

# Laboratorio 3

Alvaro Frias Garay - Ary Lautaro Di Bartolo

Universidad Nacional de Córdoba - Universidad Nacional de Cuyo

2021

# Estrategias e Implementaciones

- ▶ Paralelizar tinymc no vectorizado
- ▶ Paralelizar tinymc vectorizado

# Paralelizando el tinymc no vectorizado

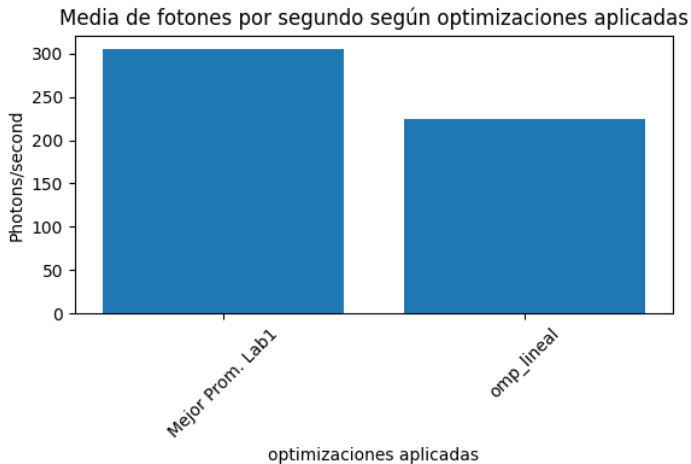
```
#pragma omp parallel for  
for (unsigned int i = 0; i < PHOTONS; ++i) {  
    ... photon(r);  
}
```

# Paralelizando el tinymc no vectorizado

```
#pragma omp parallel for
for (unsigned int i = 0; i < PHOTONS; ++i) {
    ... photon(r);
}
```

```
#pragma omp critical
heat[shell] += (1.0f - albedo) * weight;
#pragma omp critical
heat2[shell] += (1.0f - albedo) * (1.0f - albedo) * weight * weight;
```

# Paralelizando el tinymc no vectorizado



# Paralelizando el tinymc no vectorizado

## Principales Problemas

- ▶ False sharing
- ▶ scheduling no dinámico

# Paralelizando el tinymc no vectorizado

```
#pragma omp parallel for schedule(dynamic) private(heat_pr, heat2_pr, x, y, z, u, v, w, t, xi1, xi2, shell, weight)
for (unsigned int i = 0; i < PHOTONS; ++i) {
```

# Paralelizando el tinymc no vectorizado

```
heat_pr[shell] += (1.0f - albedo) * weight;  
heat2_pr[shell] += (1.0f - albedo) * (1.0f - albedo) * weight * weight;
```

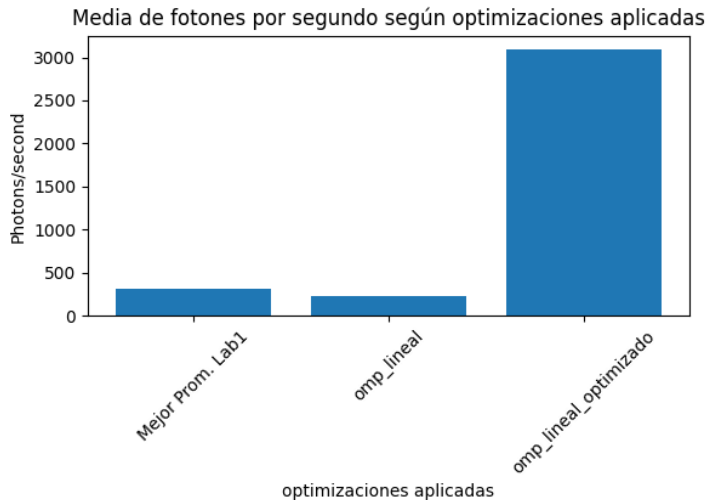


# Paralelizando el tinymc no vectorizado

```
heat_pr[shell] += (1.0f - albedo) * weight;
heat2_pr[shell] += (1.0f - albedo) * (1.0f - albedo) * weight * weight;

if ((float)genRand(&r) > 0.1f) {
    // cargando a los shared heats a partir de los privados
    #pragma omp critical
    {
        for (int n = 0; n < SHELLS; ++n) {
            heat[n] += heat_pr[n];
            heat2[n] += heat2_pr[n];
        }
    }
    break;
}
```

# Paralelizando el tinymc no vectorizado



# Paralelizando tinymc vectorizado

- ▶ Muy difícil paralelizar intrinsics
- ▶ Refactorizar el código

# Paralelizando tinymc vectorizado

```
#pragma omp simd [clause[[,] clause]....] new-line  
    for-loops
```

OpenMP incluye directivas para vectorizar for loops

# Detalle de vectorización con OpenMP

```
float heat_pr[SHELLS];  
float heat2_pr[SHELLS];  
float x[8] = {0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f};  
float y[8] = {0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f};  
float z[8] = {0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f};  
float u[8] = {0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f};  
float v[8] = {0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f};  
float w[8] = {  
    1.0f,  
    1.0f,  
    1.0f,  
    1.0f,  
    1.0f,  
    1.0f,  
    1.0f,  
    1.0f,  
};  
float weight[8] = {  
    1.0f,  
    1.0f,  
    1.0f,  
    1.0f,  
    1.0f,  
    1.0f,  
    1.0f,  
    1.0f,  
};
```

# Detalle de vectorización con OpenMP

```
#pragma omp simd aligned(albedo, shells_per_mfp:32)
for (unsigned int i = 0; i < 8; ++i) {
    albedo[i] = MU_S * (1.0f / (MU_S + MU_A));
    shells_per_mfp[i] = 1e4 * (1.0f / MICRONS_PER_SHELL) * (1.0f / (MU_A + MU_S));
}
```

# Detalle de vectorización con OpenMP

```
while (photon_count < PHOTONS) {  
    launch:*/  
    ... #pragma omp simd aligned(t, x, y, z, shell, weight:32)  
    ... for (unsigned int i = 0; i < 8; ++i) {  
    ...     t[i] = -logf((float)genRand(&r)); /*move*/  
    ...     x[i] += t[i] * u[i];  
    ...     y[i] += t[i] * v[i];  
    ...     z[i] += t[i] * w[i];  
    ...     shell[i] = (unsigned int)sqrtf(x[i]*x[i] + y[i]*y[i] + z[i]*z[i]) * shells_per_mfp[i];  
    ...     if (shell[i] > SHELLS - 1) {  
    ...         shell[i] = SHELLS - 1;  
    ...     }  
    ...     heat_pr[shell[i]] += (1.0f - albedo[i]) * weight[i];  
    ...     heat2_pr[shell[i]] += (1.0f - albedo[i]) * (1.0f - albedo[i]) * weight[i] * weight[i]; /* ad  
    ...     weight[i] *= albedo[i];  
    ... }
```

## Detalle de vectorización con OpenMP

```
do {  
    #pragma omp simd aligned(x1, x2:32)  
    for (unsigned int i = 0; i < 8; ++i) {  
        x1[i] = 2.0f * genRand(&r) - 1.0f;  
        x2[i] = 2.0f * genRand(&r) - 1.0f;  
  
        if (t_mask[i] == 0) {  
            t[i] = x1[i] * x1[i] + x2[i] * x2[i];  
        }  
    }  
  
    #pragma omp simd aligned(t:32)  
    for (unsigned int i = 0; i < 8; ++i) {  
        if (t[i] <= 1.0f) {  
            t_mask[i] = 1;  
        }  
    }  
} while (check_t_greater_1(t));
```



# Detalle de vectorización con OpenMP

```
#pragma omp simd aligned(u, v, w, t, xi1, xi2, weight, x, y, z:32)
for (unsigned int i = 0; i < 8; ++i) {
    u[i] = 2.0f * t[i] - 1.0f;
    v[i] = xi1[i] * sqrtf((1.0f - u[i] * u[i]) * (1.0f / t[i]));
    w[i] = xi2[i] * sqrtf((1.0f - u[i] * u[i]) * (1.0f / t[i]));

    if (weight[i] < 0.001f) { /* roulette */
        weight[i] *= 10.0f;

        if ((float)genRand(&r) > 0.1f) {
            // reset lane
            x[i] = 0.0f;
            y[i] = 0.0f;
            z[i] = 0.0f;
            u[i] = 0.0f;
            v[i] = 0.0f;
            w[i] = 1.0f;
            weight[i] = 1.0f;
            #pragma omp atomic
            photon_count++;
            #pragma omp flush
        }
    }
}
```

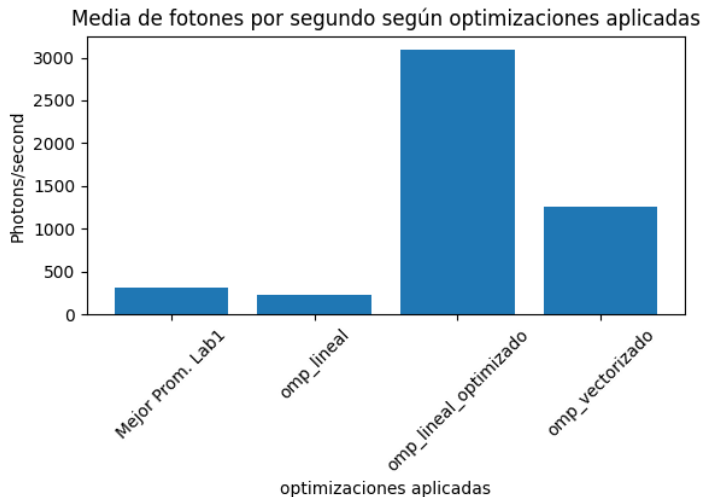
## Detalle de vectorización con OpenMP

```
#pragma omp simd
for (unsigned int i = 0; i < SHELLS; ++i) {
    . . . #pragma omp atomic
    . . . | . . . heat[i] += heat_pr[i];
    . . . #pragma omp atomic
    . . . | . . . heat2[i] += heat2_pr[i];
    . . .
    . . .
}
```

# Detalle de vectorización con OpenMP

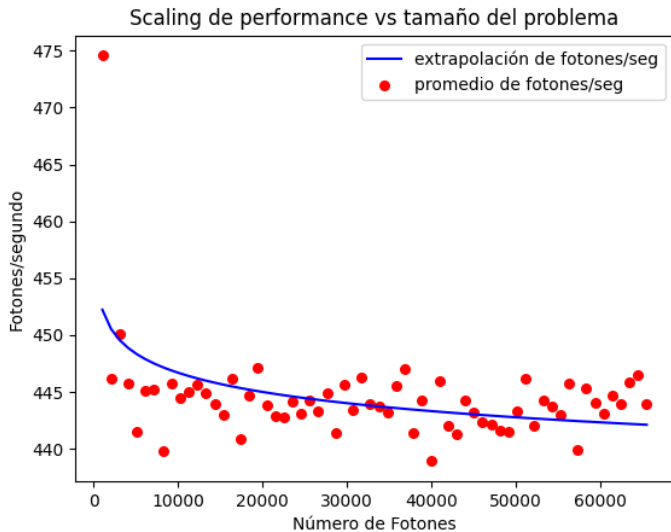
```
#pragma omp parallel shared(photon_count, heat, heat2)
{
    photon(r);
}
```

# Comparación de velocidades



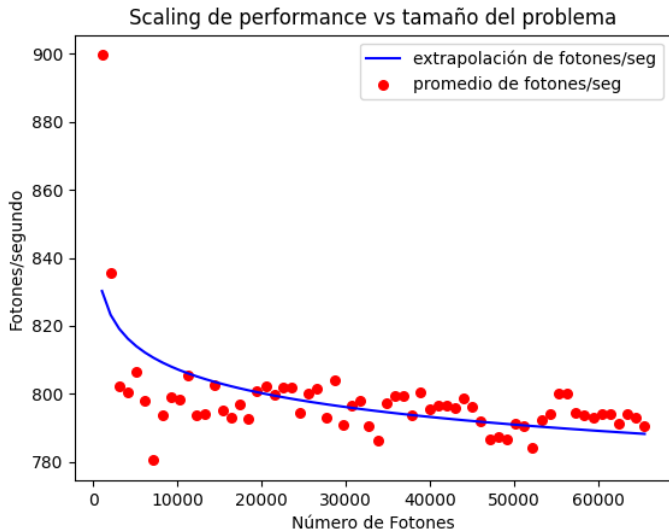
# Scaling según número de threads

## 2 Threads



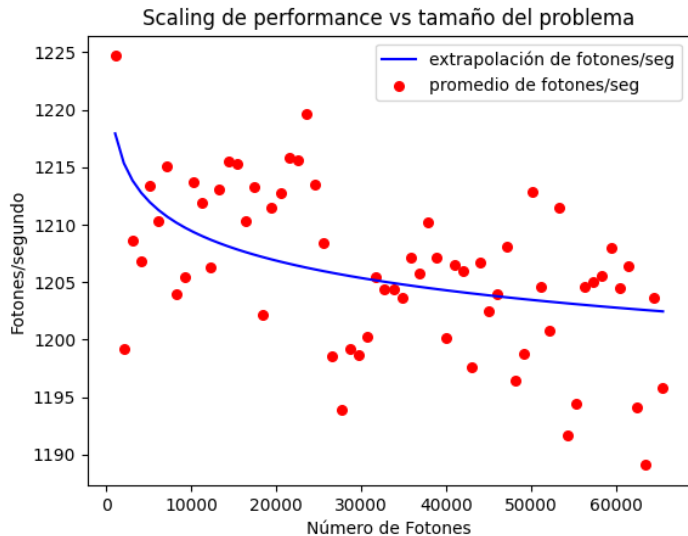
# Scaling según número de threads

## 4 Threads

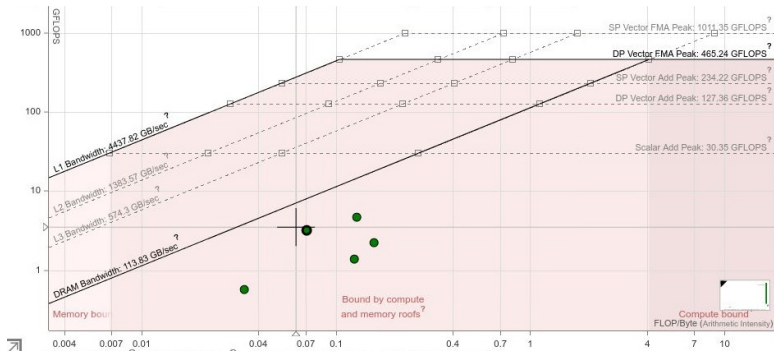


# Scaling según número de threads

8 Threads



# Roofline





# Conclusiones

- ▶ Código mucho más legible
- ▶ Mejoras notables en velocidad respecto a versiones anteriores

# Potenciales Mejoras

- ▶ Seguimos sin vectorizar el generador de números