

How to use Compiler Explorer to make easier auto-vectorization

Link

❖ <https://godbolt.org/>

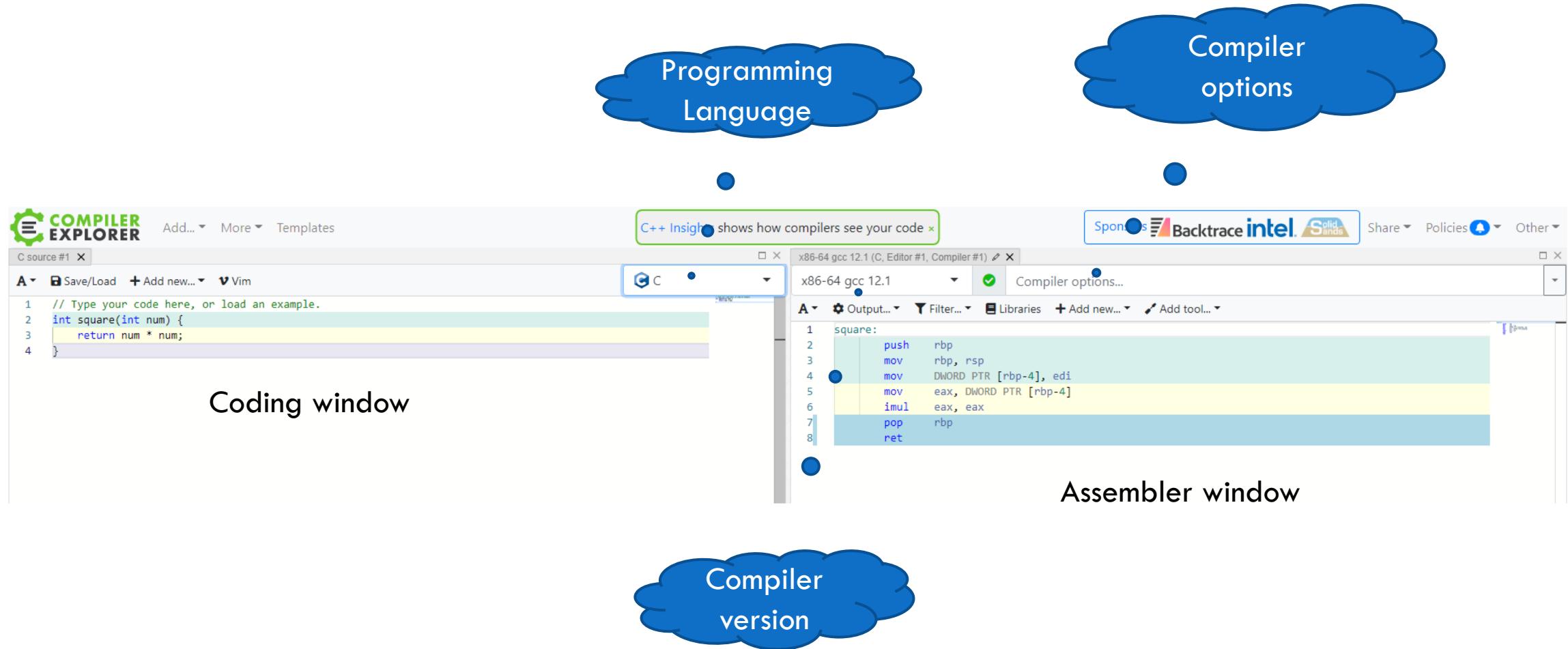
The screenshot shows the Compiler Explorer interface. On the left, the C++ source code is displayed:

```
1 #include <cmath>
2
3 double BlackBoxFunction(const double x) {
4     return 1.0/sqrt(x);
5 }
6
7
8 double ComputeIntegral(const int n, const double a, const double b) {
9     const double dx = (b - a)/n;
10    double I = 0.0;
11    double d_i = 0.0;
12    for (int i = 0; i < n; i++) {
13        const double xip12 = a + dx*(d_i + 0.5);
14        d_i = d_i + 1.0;
15        const double yip12 = BlackBoxFunction(xip12);
16        const double dI = yip12*dx;
17        I += dI;
18    }
19    return I;
20 }
```

On the right, the assembly output for x86-64 gcc 8.2 is shown:

```
1 BlackBoxFunction(double):
2     vsqrtsd xmm1, xmm1, xmm0
3     vmovsd xmm0, QWORD PTR .LC0[rip]
4     vdivsd xmm0, xmm0, xmm1
5     ret
6 ComputeIntegral(int, double, double):
7     vxorpd xmm2, xmm2, xmm2
8     vsubsd xmm1, xmm1, xmm0
9     vcvtts2sd    xmm2, xmm2, edi
10    vdivsd xmm1, xmm1, xmm2
11    test    edi, edi
12    jle     .L9
13    lea     eax, [rdi-1]
14    cmp     eax, 2
15    jbe     .L10
16    mov     edx, edi
17    vmovapd ymm4, YMMWORD PTR .LC1[rip]
18    xor     eax, eax
19    vxorpd ymm3, xmm3, xmm3
20    vmovapd ymm8, YMMWORD PTR .LC3[rip]
21    shr     edx, 2
22    vbroadcastsd ymm5, xmm1
23    vbroadcastsd ymm9, xmm0
24    vmovapd ymm7, YMMWORD PTR .LC4[rip]
25    vmovapd ymm6, YMMWORD PTR .LC5[rip]
26 .L7:
27    vaddpd ymm2, ymm4, ymm7
28    inc     eax
29    vaddpd ymm4, ymm4, ymm8
30    vfmaadd132pd ymm2, ymm9, ymm5
31    vsqrtpd ymm2, ymm2
32    vdivpd ymm2, ymm6, ymm2
33    vfmaadd231pd ymm3, ymm5, ymm2
34    cmp     eax, edx
35    jne     .L7
36    vhaddpd ymm3, ymm3, ymm3
37    mov     eax, edi
```

Select prog. language and compiler version



Auto-vectorización

The screenshot shows the Compiler Explorer interface with the following components:

- Compiler options:** A blue cloud-shaped callout pointing to the command line arguments: `-O3 -march=native -fopt-info-vec-all`.
- simple2.c:** A blue cloud-shaped callout pointing to the source code file.
- C++ Insights:** A green bar at the top of the editor window.
- Output window:** A blue cloud-shaped callout pointing to the bottom pane displaying compiler messages and statistics.
- Code Editor:** The main window showing the C source code `simple2.c`. The code performs operations on arrays `a`, `b`, and `c`. The output window shows the compiler's analysis and optimization decisions.

simple2.c

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 #define ARRAY_SIZE 2048
5 #define NUMBER_OF_TRIALS 1000000
6
7 /*
8  * Statically allocate our arrays. Compilers can
9  * align them correctly.
10 */
11 static double a[ARRAY_SIZE], b[ARRAY_SIZE], c;
12
13 int main(int argc, char *argv[]) {
14     int i;
15
16     double m = 1.0001;
17
18     /* Populate A and B arrays */
19     for (i=0; i < ARRAY_SIZE; i++) {
20         b[i] = i;
21         a[i] = i+1;
22     }
23
24     /* Perform an operation a number of times */
25     for (t=0; t < NUMBER_OF_TRIALS; t++) {
26         for (i=0; i < ARRAY_SIZE; i++) {
27             c += m*a[i] + b[i];
28         }
29     }
30
31     return 0;
32 }
```

C++ Insights shows how compilers see your code

x86-64 gcc 12.1 (C, Editor #1, Compiler #1)

Output (0/8) x86-64 gcc 12.1

Output of x86-64 gcc 12.1 (Compiler #1)

Compiler returned: 0

Compiler options: -O3 -march=native -fopt-info-vec-all

Output window:

```
<source>:25:17: missed: couldn't vectorize loop
<source>:27:21: missed: not vectorized: no vectype for stmt: _4 = a[i_29];
scalar_type: double
<source>:26:21: optimized: loop vectorized using 32 byte vectors
<source>:19:17: optimized: loop vectorized using 32 byte vectors
<source>:13:5: note: vectorized 2 loops in function.
<source>:25:17: note: **** Analysis failed with vector mode V4DF
<source>:25:17: note: **** Skipping vector mode V32QI, which would repeat the analysis for V4DF
```

Click on a message to see the related line of code

Greyscale Autovectorization

No changes. No auto-vectorization

The screenshot shows the Compiler Explorer interface with two tabs: 'C source #1' and 'x86-64 gcc 12.1'. The C source code contains functions for reading RGB values from an image and calculating grayscale values. The assembly output shows the generated x86-64 assembly code for the grayscale calculation loop. A red box highlights a note in the compiler's diagnostic output indicating that no loops were vectorized.

```
#include <stdio.h>
#include <stdint.h>
#include <math.h>

static inline void getRGB(uint8_t *im, int width, int height, int nchannels, int x, int y, int *r, int *g, int *b)
{
    unsigned char *offset = im + (x + width * y) * nchannels;
    *r = offset[0];
    *g = offset[1];
    *b = offset[2];
}

void computegreyScale(uint8_t *rgb_image, uint8_t *grey_image, int width, int height) {
    int r, g, b;
    for (int i = 0; i < width; i++)
    {
        for (int j = 0; j < height; j++)
        {
            getRGB(rgb_image, width, height, 4, i, j, &r, &g, &b);
            grey_image[j * width + i] = (int)(0.2989 * r + 0.5870 * g + 0.1140 * b);
        }
    }
}
```

```
computegreyScale:
    test    edx, edx
    jle    .L8
    mov    r8d, ecx
    test    ecx, ecx
    jle    .L8
    vmovsd xmm5, QWORD PTR .LC0[rip]
    vmovsd xmm4, QWORD PTR .LC1[rip]
    mov    r9, rsi
    movsx  rsi, edx
    vmovsd xmm3, QWORD PTR .LC2[rip]
    sal    edx, 2
    mov    r11, rdi
    push   rbx
    vxorps xmm2, xmm2, xmm2
    movsx  rdi, edx
    xor    r10d, r10d

Output (0/7) x86-64 gcc 12.1 - cached (18954B) ~1139 lines filtered
Output of x86-64 gcc 12.1 (Compiler #1)
<source>:16:23: missed: couldn't vectorize loop
<source>:9:16: missed: not vectorized: not suitable for strided load _Z30 = *offset_29;
<source>:18:27: missed: couldn't vectorize loop
<source>:21:54: missed: not vectorized: unsupported data-type double
<source>:14:6: note: vectorized 0 loops in function.
<source>:24:1: note: **** Analysis failed with vector mode V32QI
<source>:24:1: note: ***** Skipping vector mode V32QI, which would repeat the analysis for V32QI
Compiler returned: 0
```

After changing the order of the loops

The screenshot shows the Compiler Explorer interface with the following components:

- Left Panel (Code Editor):** Displays the C source code for a grayscale conversion function. Two loops are highlighted with red boxes:
 - The outer loop (j) runs from 0 to height.
 - The inner loop (i) runs from 0 to width.
- Middle Panel (Assembly View):** Shows the generated assembly code for the `computegreyScale` function. The assembly is color-coded by section:
 - Red: Function prologue (push rbp, mov rbp, push r15, etc.)
 - Blue: Main loop body (test ecx, ecx, jle .L22, mov r11, rdi, mov edi, edx, test edx, edx, jle .L22)
 - Green: Epilogue (and rsp, -32, sub rsp, 104, mov DWORD PTR [rsp-64], ecx)
- Bottom Panel (Compiler Log):** Shows the compiler's output log. It includes messages about missed vectorization opportunities due to complicated access patterns and aliasing, as well as notes about successfully vectorized loops (e.g., "note: vectored 1 loops in function").