code, generated by `x86-64 gcc 8.2`. The assembly code is highlighted with a red box and labeled "Kod wynikowy". The compiler options are set to `-std=c++17 -O3 -Wall -Wextra -Wpedantic`. The output pane shows the compiler returned 0, which is also highlighted with a red box and labeled "Komunikaty". The "Share" button in the top right corner is highlighted with a red box and labeled "Unikalny adres".

Unikalny adres

Kod wynikowy

Komunikaty

<https://godbolt.org/>

Compiler Explorer – tools (mca)

The screenshot displays the Compiler Explorer web application interface. It features three main panes: a source code editor on the left, an assembly output pane in the center, and an analysis tools pane on the right.

Source Code (C++):

```
1 long long fib(int x)
2 {
3     if (x == 1)
4     {
5         return 1;
6     }
7     if (x == 2)
8     {
9         return 1;
10    }
11    return fib(x-1)*fib(x-2);
12 }
13
14 int main()
15 {
16     return static_cast<int>(fib(13)%128);
17 }
```

Assembly Output (x86-64 clang 11.0.0):

```
1 fib(int):
2     push    rbp
3     mov     rbp, rsp
4     sub     rsp, 32
5     mov     dword ptr [rbp-12], 0
6     cmp     dword ptr [rbp-12], 1
7     jne     .LBB0_2
8     mov     qword ptr [rbp-24], 0
9     jmp     .LBB0_5
10 .LBB0_2:
11     cmp     dword ptr [rbp-12], 2
12     jne     .LBB0_4
13     mov     qword ptr [rbp-24], 1
14     jmp     .LBB0_5
15 .LBB0_4:
16     mov     eax, dword [rbp-12]
17     sub     eax, 1
18     mov     edi, eax
19     call    fib(int)
20     mov     ecx, dword ptr [rbp-12]
21     sub     ecx, 2
22     mov     edi, ecx
23     mov     qword ptr [rbp-24], rax
24     call    fib(int)
25     mov     rdx, qword ptr [rbp-24]
```

Analysis Tools (llvm-mca):

Instruction Info:

- [1]: #uOps
- [2]: Latency
- [3]: RThroughput
- [4]: MayLoad
- [5]: MayStore
- [6]: HasSideEffects (U)

[1]	[2]	[3]	[4]	[5]	[6]	Instructions:
3	2	1.00		*		push rbp
1	1	0.25				mov rbp, rsp
1	1	0.25				sub rsp, 32
1	1	1.00		*		mov dword ptr [rbp-12], 0
2	6	0.50	*			cmp dword ptr [rbp-12], 1
1	1	0.50				jne .LBB0_2
1	1	1.00		*		mov qword ptr [rbp-24], 0
1	1	0.50				jmp .LBB0_5
2	6	0.50	*			cmp dword ptr [rbp-12], 2
1	1	0.50				jne .LBB0_4

Compiler returned: 0

Ćwiczenie 4 – Mnożenie a przesunięcie

mnożenie

```
int foo(int x) {  
    return x * 2;  
}
```

przesunięcie bitowe

```
int bar(int x) {  
    return x << 1;  
}
```

Optymalizacje kompilatora

Ćwiczenie 4 – Mnożenie a przesunięcie

mnożenie

```
int foo(int x) {  
    return x * 2;  
}
```

```
foo(int):  
    lea eax, [rdi+rdi]  
    ret
```

przesunięcie bitowe

```
int bar(int x) {  
    return x << 1;  
}
```

```
bar(int):  
    lea eax, [rdi+rdi]  
    ret
```

Ćwiczenie 5 – Dodawanie

dodawanie

```
int foo(int x, int y) {  
    x = x + y;  
    return x;  
}
```

dodawanie z przypisaniem

```
int bar(int x, int y) {  
    x += y;  
    return x;  
}
```

Ćwiczenie 5 – Dodawanie

dodawanie

```
int foo(int x, int y) {  
    x = x + y;  
    return x;  
}
```

```
foo(int, int):  
    lea eax, [rdi+rsi]  
    ret
```

dodawanie z przypisaniem

```
int bar(int x, int y) {  
    x += y;  
    return x;  
}
```

```
bar(int, int):  
    lea eax, [rdi+rsi]  
    ret
```

Ćwiczenie 6 – Dodawanie a inkrementacja

dodawanie

```
int foo(int x) {  
    x = x + 1;  
    return x;  
}
```

inkrementowanie

```
int bar(int x) {  
    ++x;  
    return x;  
}
```

Ćwiczenie 6 – Dodawanie a inkrementacja

dodawanie

```
int foo(int x) {  
    x = x + 1;  
    return x;  
}
```

```
foo(int):  
    lea eax, [rdi+1]  
    ret
```

inkrementowanie

```
int bar(int x) {  
    ++x;  
    return x;  
}
```

```
bar(int):  
    lea eax, [rdi+1]  
    ret
```


Ćwiczenie 7 – Inkrementacja post i pre

Post-inkrementacja

```
int foo(int x) {  
    x++;  
    return x;  
}
```

Pre-inkrementacja

```
int bar(int x) {  
    ++x;  
    return x;  
}
```

Ćwiczenie 7 – Inkrementacja post i pre

Post-inkrementacja

```
int foo(int x) {  
    x++;  
    return x;  
}
```

```
foo(int):  
    lea eax, [rdi+1]  
    ret
```

Pre-inkrementacja

```
int bar(int x) {  
    ++x;  
    return x;  
}
```

```
bar(int):  
    lea eax, [rdi+1]  
    ret
```

Ćwiczenie 8.1 – Dzielenie przez zmienną

```
int foo(int x, int y) {  
    return x / y;  
}
```

Ćwiczenie 8.1 – Dzielenie przez zmienną

```
int foo(int x, int y) {  
    return x / y;  
}
```

```
foo(int, int):  
    mov eax, edi  
    cdq  
    idiv esi  
    ret
```

Ćwiczenie 8.2 – Dzielenie przez stałą cz. 1

```
int foo(int x) {  
    return x / 2;  
}
```

Ćwiczenie 8.2 – Dzielenie przez stałą cz. 1

```
int foo(int x) {  
    return x / 2;  
}
```

```
foo(int):  
    mov eax, edi  
    shr eax, 31  
    add eax, edi  
    sar eax  
    ret
```

Ćwiczenie 8.3 – Dzielenie przez stałą cz. 2

```
int foo(int x) {  
    return x / 10;  
}
```

Ćwiczenie 8.3 – Dzielenie przez stałą cz. 2

```
int foo(int x) {  
    return x / 10;  
}
```

```
foo(int):  
    mov eax, edi  
    mov edx, 1717986919  
    sar edi, 31  
    imul edx  
    sar edx, 2  
    mov eax, edx  
    sub eax, edi  
    ret
```


Ćwiczenie 9.1 – Nazwy enumeracji

```
enum class color {  
    black,  
    maroon,  
    green,  
    olive,  
    navy,  
    purple,  
    teal,  
    silver,  
    gray,  
    red,  
    lime,  
    yellow,  
    blue,  
    fuchsia,  
    aqua,  
    white  
};
```

Ćwiczenie 9.2 – Nazwy enumeracji, **switch**

```
char const* enum_to_c_str(color v) {  
    switch (v) {  
#define CASE(x) case x: return #x  
        CASE(color::black);  
        CASE(color::maroon);  
        CASE(color::green);  
        CASE(color::olive);  
        CASE(color::navy);  
        CASE(color::purple);  
        CASE(color::teal);  
        CASE(color::silver);  
        CASE(color::gray);  
        CASE(color::red);  
        CASE(color::lime);  
        CASE(color::yellow);  
        CASE(color::blue);  
        CASE(color::fuchsia);  
        CASE(color::aqua);  
        CASE(color::white);  
#undef CASE  
    }  
}
```

Ćwiczenie 9.2 – Nazwy enumeracji, **switch**

```
char const* enum_to_c_str(color v) {  
    switch (v) {  
#define CASE(x) case x: return #x  
        CASE(color::black);  
        CASE(color::maroon);  
        CASE(color::green);  
        CASE(color::olive);  
        CASE(color::navy);  
        CASE(color::purple);  
        CASE(color::teal);  
        CASE(color::silver);  
        CASE(color::gray);  
        CASE(color::red);  
        CASE(color::lime);  
        CASE(color::yellow);  
        CASE(color::blue);  
        CASE(color::fuchsia);  
        CASE(color::aqua);  
        CASE(color::white);  
#undef CASE  
    }  
}
```

```
enum_to_c_str(color):  
    mov edi, edi  
    mov rax, QWORD PTR CSWTCH.0[0+rdi*8]  
    ret  
// (...) stałe
```

Ćwiczenie 9.3 – Nazwy enumeracji, `std::map`

```
char const* enum_to_c_str(color v) {  
    static std::map<color, char const*> const  
    mapping{  
#define CASE(x) {x, #x}  
        CASE(color::black),  
        CASE(color::maroon),  
        CASE(color::green),  
        CASE(color::olive),  
        CASE(color::navy),  
        CASE(color::purple),  
        CASE(color::teal),  
        CASE(color::silver),  
        CASE(color::gray),  
        CASE(color::red),  
        CASE(color::lime),  
        CASE(color::yellow),  
        CASE(color::blue),  
        CASE(color::fuchsia),  
        CASE(color::aqua),  
        CASE(color::white)  
#undef CASE  
    };  
    return mapping.find(v)->second;  
}
```

Ćwiczenie 9.3 – Nazwy enumeracji, std::map

```
char const* enum_to_c_str(color v) {  
    static std::map<color, char const*> const  
    mapping{  
#define CASE(x) {x, #x}  
        CASE(color::black),  
        CASE(color::maroon),  
        CASE(color::green),  
        CASE(color::olive),  
        CASE(color::navy),  
        CASE(color::purple),  
        CASE(color::teal),  
        CASE(color::silver),  
        CASE(color::gray),  
        CASE(color::red),  
        CASE(color::lime),  
        CASE(color::yellow),  
        CASE(color::blue),  
        CASE(color::fuchsia),  
        CASE(color::aqua),  
        CASE(color::white)  
#undef CASE  
    };  
    return mapping.find(v)->second;  
}
```



Ćwiczenie 9.4 – Nazwy enumeracji, `std::unordered_map`

```
char const* enum_to_c_str(color v) {  
    static std::unordered_map<color, char const*>  
    const mapping{  
#define CASE(x) {x, #x}  
        CASE(color::black),  
        CASE(color::maroon),  
        CASE(color::green),  
        CASE(color::olive),  
        CASE(color::navy),  
        CASE(color::purple),  
        CASE(color::teal),  
        CASE(color::silver),  
        CASE(color::gray),  
        CASE(color::red),  
        CASE(color::lime),  
        CASE(color::yellow),  
        CASE(color::blue),  
        CASE(color::fuchsia),  
        CASE(color::aqua),  
        CASE(color::white)  
#undef CASE  
    };  
    return mapping.find(v)->second;  
}
```

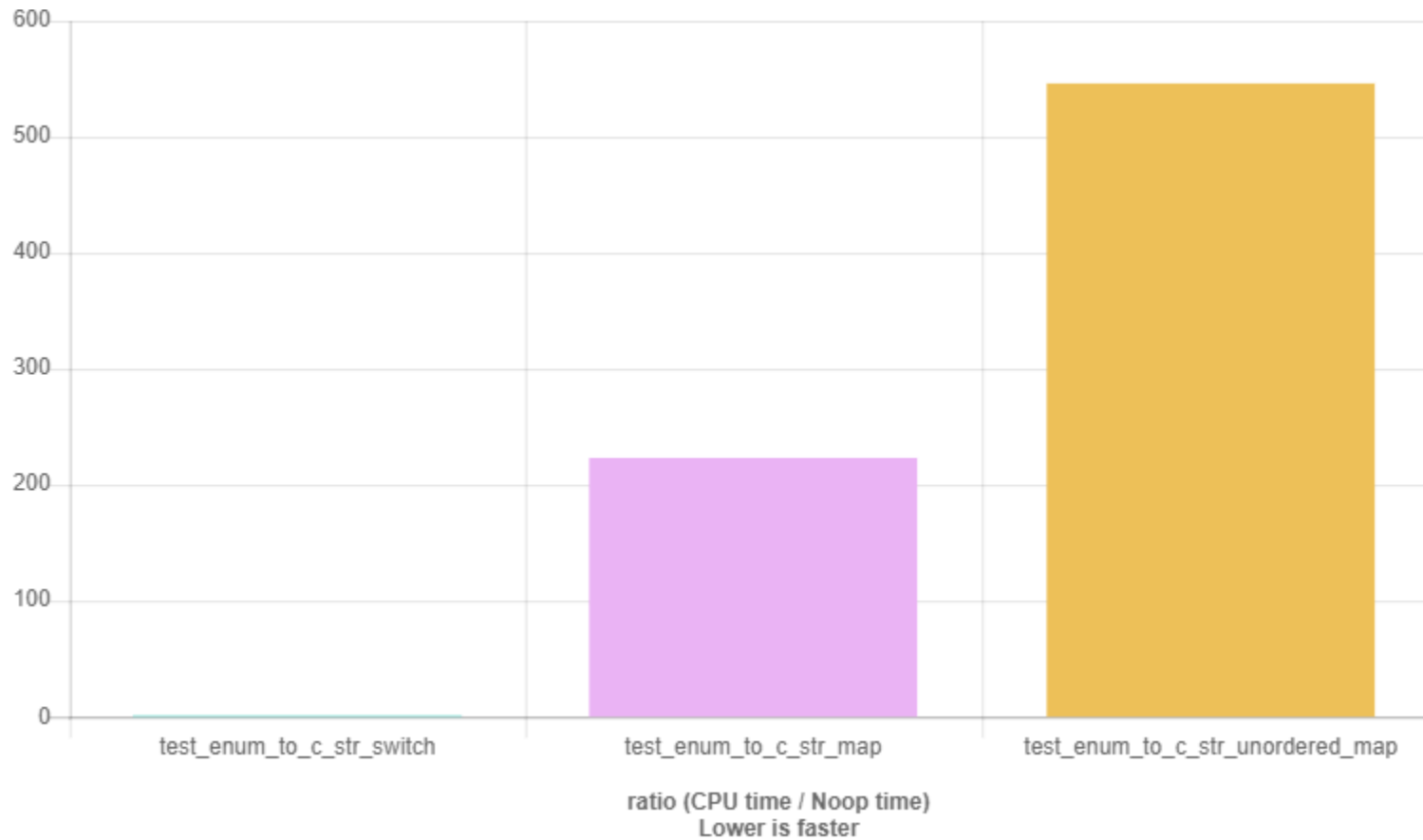
Ćwiczenie 9.4 – Nazwy enumeracji, `std::unordered_map`

```
char const* enum_to_c_str(color v) {  
    static std::unordered_map<color, char const*>  
    const mapping{  
#define CASE(x) {x, #x}  
        CASE(color::black),  
        CASE(color::maroon),  
        CASE(color::green),  
        CASE(color::olive),  
        CASE(color::navy),  
        CASE(color::purple),  
        CASE(color::teal),  
        CASE(color::silver),  
        CASE(color::gray),  
        CASE(color::red),  
        CASE(color::lime),  
        CASE(color::yellow),  
        CASE(color::blue),  
        CASE(color::fuchsia),  
        CASE(color::aqua),  
        CASE(color::white)  
#undef CASE  
    };  
    return mapping.find(v)->second;  
}
```



Ćwiczenie 9.5 – Nazwy enumeracji, porównanie

Ćwiczenie 9.5 – Nazwy enumeracji, porównanie



http://quick-bench.com/FkZ_VVPk8CTi9f9U7wO66qgnU28

Ćwiczenie 10 – Konkatenacja napisów

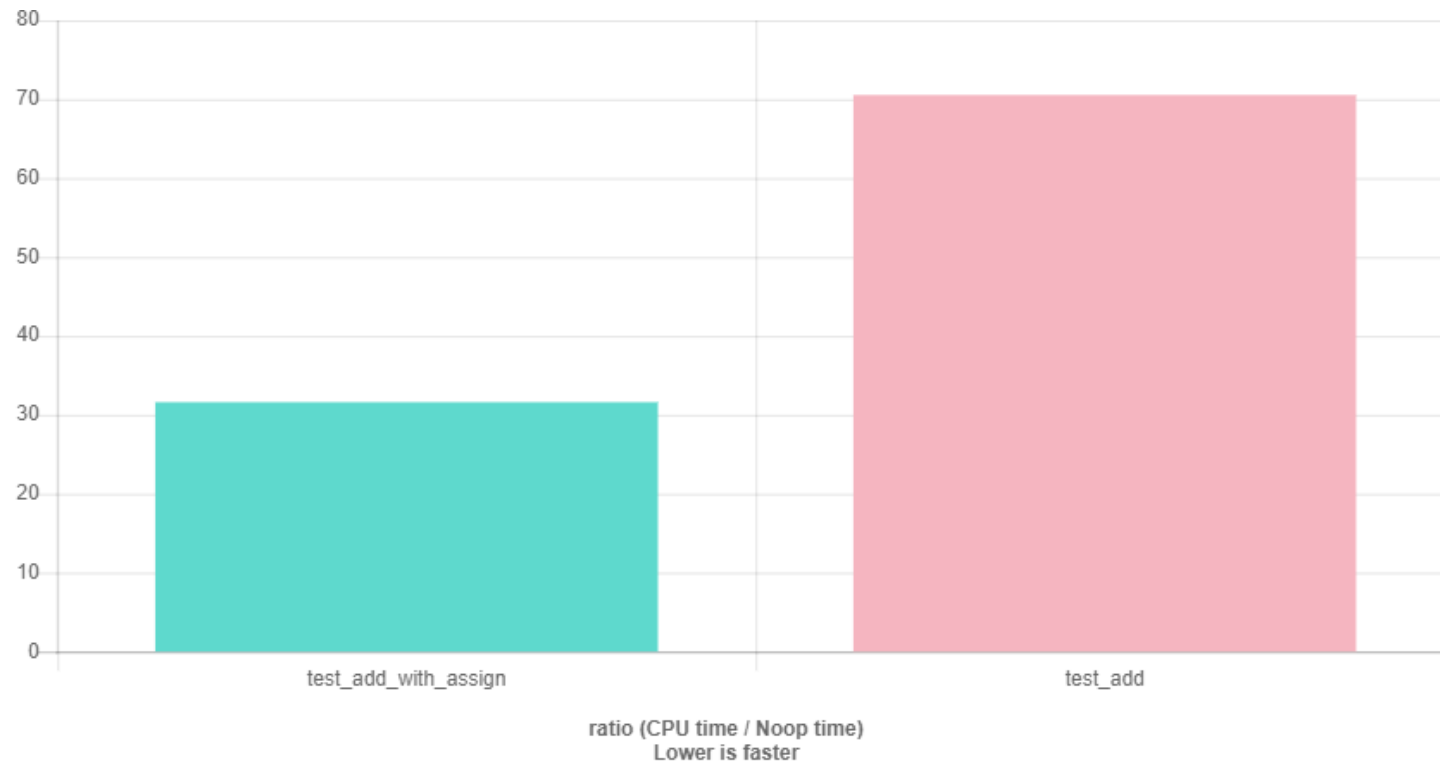
`text += right`

`text = text + right`

Ćwiczenie 10 – Konkatenacja napisów

text += right

text = text + right



<http://quick-bench.com/ATuuralLJ3OOiMMkHJiB2pXWL8g>

Ćwiczenie 11 – Sumowanie, wiele wątków

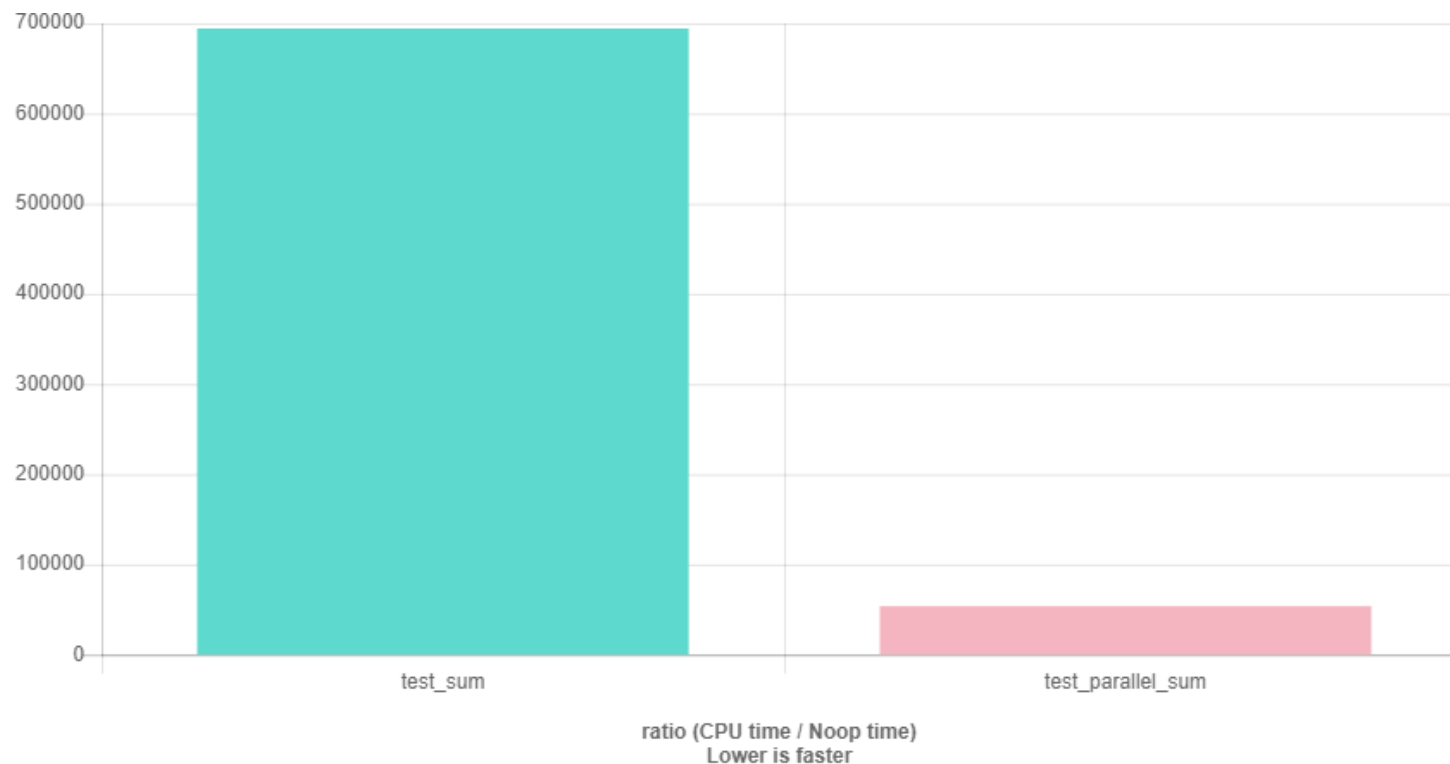
Jeden wątek

Sprzętowa liczba wątków

Ćwiczenie 11 – Sumowanie, wiele wątków

Jeden wątek

Sprzętowa liczba wątków



<http://quick-bench.com/xRmYZBeFPCa9-MCFTOtINJNhVGU>

Ćwiczenie 12 – Sumowanie, bardzo wiele wątków

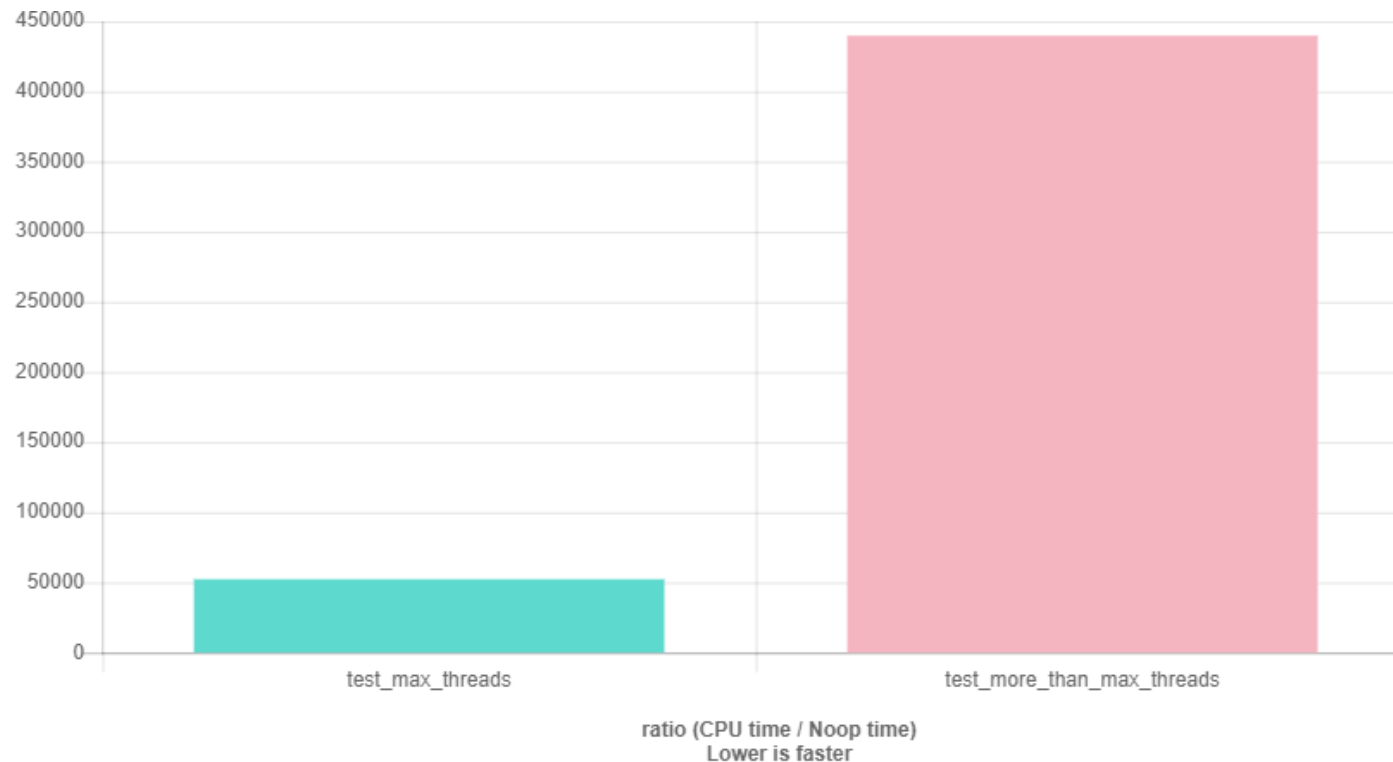
Sprzętowa liczba wątków

Jeszcze więcej wątków

Ćwiczenie 12 – Sumowanie, bardzo wiele wątków

Sprzętowa liczba wątków

Jeszcze więcej wątków



http://quick-bench.com/INFBg-4gcsSpBD7i_cBkIIW9g8k

Ćwiczenie 13 – Sumowanie, małe dane

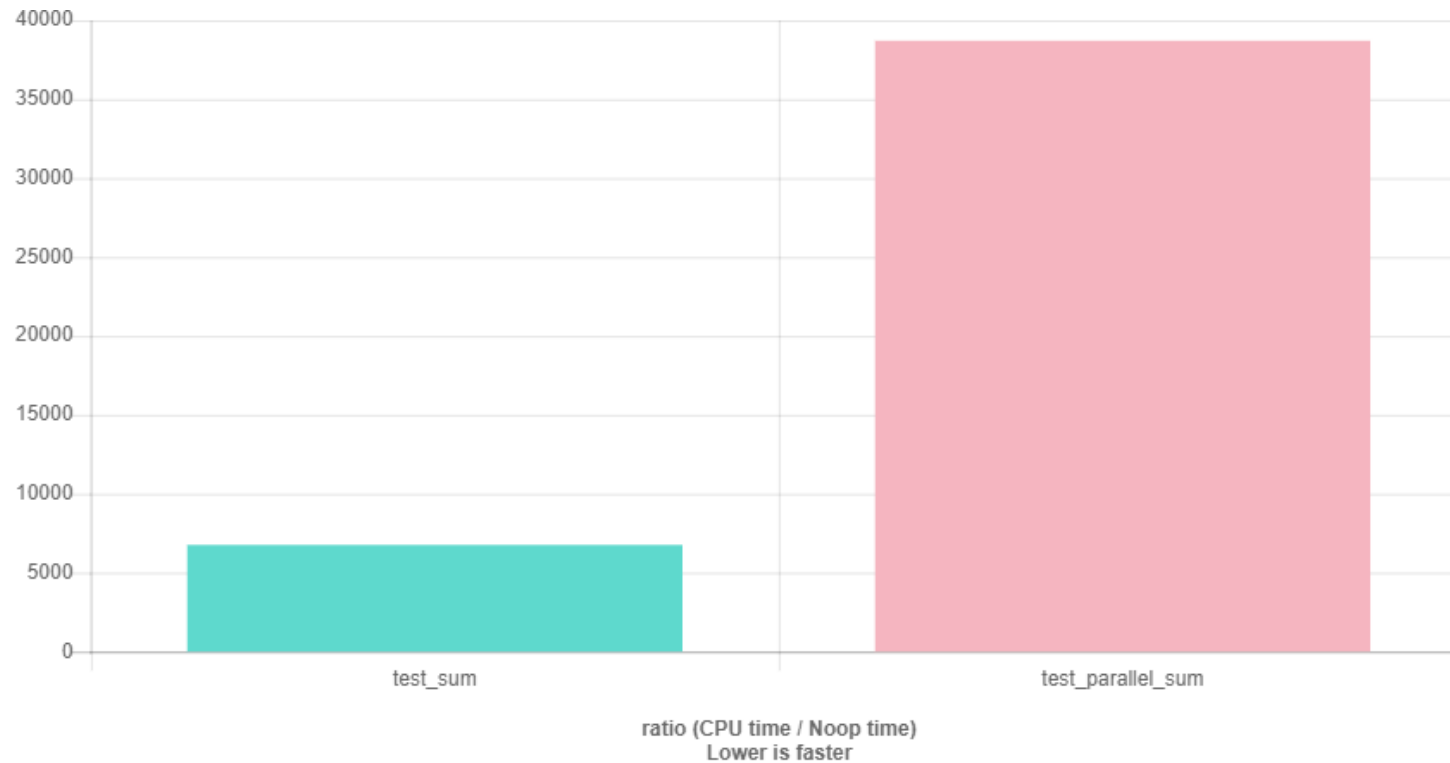
Jeden wątek

Sprzętowa liczba wątków

Ćwiczenie 13 – Sumowanie, małe dane

Jeden wątek

Sprzętowa liczba wątków



<http://quick-bench.com/G1irEF3EdfANeyP2NaDBJexSSAY>

Programowanie dynamiczne

Programowanie dynamiczne

- Dzielenie problemu na podproblemy o tym samym typie, ale mniejszym rozmiarze.

Programowanie dynamiczne

- Dzielenie problemu na podproblemy o tym samym typie, ale mniejszym rozmiarze.
- Podproblemy są rozwiązywane tylko raz, a wynik jest zapamiętywany („memoizacja”).

Programowanie dynamiczne

- Dzielenie problemu na podproblemy o tym samym typie, ale mniejszym rozmiarze.
- Podproblemy są rozwiązywane tylko raz, a wynik jest zapamiętywany („memoizacja”).
- Kluczowe jest rekurencyjne rozbitcie problemu na podproblemy.

Ćwiczenie 14 – Dyskretny problem plecakowy

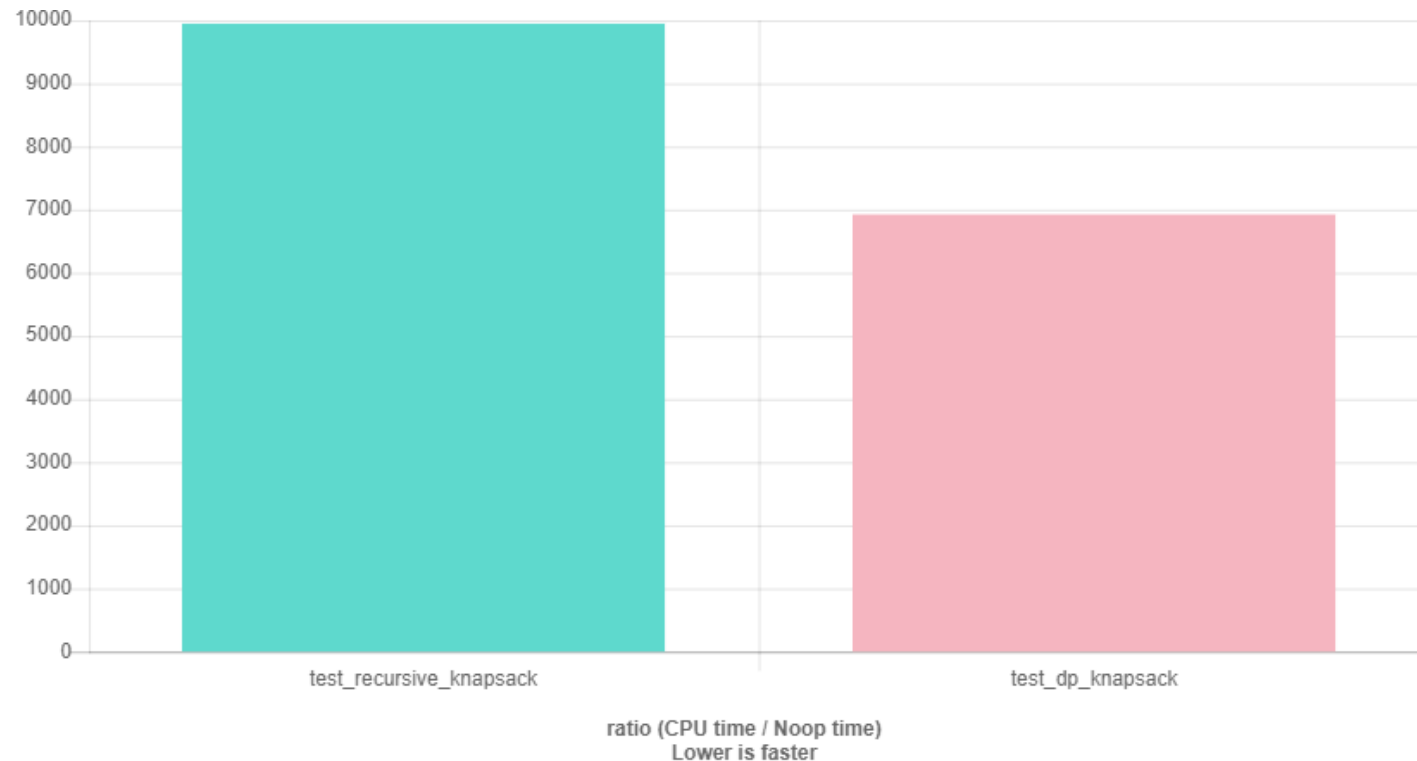
rekurencja

programowanie dynamiczne

Ćwiczenie 14 – Dyskretny problem plecakowy

rekurencja

programowanie dynamiczne

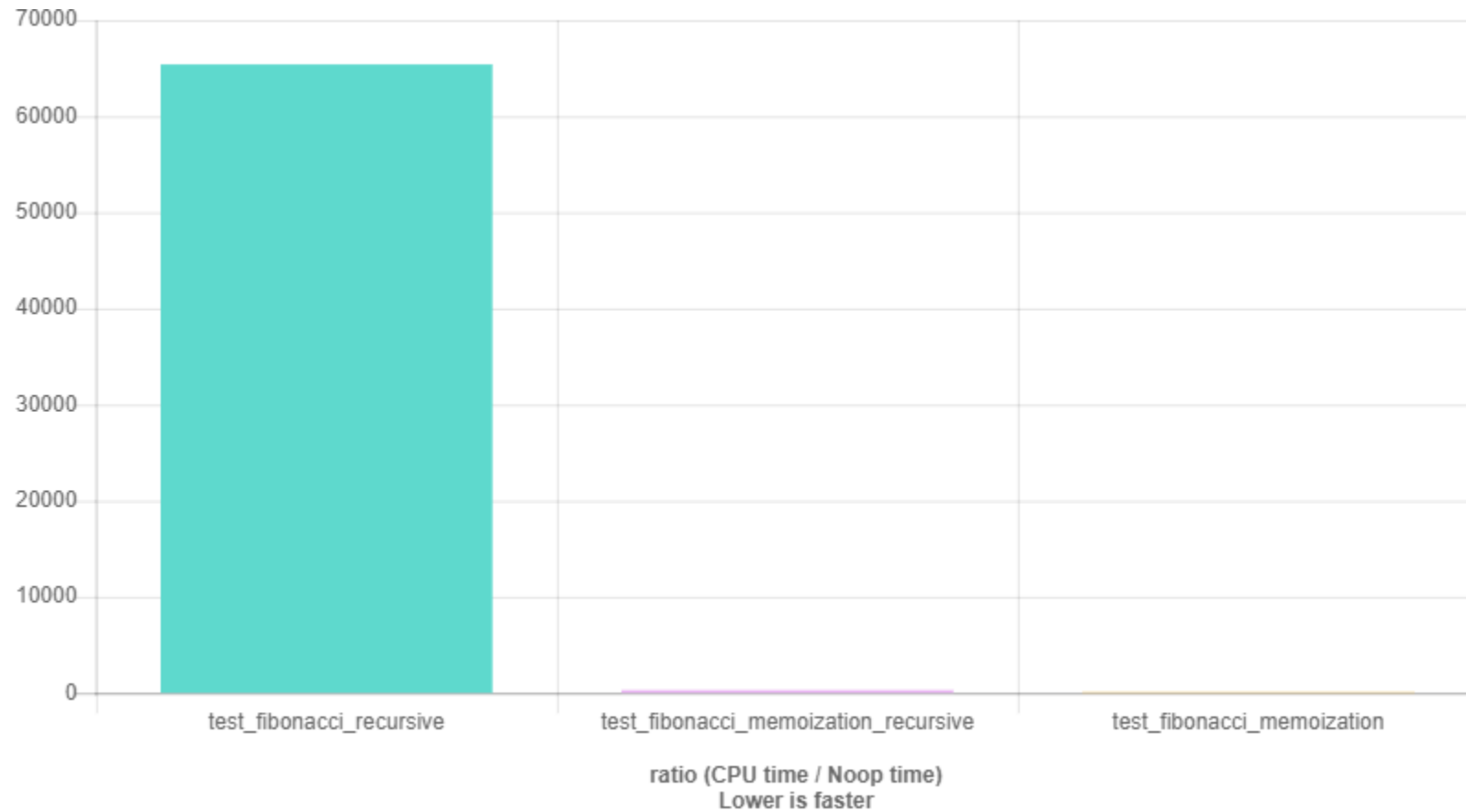


<http://quick-bench.com/f7xJnvvQF4ITEmBP4ZZ5qkBYaLA>

Ćwiczenie 15 – Fibonacci

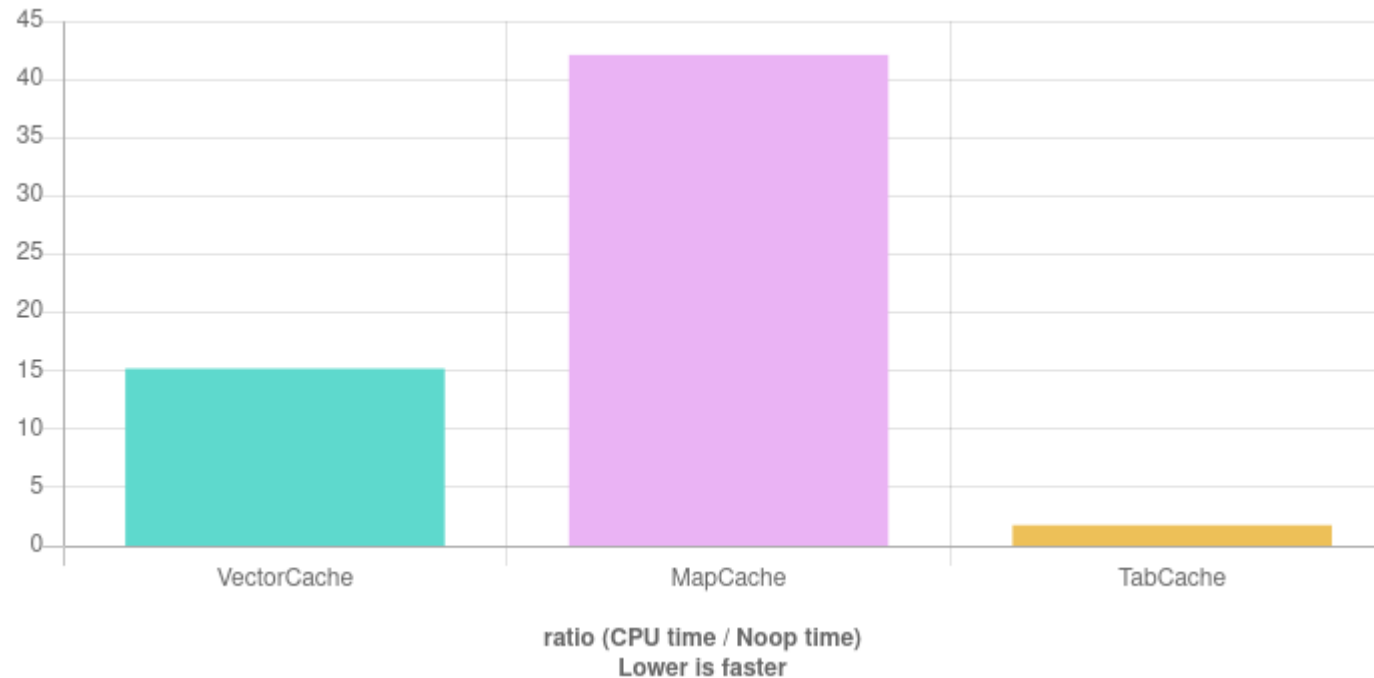
1. rekursja
2. rekursja z memoizacją
3. iteracja z memoizacją

Ćwiczenie 15 – Fibonacci



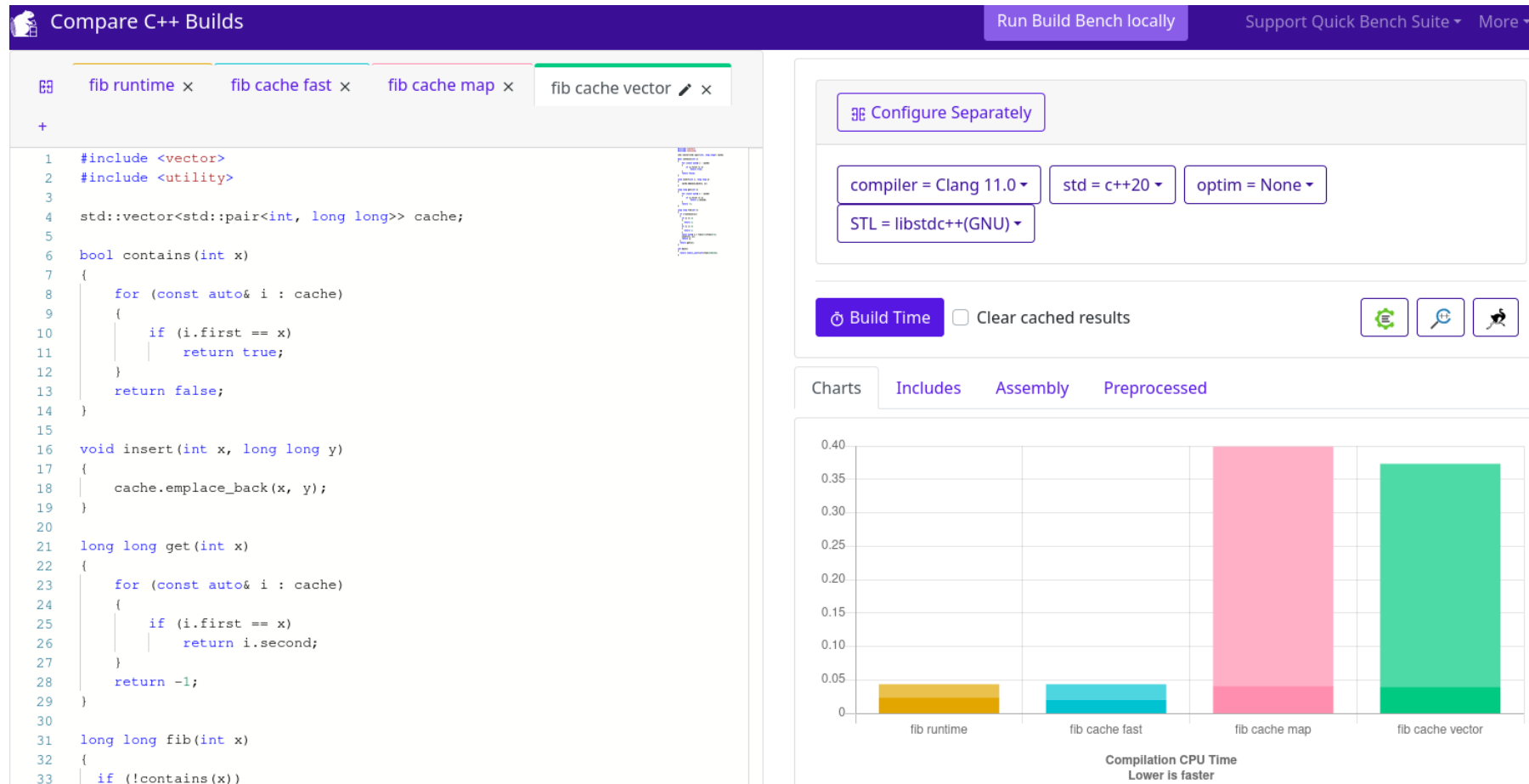
http://quick-bench.com/zTYodGv03fLo1n3_hVhyLCG530A

Ćwiczenie 16 – Fibonacci – sposoby memoizacji



<https://quick-bench.com/q/oheePHALIUF0txx2znyXaX-Hs9Q>

Build bench



<https://build-bench.com/b/EVfAaB0bU2-e0QSoR9PO2VofAME>

Przykłady – power runtime implementation

```
long long power(int x)
{
    if (x == 0)
    {
        return 1;
    }
    return power(x-1)*x;
}
```

```
int main()
{
    return static_cast<int>(power(13)%128);
}
```

Przykłady – power constexpr implementation

```
template <int x>
constexpr long long power()
{
    if constexpr(x == 0)
    {
        return 1;
    }
    else
    {
        return power<x-1>()*x;
    }
}

int main()
{
    return static_cast<int>(power<13>()%128);
}
```

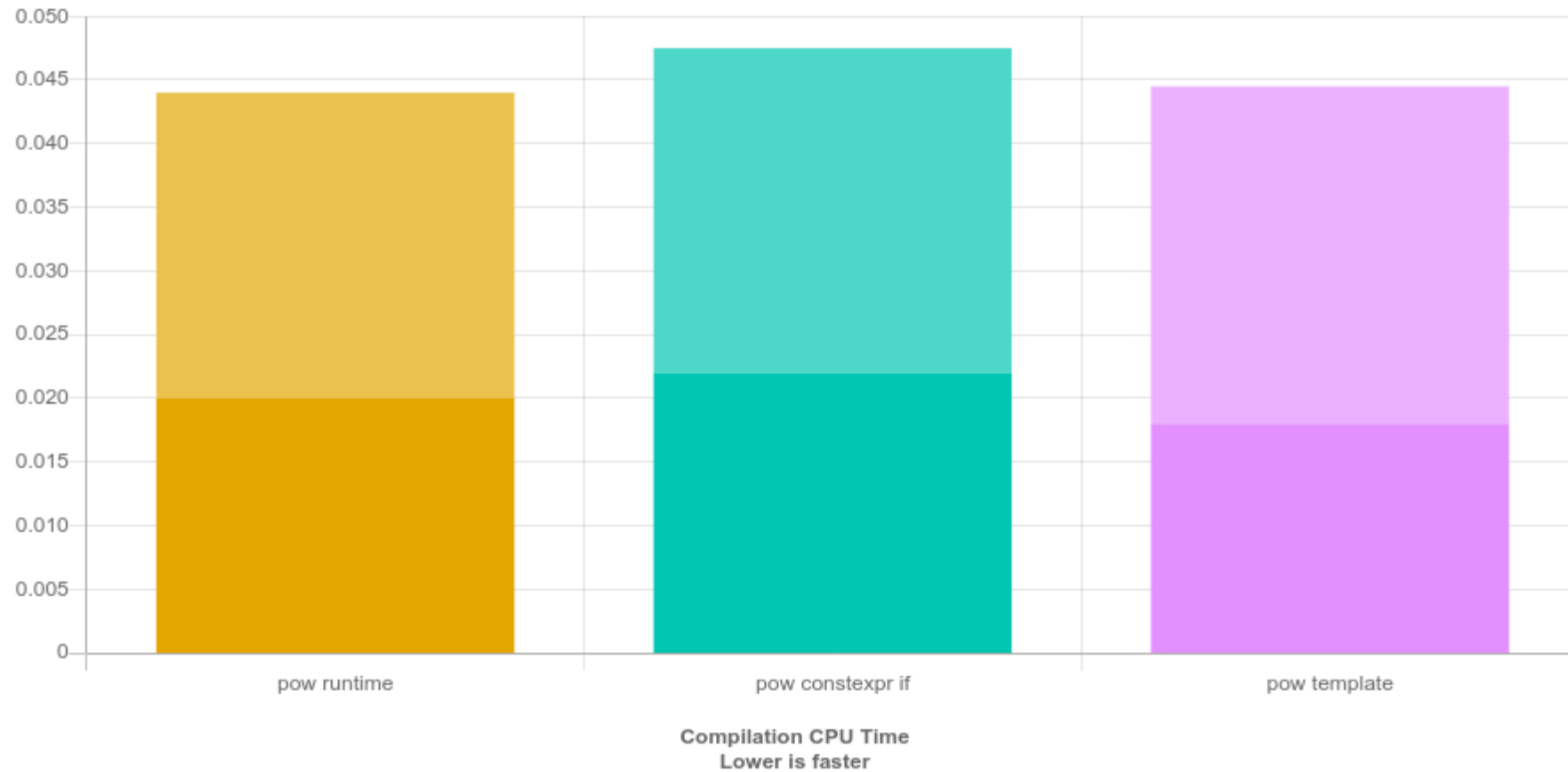
Przykłady – power template implementation

```
template <int x>
constexpr long long power()
{
    return power<x-1>()*x;
}
```

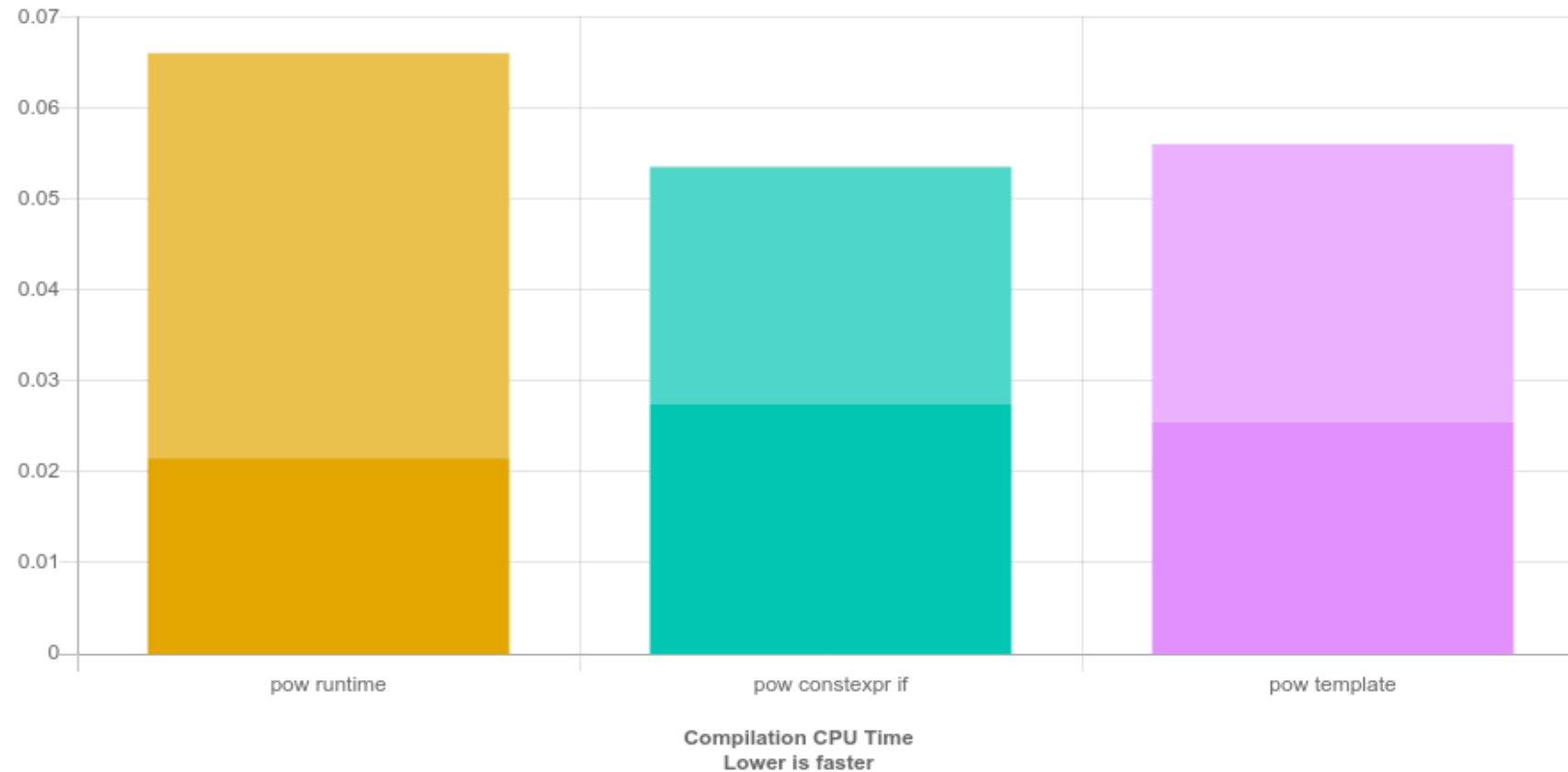
```
template<>
constexpr long long power<0>()
{
    return 1;
}
```

```
int main()
{
    return static_cast<int>(power<13>()%128);
}
```

Przykłady... clang-11, no optimization



Przykłady... clang-11, O3 optimization



Na co dodatkowo warto zwrócić uwagę?

- Ciężkie nagłówki
- Narzędzia:
 - Distcc
 - Ccache
 - Make/ninja

Ćwiczenie 16 – Semantyka przenoszenia

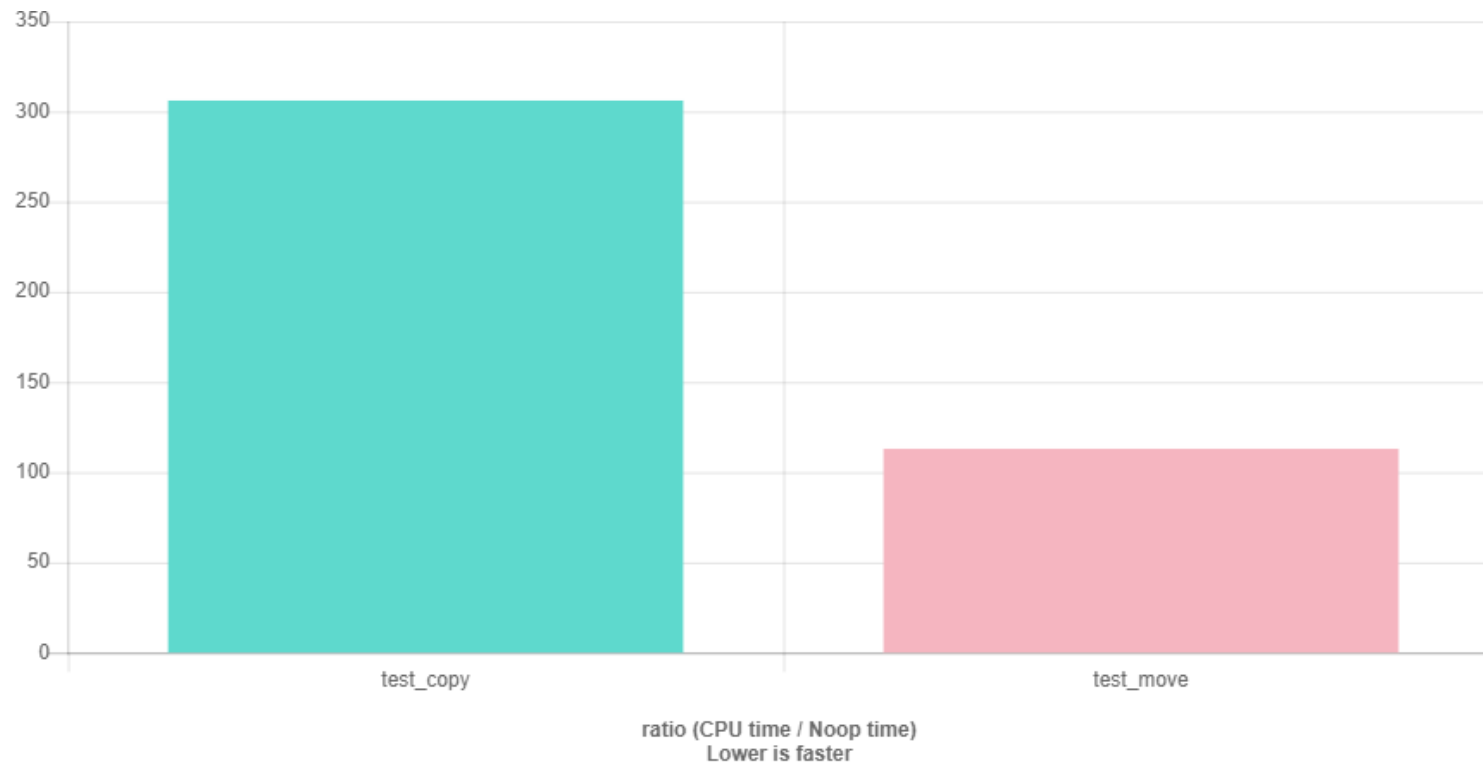
kopiowanie

przenoszenie

Ćwiczenie 16 – Semantyka przenoszenia

kopiowanie

przenoszenie



http://quick-bench.com/4_5K_1EI5BASflrHJqSGXjeqR3I

As-if rule

The as-if rule

Allows any and all code transformations that do not change the observable behavior of the program.

Reguła pozwala na dowolne transformacje kodu przez kompilator dopóki widoczne zachowanie programu się nie zmienia.

As-if rule – co to dla nas oznacza?

Zgodnie z zasadą, kompilator może:

- usuwać martwy kod,
- zmieniać kolejność wykonywania kodu,
- zmieniać kolejność dostępu do zmiennych,
- redukować wyrażenia,
- itp...

As-if rule – przykłady

```
11 void as_if_test()
12 {
13     { // this loop can be deleted
14         int end = 5;
15         while (end--)
16         {
17         }
18     }
19
20     { // this loop can't be deleted
21         volatile int end = 5;
22         while (end--)
23         {
24         }
25     }
26
27     { // this loop can't be deleted
28         int end = 5;
29         while (end++)
30         {
31         }
32     }
33 }
```

```
1 1 as_if_test():
2      mov     dword ptr [rsp - 4], 5
3 2 .LBB0_1:
4      sub     dword ptr [rsp - 4], 1
5      jae     .LBB0_1
6 3 .LBB0_2:
7      jmp     .LBB0_2
```

As-if rule – przykłady

```
35  int glob_a = 5;
36  int glob_b = 10;
37
38  int glob_arr[100];
39
40  void as_if_test2()
41  {
42      glob_a = 15;
43      glob_b = glob_a;
44
45      for (int i{}; i < 100; ++i)
46      {
47          glob_arr[i] = i;
48      }
49
50      glob_a = 10;
51  }
```

```
121  as_if_test2():                                # @as_if_test2()
122      vmovaps ymm0, ymmword ptr [rip + .LCPI1_0] # y
123      vmovaps ymmword ptr [rip + glob_arr], ymm0
124      vmovaps ymm0, ymmword ptr [rip + .LCPI1_1] # y
125      vmovaps ymmword ptr [rip + glob_arr+32], ymm0
126      vmovaps ymm0, ymmword ptr [rip + .LCPI1_2] # y
127      vmovaps ymmword ptr [rip + glob_arr+64], ymm0
128      vmovaps ymm0, ymmword ptr [rip + .LCPI1_3] # y
129      vmovaps ymmword ptr [rip + glob_arr+96], ymm0
130      vmovaps ymm0, ymmword ptr [rip + .LCPI1_4] # y
131      vmovaps ymmword ptr [rip + glob_arr+128], ymm0
132      vmovaps ymm0, ymmword ptr [rip + .LCPI1_5] # y
133      vmovaps ymmword ptr [rip + glob_arr+160], ymm0
134      vmovaps ymm0, ymmword ptr [rip + .LCPI1_6] # y
135      vmovaps ymmword ptr [rip + glob_arr+192], ymm0
136      vmovaps ymm0, ymmword ptr [rip + .LCPI1_7] # y
137      vmovaps ymmword ptr [rip + glob_arr+224], ymm0
138      vmovaps ymm0, ymmword ptr [rip + .LCPI1_8] # y
139      vmovaps ymmword ptr [rip + glob_arr+256], ymm0
140      vmovaps ymm0, ymmword ptr [rip + .LCPI1_9] # y
141      vmovaps ymmword ptr [rip + glob_arr+288], ymm0
142      vmovaps ymm0, ymmword ptr [rip + .LCPI1_10] # y
143      vmovaps ymmword ptr [rip + glob_arr+320], ymm0
144      vmovaps ymm0, ymmword ptr [rip + .LCPI1_11] # y
145      vmovaps ymmword ptr [rip + glob_arr+352], ymm0
146      mov     dword ptr [rip + glob_b], 15
147      vmovaps xmm0, xmmword ptr [rip + .LCPI1_12] # x
148      vmovaps xmmword ptr [rip + glob_arr+384], xmm0
149      mov     dword ptr [rip + glob_a], 10
150      vzeroupper
151      ret
```

As-if rule + inline'ing – przykłady

```
57  int glob_eval = 10;
58
59  void eval1()
60  {
61      glob_eval = 10 * 10 * glob_eval;
62  }
63
64  void eval2()
65  {
66      glob_eval = 10 * 10 * glob_eval;
67  }
68
69  void as_if_test3()
70  {
71      eval1();
72      eval2();
73  }
```

```
152  eval1():                                # @eval1()
153      imul    eax, dword ptr [rip + glob_eval], 100
154      mov     dword ptr [rip + glob_eval], eax
155      ret
156  eval2():                                # @eval2()
157      imul    eax, dword ptr [rip + glob_eval], 100
158      mov     dword ptr [rip + glob_eval], eax
159      ret
160  as_if_test3():                          # @as_if_test3()
161      imul    eax, dword ptr [rip + glob_eval], 10000
162      mov     dword ptr [rip + glob_eval], eax
163      ret
```


As-if rule – jak można przeciwdziałać?

```
79 volatile int a_disobey{10};
80
81 void as_if_disobey()
82 {
83     a_disobey = 10 * 10 * a_disobey;
84     a_disobey = 10 * 10 * a_disobey;
85 }
86
87 #include <atomic>
88
89 std::atomic<int> atomic_disobey{10};
90
91 void as_if_disobey_atomic()
92 {
93     atomic_disobey = 10 * 10 * atomic_disobey;
94     atomic_disobey = 10 * 10 * atomic_disobey;
95 }
```

```
164 as_if_disobey():                                # @as_if_disobey(
165     imul    eax, dword ptr [rip + a_disobey], 100
166     mov     dword ptr [rip + a_disobey], eax
167     imul    eax, dword ptr [rip + a_disobey], 100
168     mov     dword ptr [rip + a_disobey], eax
169     ret
170 as_if_disobey_atomic():                          # @as_if_disobey_
171     mov     eax, dword ptr [rip + atomic_disobey]
172     imul    eax, eax, 100
173     xchg    dword ptr [rip + atomic_disobey], eax
174     mov     eax, dword ptr [rip + atomic_disobey]
175     imul    eax, eax, 100
176     xchg    dword ptr [rip + atomic_disobey], eax
177     ret
```

As-if rule – `volatile` type qualifier

Dostęp do zmiennej typu `volatile` nie jest optymalizowany przez kompilator dowolnie. Użycie ma również wpływ na lokalny kontekst.

Z kwalifikatora głównie korzystamy w komunikacji z signal-handlerami oraz urządzeniami.

As-if rule – `std::atomic`

Dostęp do zmiennej typu `std::atomic` nie może być optymalizowany przez kompilator dowolnie.

Typ służy do komunikacji między wątkami. Wpływ na lokalny kontekst można wywierać poprzez `std::memory_order`.

As-if rule – Non TU local

As-if rule – złamanie (copy elision + RVO)

```
105 struct A
106 {
107     A()
108     {
109         std::cout << "created" << std::endl;
110     }
111
112     A(A &&)
113     {
114         std::cout << "moved" << std::endl;
115     }
116 };
117
118 [[gnu::noinline]] auto as_if_break()
119 {
120     return A(A(A(A(A(A(A())))))))
121 }
```

Output >> created

https://en.cppreference.com/w/cpp/language/copy_elision

<https://godbolt.org/z/69sGnjns1>

As-if rule – podsumowanie

Kompilator może wykonywać najróżniejsze optymalizacje dopóki potrafi udowodnić, że zachowanie programu się nie zmieni.

Jeśli nie ma takiej pewności to nie podejmuje agresywnych lub nawet jakichkolwiek optymalizacji.

Evaluation order – warto wiedzieć

Standard definiuje relację kolejności „sequenced-before”, która mówi o kolejności wykonywania obliczeń oraz efektów ubocznych wyrażeń.

Evaluation order definiuje jakie wyrażenia i w jaki sposób są ze sobą w relacji.

Evaluation order – przykład

```
85 int a_eval()
86 {
87     std::cout << "a_eval" << std::endl;
88
89     return 0;
90 }
91
92 int b_eval()
93 {
94     std::cout << "b_eval" << std::endl;
95
96     return 1;
97 }
98
99 int c_eval()
100 {
101     std::cout << "c_eval" << std::endl;
102
103     return 2;
104 }
105
106 void eval_tester(int a, int b, int c) {}
107
108 [[gnu::noinline]] void eval_order_test()
109 {
110     a_eval(), b_eval(), c_eval();
111     auto x = a_eval() * b_eval() * c_eval();
112     eval_tester(a_eval(), b_eval(), c_eval());
113 }
```

Clang 11.0.1

```
a_eval
b_eval
c_eval

a_eval
b_eval
c_eval

a_eval
b_eval
c_eval
```

Gcc 11.2

```
a_eval
b_eval
c_eval

a_eval
b_eval
c_eval

c_eval
b_eval
a_eval
```


Optymalizacja przez HW - autowektoryzacja

```
124 constexpr auto test_size = 64;
125
126 void add_vectors(float (&a)[test_size],
127                 float (&b)[test_size],
128                 float (&c)[test_size]) noexcept
129 {
130     for (int idx{}; idx < test_size; ++idx)
131     {
132         c[idx] = a[idx] + b[idx];
133     }
134 }
```

Jak to zrobić lepiej? - Strict aliasing rule

Kompilator musi założyć, że dwa wskaźniki podobnego typu mogą potencjalnie wskazywać na ten sam region pamięci.

Aby zastosować operacje typu SIMD, musi to wykluczyć. W przeciwnym przypadku może dojść do Undefined Behaviour.

Strict aliasing rule - przykład

```
167 [[gnu::noinline]] auto restrict_Normal(float *a, float *b)
168 {
169     *b = 1;
170     *a = 0;
171
172     return *b;
173 }
```

```
159 [[gnu::noinline]] auto restrict_UB(float *a, int *b)
160 {
161     *b = 1;
162     *a = 0;
163
164     return *b; // moves 1 instead of &b
165 }
```

```
701 restrict_Normal(float*, float*):
702     mov     dword ptr [rsi], 1065353216
703     mov     dword ptr [rdi], 0
704     vmovss  xmm0, dword ptr [rsi]
705     ret
```

```
696 restrict_UB(float*, int*):
697     mov     dword ptr [rsi], 1
698     mov     dword ptr [rdi], 0
699     mov     eax, 1
700     ret
```

Type qualifier - `restrict`

`restrict` mówi o tym, że dostęp do danego obiektu jest wykonywany jedynie przez ten konkretny wskaźnik/referencję.

Type qualifier - `restrict`

`restrict` nie jest częścią standardu C++, jedynie C. Mimo wszystko jest bardzo przydatny i praktycznie wszystkie kompilatory dodają go jako rozszerzenie.

Uwaga! Ze względu na to może powodować problemy, np. w dedukcji typów.

Autowektoryzacja – co z tym aliasingiem?

```
131 void add_vectors_restrict(float (&__restrict__ a)[test_size],  
132                          float (&__restrict__ b)[test_size],  
133                          float (&__restrict__ c)[test_size]) noexcept  
134 {  
135     for (int idx{}; idx < test_size; ++idx)  
136     {  
137         c[idx] = a[idx] + b[idx];  
138     }  
139 }
```

restrict – trzeba uważać!

```
146 void restrict_NoUB_vec(float *a, float *b) noexcept
147 {
148     for (int idx{}; idx < test_size; ++idx)
149     {
150         a[idx] += b[idx];
151     }
152 }
153
154 // can produce proper output when vectorization isn't applied :)
155 void restrict_UB_vec(float *__restrict__ a,
156                     float *__restrict__ b) noexcept
157 {
158     for (int idx{}; idx < test_size; ++idx)
159     {
160         a[idx] += b[idx];
161     }
162 }
```

Podsumowanie

C++ to potężne narzędzie, które daje duże pole do optymalizacji. Oczywiście jakość optymalizacji w dużej mierze zależy od kompilatora, ale sam język jest na tyle ekspresywny, że dostarcza dużo narzędzi potrzebnych do lepszego „porozumienia się” z kompilatorem.

Przez „porozumienie się” mamy na myśli dostarczanie większej ilości informacji dla kompilatora. To **MOŻE** zwiększyć prawdopodobieństwo otrzymania bardziej optymalnego kodu, ale to nie reguła. Kluczem do rozwiązania tego problemu jest dobra znajomość języka, znajomość sprzętu, profilowanie, testowanie oraz doświadczenie.

Profilowanie

Profilowanie

- Wyróżniamy 3 typy profilowania

Profilowanie

- Wyróżniamy 3 typy profilowania
 - Profilowanie zdarzeń (event based profiling) – alokacje, zawołania itd.

Profilowanie

- Wyróżniamy 3 typy profilowania
 - Profilowanie zdarzeń (event based profiling) – alokacje, zawołania itd.
 - Profilowanie statystyczne – sprawdzanie stosu z określoną częstotliwością.

Profilowanie

- Wyróżniamy 3 typy profilowania
 - Profilowanie zdarzeń (event based profiling) – alokacje, zawołania itd.
 - Profilowanie statystyczne – sprawdzanie stosu z określoną częstotliwością.
 - Instrumentalizacja kodu – wprowadzenie do programu dodatkowych funkcji odpowiedzialnych za zbieranie statystyk.

Profilowanie zdarzeń

Profilowanie zdarzeń

- Nie wymaga ingerencji w binarkę.

Profilowanie zdarzeń

- Nie wymaga ingerencji w binarkę.
- Wymaga środowiska które zapewni zliczanie zdarzeń.

Profilowanie zdarzeń

- Nie wymaga ingerencji w binarkę.
- Wymaga środowiska które zapewni zliczanie zdarzeń.
- Dodatkowe środowisko znacząco wpływa na czas wykonania.

Profilowanie zdarzeń

- Nie wymaga ingerencji w binarkę.
- Wymaga środowiska które zapewni zliczanie zdarzeń.
- Dodatkowe środowisko znacząco wpływa na czas wykonania.
- Najpopularniejszy profiler zdarzeniowy dla C i C++: valgrind.

valgrind

valgrind

- Zestaw narzędzi służących do debugowania i profilowania aplikacji odpalanych w „piaskownicy”.

valgrind

- Zestaw narzędzi służących do debugowania i profilowania aplikacji odpalanych w „piaskownicy”.
- Najpopularniejsze narzędzie do sprawdzania wycieków pamięci.

valgrind

- Zestaw narzędzi służących do debugowania i profilowania aplikacji odpalanych w „piaskownicy”.
- Najpopularniejsze narzędzie do sprawdzania wycieków pamięci.
- Zapewnia narzędzia do profilowania użycia pamięci cache, profilowania wywołań funkcji, błędów wątków, zużycia pamięci dynamicznej.

Przykładowe użycie valgrind'a

Przykładowe użycie valgrind'a

- `valgrind ls -a`

Przykładowe użycie valgrind'a

- `valgrind ls -a`
- `valgrind -tool=callgrind ls -a`

Przykładowe użycie valgrind'a

- `valgrind ls -a`
- `valgrind -tool=callgrind ls -a`
- W przypadku użycia callgrind'a powstanie plik `callgrind.out.[PID]`, którego możemy użyć w narzędziach do interpretacji takich jak `kcachegrind`, `callgrind_annotate`, czy `gprof2dot`.

kcachegrind – przykład

./callgrind.out [/home/pawluch/CLionProjects/paro_profiling/cmake-build-debug/paro_profiling] — KCachegrind

File View Go Settings Help

Open... Back Forward Up Relative Cycle Detection Relative to Parent Instruction Fetch

Flat Profile

Search: Search Query (No Grouping)

Incl.	Self	Called	Function
91.74	0.00	(0)	0x00000000
59.01	0.00	1	_start
59.01	0.00	1	(below main)
58.50	0.00	1	main
38.33	1.47	1	auto make_data<int, 10000ul>() (paro_profiling: main.cpp)
32.08	0.01	1	_dl_start
32.07	0.01	1	_dl_sysdep_start (ld-2.29.so)
31.85	0.01	1	dl_main
31.17	4.15	8	_dl_relocate_static_initial (ld-2.29.so)
27.27	12.82	3 052	_dl_lookup_symbol_x (ld-2.29.so)
21.48	1.56	10 000	std::vector<int, std::allocator<int>>::operator[] (std::vector<int, std::allocator<int>>:1)
19.33	3.42	10 000	int& std::vector<int, std::allocator<int>>::operator[] (std::vector<int, std::allocator<int>>:1)
16.86	0.00	1	fibonacci_recursive(unsigned int) (paro_profiling: main.cpp)
16.86	16.86	92 734	fibonacci_recursive(unsigned int) (paro_profiling: main.cpp)
14.80	1.27	10 000	int std::uniform_int_distribution<int>::operator[] (std::uniform_int_distribution<int>:1)
14.45	10.26	3 052	do_lookup_x (ld-2.29.so)
13.53	4.69	10 000	int std::uniform_int_distribution<int>::operator[] (std::uniform_int_distribution<int>:1)
9.67	2.44	10 000	std::vector<int, std::allocator<int>>::operator[] (std::vector<int, std::allocator<int>>:1)
8.26	0.03	40	start_thread (libc-2.29.so)
8.13	0.04	81	__pthread_once (libc-2.29.so)
7.95	0.01	101	pthread_once (libc-2.29.so)

main

Types Callers All Callers Callee Map Source Code

Ir	Ir per call	Count	Caller
58.50	5 990 636	1	(below main) (libc-2.29.so)

Ir	Ir per call	Count	Callee
38.33	3 924 933	1	auto make_data<int, 10000ul>() (paro_profiling: main.cpp)
16.86	1 726 539	1	fibonacci_recursive(unsigned int) (paro_profiling: main.cpp)
3.14	321 292	1	auto parallel_sum<__gnu_cxx::__normal_iterator<int const*, std::vector<int, std::allocator<int>>>, int> (paro_profiling: main.cpp)
0.08	2 050	4	_dl_runtime_resolve_xsave (ld-2.29.so)
0.05	4 893	1	fibonacci_memoization_recursive(unsigned int) (paro_profiling: main.cpp)
0.02	2 233	1	fibonacci_memoization(unsigned int) (paro_profiling: main.cpp)
0.01	394	3	std::ostream::operator<<(std::ostream& (*) (std::ostream&)) (libstdc++.so.6.0.25: ostream)
0.01	446	2	std::ostream::operator<<(unsigned int) (libstdc++.so.6.0.25: ostream)
0.00	301	1	std::vector<int, std::allocator<int>>::~vector() (paro_profiling: stl_vector.h)
0.00	32	1	std::vector<int, std::allocator<int>>::~end() const (paro_profiling: stl_vector.h)
0.00	32	1	std::vector<int, std::allocator<int>>::~begin() const (paro_profiling: stl_vector.h)

Parts Callees Call Graph All Callees Caller Map Machine Code

callgrind.out [1] - Total Instruction Fetch Cost: 10 240 815

Profilowanie statystyczne

Profilowanie statystyczne

- Nie wymaga ingerencji w binarkę.

Profilowanie statystyczne

- Nie wymaga ingerencji w binarkę.
- Szybkie.

Profilowanie statystyczne

- Nie wymaga ingerencji w binarkę.
- Szybkie.
- Mniej dokładne.

perf

perf

- Dostępny od jądra 2.6.31.

perf

- Dostępny od jądra 2.6.31.
- Większość funkcjonalności jest zintegrowane z jądrem.

Przykładowe użycie perf'a

Przykładowe użycie perf'a

- Zebranie danych
 - `perf record -g -F 7500 1s`

Przykładowe użycie perf'a

- Zebranie danych
 - `perf record -g -F 7500 ls`
- Analiza danych
 - `perf report -g 'graph,0.5,caller'`

perf – przykład

```
Samples: 2K of event 'cycles:u', Event count (approx.): 6815962
  Children   Self  Command      Shared Object      Symbol
+  41.89%    0.00%  paro_profiling [unknown]          [.] 0x5541f689495641d7
+  41.89%    0.00%  paro_profiling libc-2.29.so        [.] __libc_start_main
-  41.82%    0.00%  paro_profiling paro_profiling      [.] main
- main
+ 24.73% make_data<int, 10000ul>
+ 10.23% fibonacci_recursive
+ 6.24% parallel_sum<__gnu_cxx::__normal_iterator<int const*, std::vector<int, std::
+ 24.73%    1.67%  paro_profiling paro_profiling      [.] make_data<int, 10000ul>
+ 13.94%    0.00%  paro_profiling [unknown]          [.] 0x000cea903d8d4866
+ 13.44%    0.04%  paro_profiling libpthread-2.29.so [.] __pthread_once_slow
+ 12.90%    0.02%  paro_profiling paro_profiling      [.] std::call_once<void (std::
+ 12.88%    0.00%  paro_profiling paro_profiling      [.] std::call_once<void (std::
+ 12.82%    0.02%  paro_profiling paro_profiling      [.] std::call_once<void (std::
+ 12.78%    0.79%  paro_profiling paro_profiling      [.] std::uniform_int_distribut
+ 12.72%    0.04%  paro_profiling paro_profiling      [.] std::__invoke<void (std::
+ 12.69%    0.15%  paro_profiling paro_profiling      [.] std::__invoke_impl<void, vo
+ 12.68%   12.68%  paro_profiling ld-2.29.so          [.] do_lookup_x
+ 12.54%    0.08%  paro_profiling paro_profiling      [.] std::__future_base::_State
+ 11.99%    0.09%  paro_profiling paro_profiling      [.] std::function<std::unique_p
+ 11.89%    0.02%  paro_profiling paro_profiling      [.] std::_Function_handler<std:
+ 11.78%    2.22%  paro_profiling paro_profiling      [.] std::uniform_int_distribut
+ 11.30%    0.00%  paro_profiling paro_profiling      [.] std::__future_base::_Task_s
+ 10.46%    0.00%  paro_profiling paro_profiling      [.] std::thread::_Invoker<std::
```

Ciekawe przykłady

- count bitset:
 - <https://gcc.godbolt.org/z/PwxhPu>
- sumTo:
 - https://gcc.godbolt.org/z/JMaE_o

Podsumowanie

Podsumowanie

- Najpierw algorytm.

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,
 - akceptowalne kompromisy.

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,
 - akceptowalne kompromisy.
- Gmatwanie kodu rzadko pomaga z wydajnością.

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,
 - akceptowalne kompromisy.
- Gmatwanie kodu rzadko pomaga z wydajnością.
- Za to często przeszkadza

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,
 - akceptowalne kompromisy.
- Gmatwanie kodu rzadko pomaga z wydajnością.
- Za to często przeszkadza
 - kompilatorowi w optymalizowaniu,

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,
 - akceptowalne kompromisy.
- Gmatwanie kodu rzadko pomaga z wydajnością.
- **Za to często przeszkadza**
 - kompilatorowi w optymalizowaniu,
 - ludziom (w tym Tobie!) w rozumieniu.

Podsumowanie

- Najpierw algorytm.
- Zrozum swoją dziedzinę
 - oczekiwane wejścia,
 - akceptowalne kompromisy.
- Gmatwanie kodu rzadko pomaga z wydajnością.
- Za to często przeszkadza
 - kompilatorowi w optymalizowaniu,
 - ludziom (w tym Tobie!) w rozumieniu.
- **Testuj!**

Patrz też

1. [CppCon 2014: Andrei Alexandrescu "Optimization Tips - Mo' Hustle Mo' Problems"](#)
2. [code::dive conference 2015 - Andrei Alexandrescu - Writing Fast Code I](#)
3. [code::dive conference 2015 - Andrei Alexandrescu - Writing Fast Code II](#)
4. [There's Treasure Everywhere - Andrei Alexandrescu](#)
5. [Fastware - Andrei Alexandrescu](#)
6. [CppCon 2015: Chandler Carruth "Tuning C++: Benchmarks, and CPUs, and Compilers! Oh My!,,](#)
7. [CppCon 2017: Matt Godbolt "What Has My Compiler Done for Me Lately? Unbolting the Compiler's Lid"](#)
8. [CppCon 2018: Nir Friedman "Understanding Optimizers: Helping the Compiler Help You"](#)
9. [Type punning done right - Andreas Weis - Lightning Talks Meeting C++ 2017](#)
10. [Performance Tools Developments](#)