Laboratorio 2

Alvaro Frias Garay - Ary Lautaro Di Bartolo

Universidad Nacional de Córdoba - Universidad Nacional de Cuyo

2021

Autovectorización

```
gcc -O1 -ftree-vectorize -fopt-info-vec -fopt-info-vec-missed
wtime.c mtwister.c tiny_mc.c -lm
```

Autovectorización

- tiny_mc.c:107:5: missed: couldn't vectorize loop

Autovectorización

- tiny_mc.c:107:5: missed: couldn't vectorize loop
- tiny_mc.c:107:5: missed: not vectorized: multiple nested loops.

Autovectorización

- tiny_mc.c:107:5: missed: couldn't vectorize loop
- tiny_mc.c:107:5: missed: not vectorized: multiple nested loops.
- tiny_mc.c:78:16: missed: couldn't vectorize loop

Autovectorización

- tiny_mc.c:107:5: missed: couldn't vectorize loop
- tiny_mc.c:107:5: missed: not vectorized: multiple nested loops.
- tiny_mc.c:78:16: missed: couldn't vectorize loop
- tiny_mc.c:78:16: missed: not vectorized: control flow in loop.

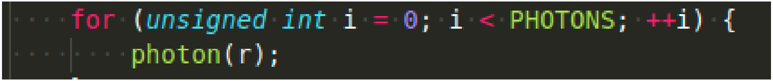
Autovectorización

- tiny_mc.c:107:5: missed: couldn't vectorize loop
- tiny_mc.c:107:5: missed: not vectorized: multiple nested loops.
- tiny_mc.c:78:16: missed: couldn't vectorize loop
- tiny_mc.c:78:16: missed: not vectorized: control flow in loop.
- tiny_mc.c:71:9: missed: couldn't vectorize loop

Autovectorización

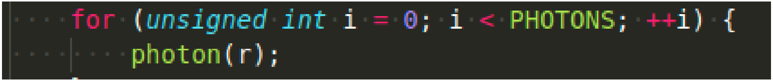
- tiny_mc.c:107:5: missed: couldn't vectorize loop
- tiny_mc.c:107:5: missed: not vectorized: multiple nested loops.
- tiny_mc.c:78:16: missed: couldn't vectorize loop
- tiny_mc.c:78:16: missed: not vectorized: control flow in loop.
- tiny_mc.c:71:9: missed: couldn't vectorize loop
- tiny_mc.c:71:9: missed: not vectorized: number of iterations cannot be computed.

Autovectorización

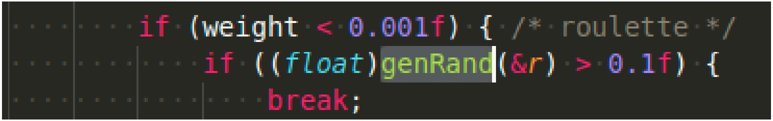


```
    for (unsigned int i = 0; i < PHOTONS; ++i) {
        photon(r);
```

Autovectorización

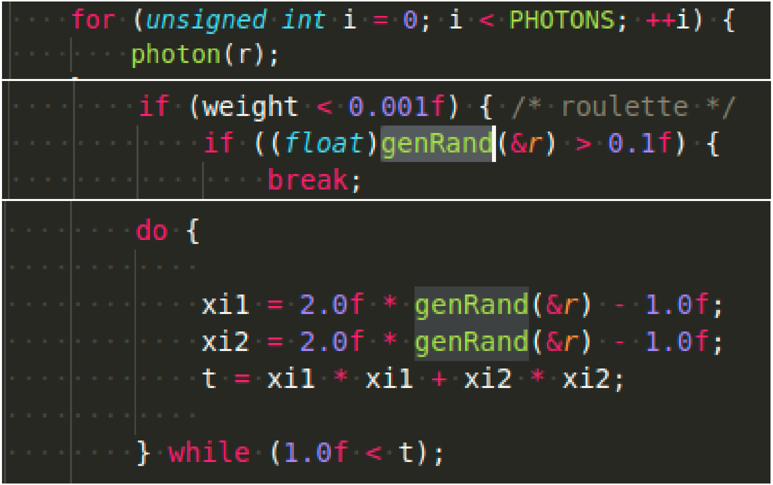


```
    for (unsigned int i = 0; i < PHOTONS; ++i) {
        photon(r);
```



```
        if (weight < 0.001f) { /* roulette */
            if ((float)genRand(&r) > 0.1f) {
                break;
```

Autovectorización



```
    for (unsigned int i = 0; i < PHOTONS; ++i) {
        photon(r);
```

```
        if (weight < 0.001f) { /* roulette */
            if ((float)genRand(&r) > 0.1f) {
                break;
```

```
        do {


            xi1 = 2.0f * genRand(&r) - 1.0f;
            xi2 = 2.0f * genRand(&r) - 1.0f;
            t = xi1 * xi1 + xi2 * xi2;


        } while (1.0f < t);
```

Vectorización Manual

Vectorización Manual

- Intel Intrisinsics

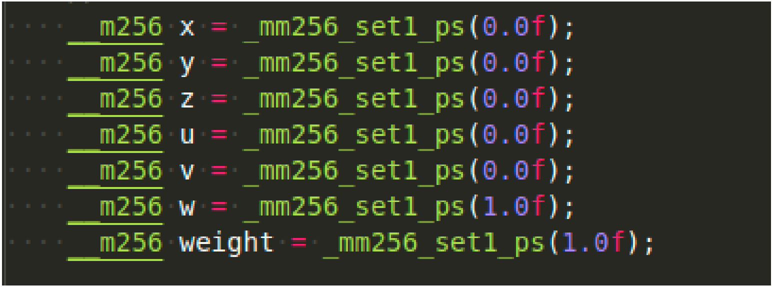
Vectorización Manual

- Intel Intrisinsics
- AVX / AVX-2

Vectorización Manual

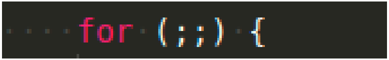
- Intel Intrisinsics
- AVX / AVX-2
- Vectores de 256 bits i.e 8 floats

Detalle de la vectorización



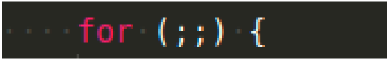
```
    __m256 x = _mm256_set1_ps(0.0f);
    __m256 y = _mm256_set1_ps(0.0f);
    __m256 z = _mm256_set1_ps(0.0f);
    __m256 u = _mm256_set1_ps(0.0f);
    __m256 v = _mm256_set1_ps(0.0f);
    __m256 w = _mm256_set1_ps(1.0f);
    __m256 weight = _mm256_set1_ps(1.0f);
```

Detalle de la vectorización

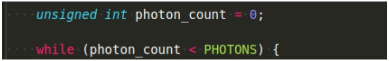


```
    for (;;) {
```

Detalle de la vectorización

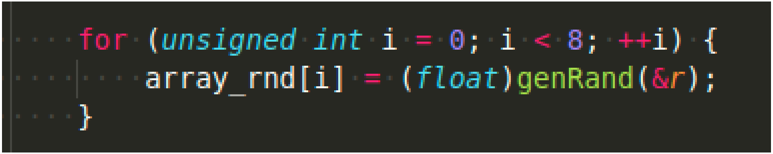


```
    for (;;) {
```



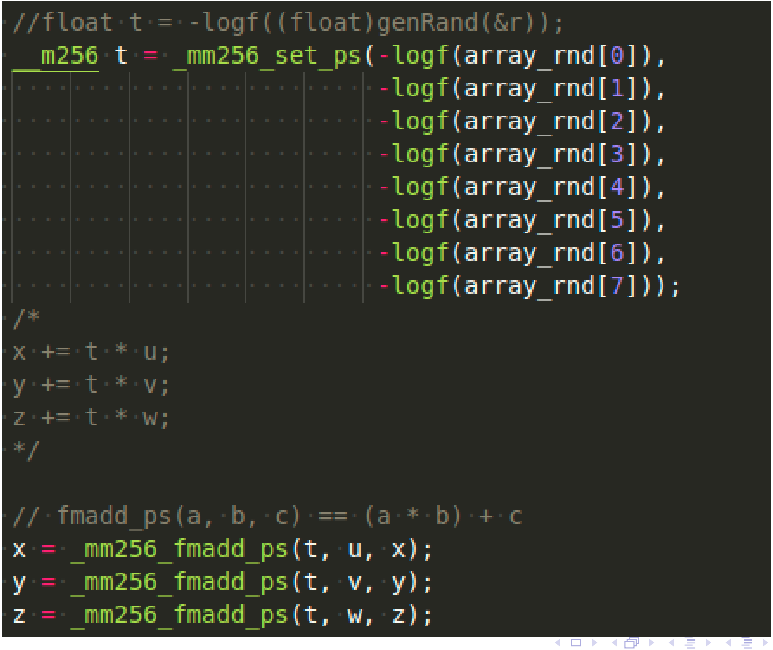
```
    unsigned int photon_count = 0;

    while (photon_count < PHOTONS) {
```

Detalle de la vectorización



```
    for (unsigned int i = 0; i < 8; ++i) {
        array_rnd[i] = (float)genRand(&r);
    }
```

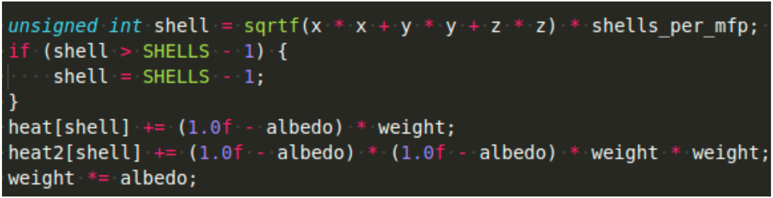
Detalle de la vectorización



```c
//float t = -logf((float)genRand(&r));
__m256 t = _mm256_set_ps(-logf(array_rnd[0]),
                         -logf(array_rnd[1]),
                         -logf(array_rnd[2]),
                         -logf(array_rnd[3]),
                         -logf(array_rnd[4]),
                         -logf(array_rnd[5]),
                         -logf(array_rnd[6]),
                         -logf(array_rnd[7]));
/*
x += t * u;
y += t * v;
z += t * w;
*/

// fmadd_ps(a, b, c) == (a * b) + c
x = _mm256_fmadd_ps(t, u, x);
y = _mm256_fmadd_ps(t, v, y);
z = _mm256_fmadd_ps(t, w, z);
```

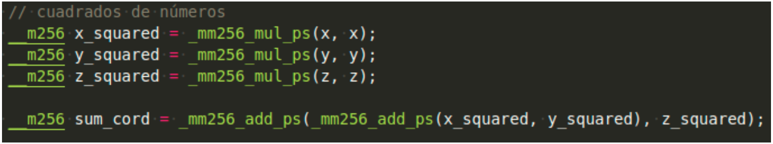
Detalle de la vectorización

tinymc sin vectorizar



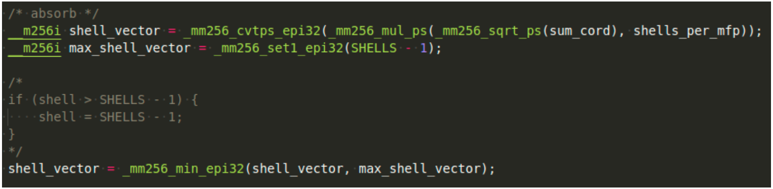
```c
unsigned int shell = sqrtf(x * x + y * y + z * z) * shells_per_mfp;
if (shell > SHELLS - 1) {
    shell = SHELLS - 1;
}
heat[shell] += (1.0f - albedo) * weight;
heat2[shell] += (1.0f - albedo) * (1.0f - albedo) * weight * weight;
weight *= albedo;
```

Detalle de la vectorización



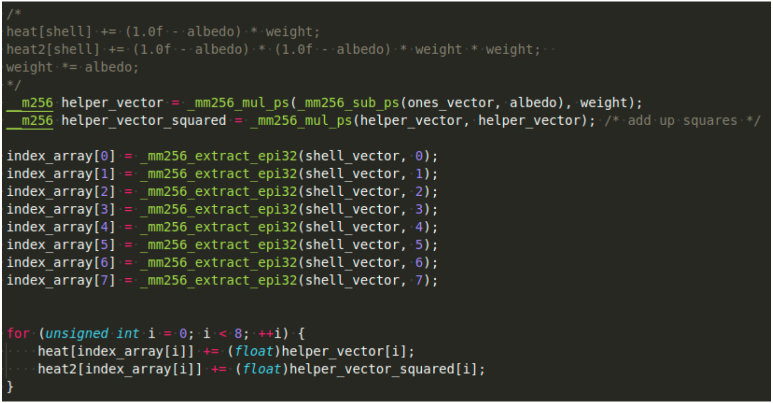
```c
// cuadrados de números
__m256 x_squared = _mm256_mul_ps(x, x);
__m256 y_squared = _mm256_mul_ps(y, y);
__m256 z_squared = _mm256_mul_ps(z, z);

__m256 sum_cord = _mm256_add_ps(_mm256_add_ps(x_squared, y_squared), z_squared);
```

Detalle de la vectorización



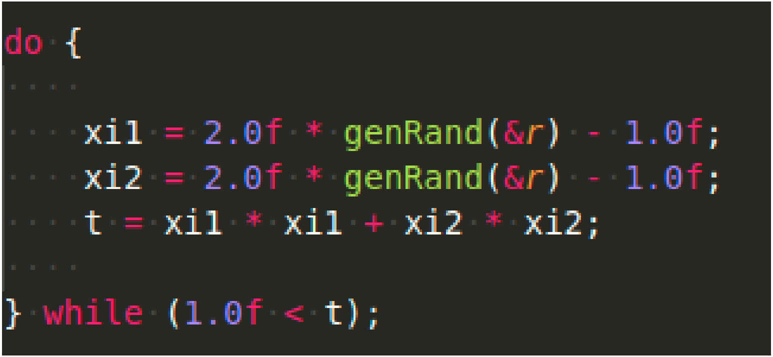
```
/* absorb */
__m256i shell_vector = _mm256_cvtps_epi32(_mm256_mul_ps(_mm256_sqrt_ps(sum_cord), shells_per_mfp));
__m256i max_shell_vector = _mm256_set1_epi32(SHELLS - 1);

/*
if (shell > SHELLS - 1) {
    shell = SHELLS - 1;
}
*/
shell_vector = _mm256_min_epi32(shell_vector, max_shell_vector);
```

Detalle de la vectorización



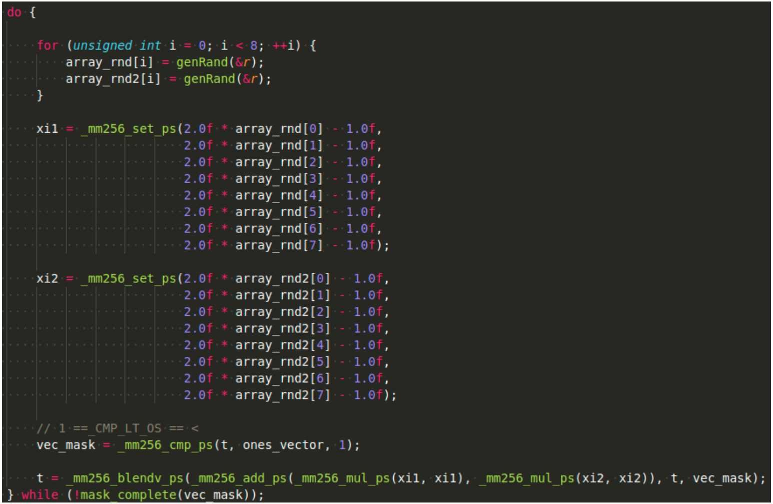
```
/*
heat[shell] += (1.0f - albedo) * weight;
heat2[shell] += (1.0f - albedo) * (1.0f - albedo) * weight * weight;
weight *= albedo;
*/
__m256 helper_vector = _mm256_mul_ps(_mm256_sub_ps(ones_vector, albedo), weight);
__m256 helper_vector_squared = _mm256_mul_ps(helper_vector, helper_vector); /* add up squares */

index_array[0] = _mm256_extract_epi32(shell_vector, 0);
index_array[1] = _mm256_extract_epi32(shell_vector, 1);
index_array[2] = _mm256_extract_epi32(shell_vector, 2);
index_array[3] = _mm256_extract_epi32(shell_vector, 3);
index_array[4] = _mm256_extract_epi32(shell_vector, 4);
index_array[5] = _mm256_extract_epi32(shell_vector, 5);
index_array[6] = _mm256_extract_epi32(shell_vector, 6);
index_array[7] = _mm256_extract_epi32(shell_vector, 7);


for (unsigned int i = 0; i < 8; ++i) {
    heat[index_array[i]] += (float)helper_vector[i];
    heat2[index_array[i]] += (float)helper_vector_squared[i];
}
```

Detalle de la vectorización



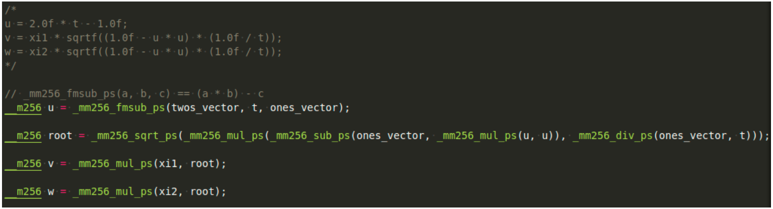
```
do {
    
    xi1 = 2.0f * genRand(&r) - 1.0f;
    xi2 = 2.0f * genRand(&r) - 1.0f;
    t = xi1 * xi1 + xi2 * xi2;
    
} while (1.0f < t);
```

Detalle de la vectorización



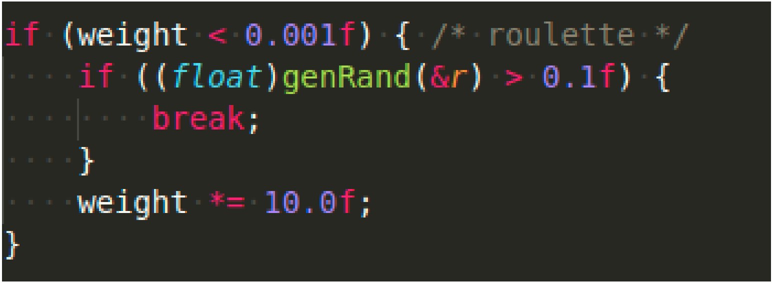
```c
do {

    for (unsigned int i = 0; i < 8; ++i) {
        array_rnd[i] = genRand(&r);
        array_rnd2[i] = genRand(&r);
    }

    xi1 = _mm256_set_ps(2.0f * array_rnd[0] - 1.0f,
                        2.0f * array_rnd[1] - 1.0f,
                        2.0f * array_rnd[2] - 1.0f,
                        2.0f * array_rnd[3] - 1.0f,
                        2.0f * array_rnd[4] - 1.0f,
                        2.0f * array_rnd[5] - 1.0f,
                        2.0f * array_rnd[6] - 1.0f,
                        2.0f * array_rnd[7] - 1.0f);

    xi2 = _mm256_set_ps(2.0f * array_rnd2[0] - 1.0f,
                        2.0f * array_rnd2[1] - 1.0f,
                        2.0f * array_rnd2[2] - 1.0f,
                        2.0f * array_rnd2[3] - 1.0f,
                        2.0f * array_rnd2[4] - 1.0f,
                        2.0f * array_rnd2[5] - 1.0f,
                        2.0f * array_rnd2[6] - 1.0f,
                        2.0f * array_rnd2[7] - 1.0f);

    // 1 == _CMP_LT_OS == <
    vec_mask = _mm256_cmp_ps(t, ones_vector, 1);

    t = _mm256_blendv_ps(_mm256_add_ps(_mm256_mul_ps(xi1, xi1), _mm256_mul_ps(xi2, xi2)), t, vec_mask);
} while (!mask_complete(vec_mask));
```

Detalle de la vectorización



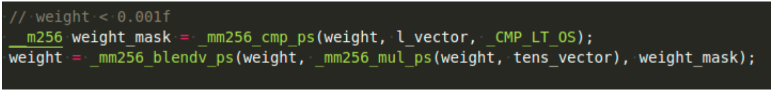
```
/*
u = 2.0f * t - 1.0f;
v = xi1 * sqrtf((1.0f - u * u) * (1.0f / t));
w = xi2 * sqrtf((1.0f - u * u) * (1.0f / t));
*/

// _mm256_fmsub_ps(a, b, c) == (a * b) - c
__m256 u = _mm256_fmsub_ps(twos_vector, t, ones_vector);

__m256 root = _mm256_sqrt_ps(_mm256_mul_ps(_mm256_sub_ps(ones_vector, _mm256_mul_ps(u, u)), _mm256_div_ps(ones_vector, t)));

__m256 v = _mm256_mul_ps(xi1, root);

__m256 w = _mm256_mul_ps(xi2, root);
```

Detalle de la vectorización



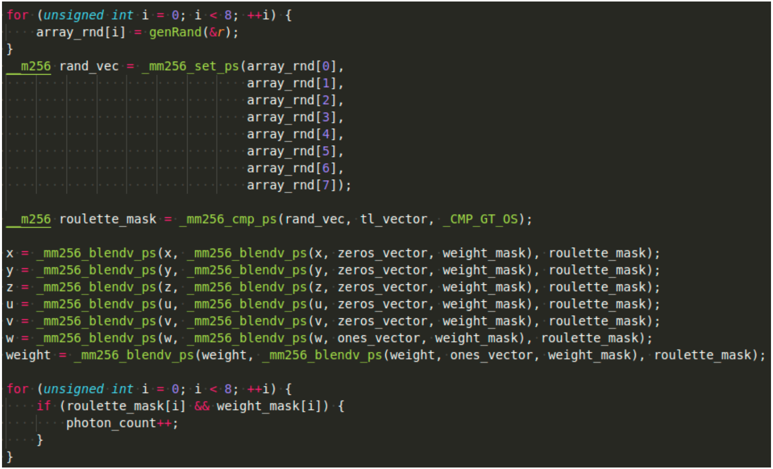
```c
if (weight < 0.001f) { /* roulette */
    if ((float)genRand(&r) > 0.1f) {
        break;
    }
    weight *= 10.0f;
}
```

Detalle de la vectorización



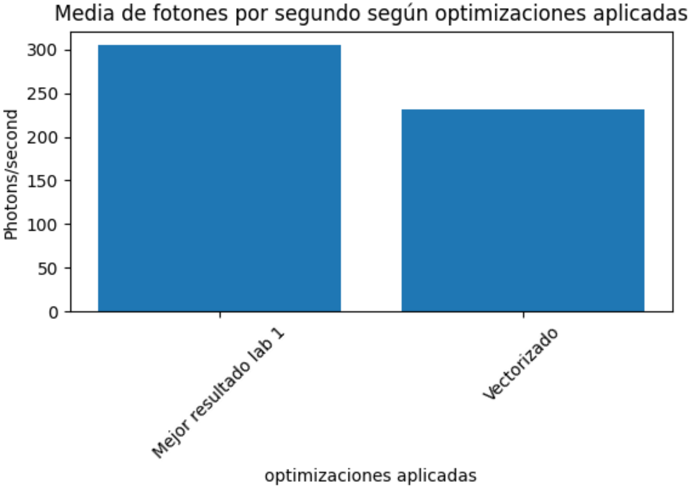
```
// weight < 0.001f
__m256 weight_mask = _mm256_cmp_ps(weight, l_vector, _CMP_LT_OS);
weight = _mm256_blendv_ps(weight, _mm256_mul_ps(weight, tens_vector), weight_mask);
```

Detalle de la vectorización

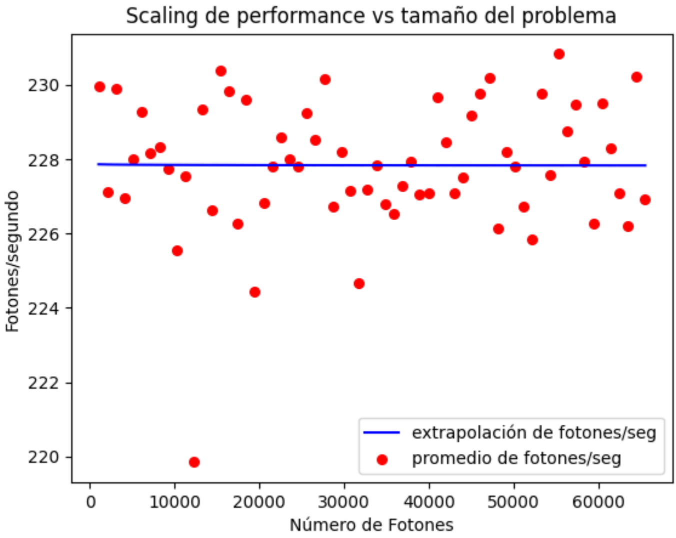


```
for (unsigned int i = 0; i < 8; ++i) {
    array_rnd[i] = genRand(&r);
}
__m256 rand_vec = _mm256_set_ps(array_rnd[0],
                                array_rnd[1],
                                array_rnd[2],
                                array_rnd[3],
                                array_rnd[4],
                                array_rnd[5],
                                array_rnd[6],
                                array_rnd[7]);

__m256 roulette_mask = _mm256_cmp_ps(rand_vec, tl_vector, _CMP_GT_OS);

x = _mm256_blendv_ps(x, _mm256_blendv_ps(x, zeros_vector, weight_mask), roulette_mask);
y = _mm256_blendv_ps(y, _mm256_blendv_ps(y, zeros_vector, weight_mask), roulette_mask);
z = _mm256_blendv_ps(z, _mm256_blendv_ps(z, zeros_vector, weight_mask), roulette_mask);
u = _mm256_blendv_ps(u, _mm256_blendv_ps(u, zeros_vector, weight_mask), roulette_mask);
v = _mm256_blendv_ps(v, _mm256_blendv_ps(v, zeros_vector, weight_mask), roulette_mask);
w = _mm256_blendv_ps(w, _mm256_blendv_ps(w, ones_vector, weight_mask), roulette_mask);
weight = _mm256_blendv_ps(weight, _mm256_blendv_ps(weight, ones_vector, weight_mask), roulette_mask);

for (unsigned int i = 0; i < 8; ++i) {
    if (roulette_mask[i] && weight_mask[i]) {
        photon_count++;
    }
}
```

Comparación entre vecotrización y mejor versión del lab 1



Scaling de vectorización



Mejoras para la vectorización

- Vectorizar el generador de números aleatorios para evitar ciclos lineales durante el programa