Lab 2 - Vectorización

Ignacio Scarinci

COMPUTACIÓN PARALELA – FaMAF - 2021

Introducción

Características de la PC

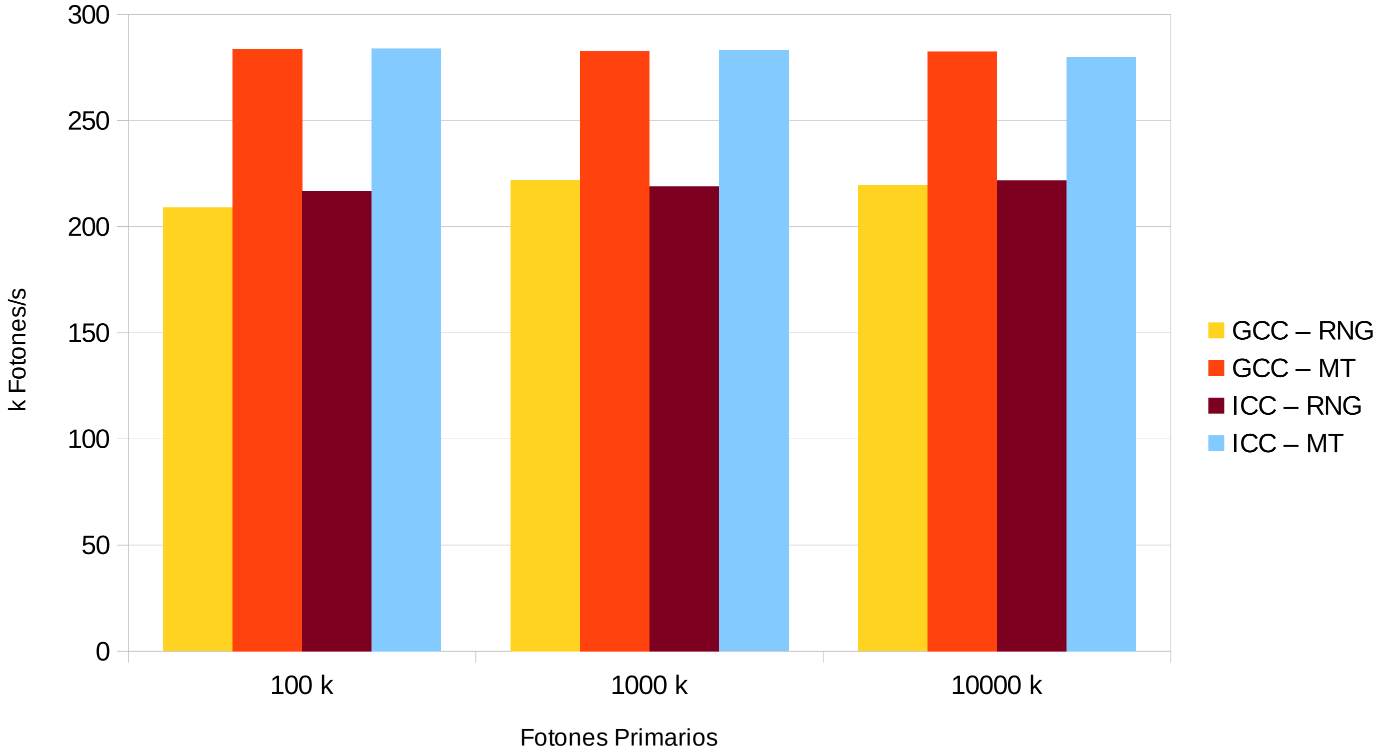
- Procesador: i7-6700HQ
- Memoria RAM: 16 GB (2x8GB - DDR4-2133)
- S.O. Ubuntu 20.04.2 LTS
- Kernel: 5.8.0-53-generic
- GCC 10.2.0
- ICC 2021.2.0

Cambio de generador

- En lab 1 se utilizo el generador por defecto
- En lab 2 se cambio por Merssene Twister* (Coded by Takuji Nishimura and Makoto Matsumoto)

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

Cambio de generador



Autovectorización

- GCC:
  - -O2 -march=native
  - -O2 -ftree-vectorize -fopt-info-vec

```
mt19937ar.c:121:9: optimized: loop vectorized using 32 byte vectors
mt19937ar.c:117:9: optimized: loop vectorized using 32 byte vectors
mt19937ar.c:117:9: optimized: loop vectorized using 16 byte vectors
```

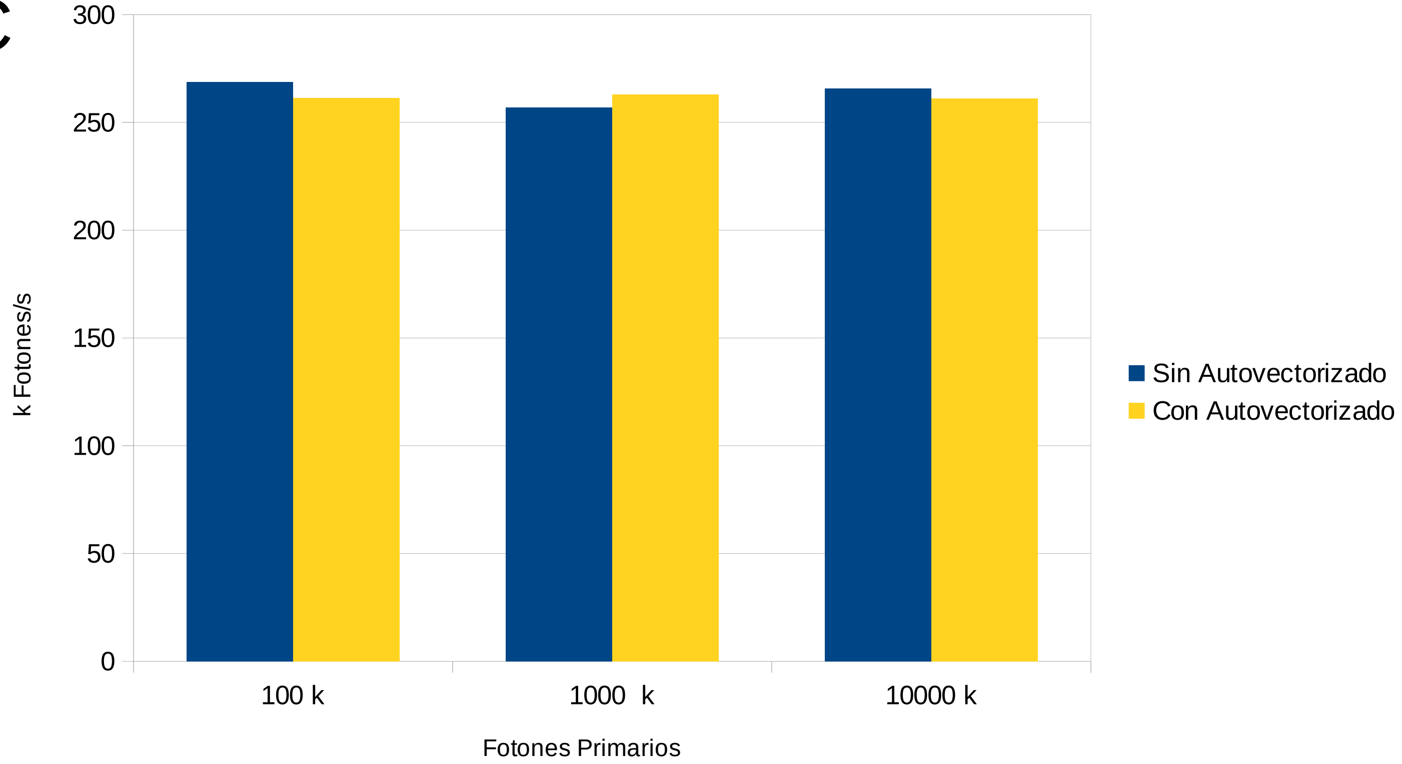
Autovectorización

- ICC:
  - -O2 march=native -qopt-report=1 -qopt-report-phase=vec

```
LOOP BEGIN at mt19937ar.c(117,9)
   remark #15388: vectorization support: reference mt[kk] has aligned access   [ mt19937ar.c(118,18) ]
   remark #15389: vectorization support: reference mt[kk+1] has unaligned access   [ mt19937ar.c(118,38) ]
   remark #15388: vectorization support: reference mt[kk] has aligned access   [ mt19937ar.c(119,13) ]
   remark #15389: vectorization support: reference mt[kk+397] has unaligned access   [ mt19937ar.c(119,22) ]
   remark #15381: vectorization support: unaligned access used inside loop body
   remark #15335: loop was not vectorized: vectorization possible but seems inefficient. Use vector always directive or -vec-threshold0 to override
   remark #15328: vectorization support: irregularly indexed load was emulated for the variable <mag01[y&1]>, 64-bit indexed, part of index is read from memory [ mt19937ar.c(119,44) ]
   remark #15305: vectorization support: vector length 2
   remark #15309: vectorization support: normalized vectorization overhead 0.062
   remark #15448: unmasked aligned unit stride loads: 1
   remark #15449: unmasked aligned unit stride stores: 1
   remark #15450: unmasked unaligned unit stride loads: 2
   remark #15462: unmasked indexed (or gather) loads: 1
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 13
   remark #15477: vector cost: 16.000
   remark #15478: estimated potential speedup: 0.810
   remark #15488: --- end vector cost summary ---
LOOP END
```

Autovectorización

Resultados GCC



Intrinsics

```
// configure RNG
init_genrand(SEED);
// start timer
double start = wtime();

// simulation
photon(PHOTONS);

for (uint i=0; i<SHELLS; ++i){
    heat2[i] = heat[i] * heat[i];
}
```

Intrinsics

```c
__m256 albedo = _mm256_set1_ps(MU_S / (MU_S + MU_A));
__m256 albedo_1 = _mm256_set1_ps(1.0f - (MU_S / (MU_S + MU_A)));
__m256 shells_per_mfp = _mm256_set1_ps (1e4 / MICRONS_PER_SHELL / (MU_A + MU_S));
__m256 negativo = _mm256_set1_ps(-1.0f);
__m256 shell_ = _mm256_set1_ps(SHELLS -1.0f);
__m256 umbral_w = _mm256_set1_ps(0.001f);
__m256 umbral_a = _mm256_set1_ps(0.01f);
__m256 factor_ruleta = _mm256_set1_ps(10.0f);
__m256 zero =  _mm256_set1_ps(0.0f);
int fotones = 0;
```

Intrinsics

```c
while (fotones < n_primarios){
fotones += 8;
/* launch */
__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);
__m256 t = _mm256_set1_ps(0.0f);

float xi1, xi2, tt;
```

Intrinsics

```
for (;;) {
  __m256 aleatorio = _mm256_set_ps(genrand_real1(),
                          genrand_real1(),
                          genrand_real1(),
                          genrand_real1(),
                          genrand_real1(),
                          genrand_real1(),
                          genrand_real1(),
                          genrand_real1());
  t = _mm256_mul_ps(negativo, _mm256_log_ps(aleatorio)); /* move */
  x = _mm256_add_ps(x, _mm256_mul_ps(t, u));
  y = _mm256_add_ps(y, _mm256_mul_ps(t, v));
  z = _mm256_add_ps(z, _mm256_mul_ps(t,  w));

  __m256 shell = _mm256_sqrt_ps(_mm256_add_ps(
                               _mm256_add_ps(_mm256_mul_ps(x,x),  _mm256_mul_ps(y,y)),
                               _mm256_mul_ps(z,z)));

  shell = _mm256_mul_ps(shell, shells_per_mfp);

  shell =_mm256_blendv_ps(shell, shell_, _mm256_cmp_ps (shell, shell_, 14 ));

  __m256 calor = _mm256_mul_ps(albedo_1, weight);

  weight = _mm256_mul_ps(weight, albedo);
```

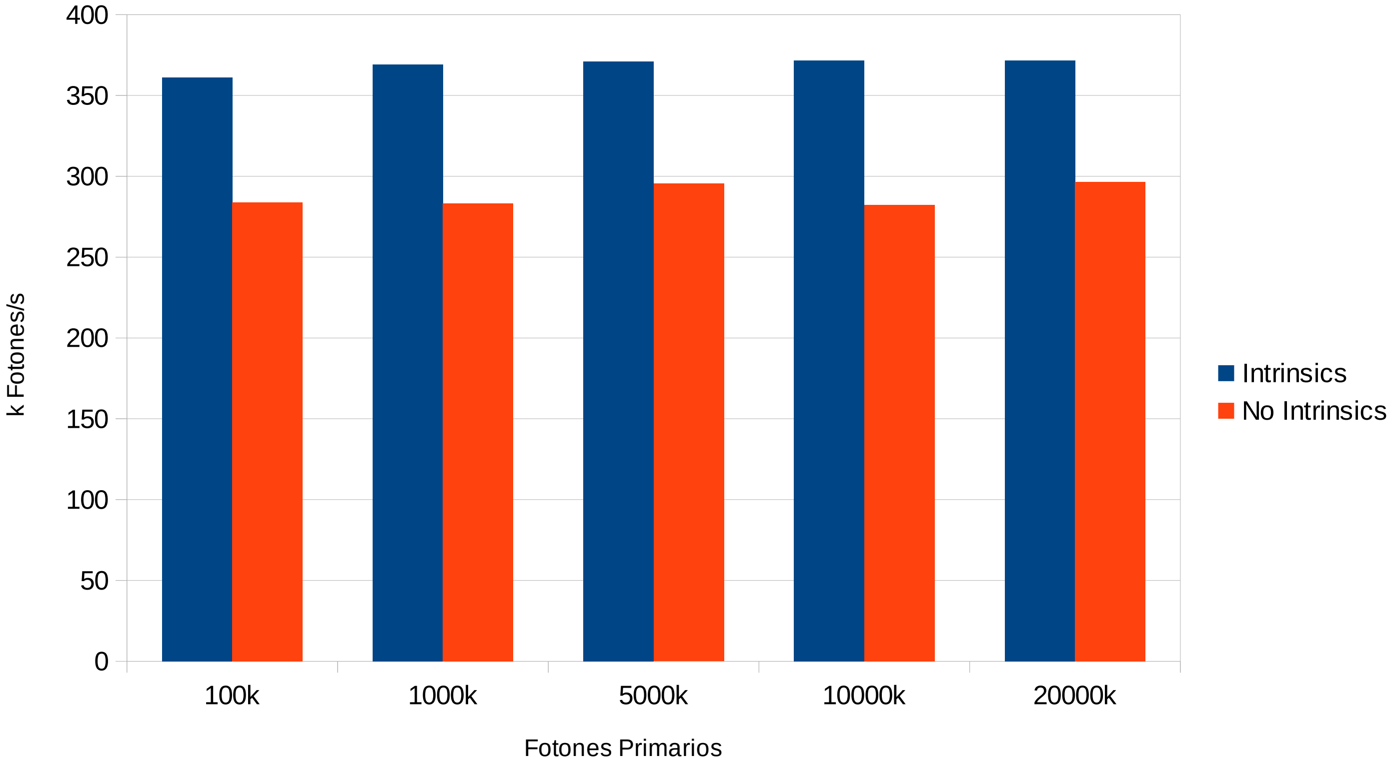
Intrinsics

```
__m256 u_w = _mm256_cmp_ps (  weight, umbral_w,1 );
int menor = _mm256_movemask_ps(u_w);

if (menor > 0){
  weight = _mm256_blendv_ps(weight, _mm256_mul_ps(weight, factor_ruleta), u_w);
 __m256 aleatorio = _mm256_set_ps(genrand_real1(),
                        genrand_real1(),
                        genrand_real1(),
                        genrand_real1(),
                        genrand_real1(),
                        genrand_real1(),
                        genrand_real1(),
                        genrand_real1());
 __m256 u_a = _mm256_cmp_ps( umbral_a,  aleatorio, 14);
 weight = _mm256_blendv_ps(weight, zero, u_a);
 if (_mm256_movemask_ps(weight) == 0){
   break;
 }
}
```

Intrinsics

```c
/* New direction, rejection method */
for (uint i=0; i<8; i=i+1){
do {
    xi1 = 2.0f * genrand_real1() - 1.0f;
    xi2 = 2.0f * genrand_real1() - 1.0f;
    tt = xi1 * xi1 + xi2 * xi2;
} while (1.0f < tt);

u[i] = 2.0f * tt - 1.0f;
v[i] = xi1 * sqrtf((1.0f - u[i] * u[i]) / tt);
w[i] = xi2 * sqrtf((1.0f - u[i] * u[i]) / tt);
```

Resultados Intrinsics



Conclusiones

- El autovectorizado puede “vectorizar” pero eso no quiere decir que realmente funcione
- Es relativamente fácil pasar un código ya escrito a intrinsics sin embargo hacerlo de forma eficiente es complejo y suele ser anti-intuitivo
- ISPC no logre que funcionara