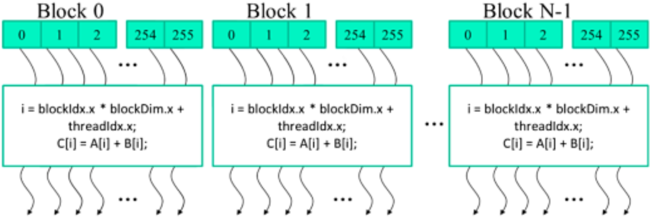
# Laboratorio 2

Alvaro Frias Garay - Ary Lautaro Di Bartolo

Universidad Nacional de Córdoba - Universidad Nacional de Cuyo

2021

# Threads y blocks



```
i = blockIdx.x * blockDim.x +
threadIdx.x;
C[i] = A[i] + B[i];
```

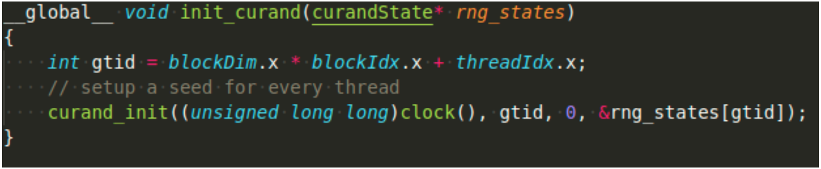
```
i = blockIdx.x * blockDim.x +
threadIdx.x;
C[i] = A[i] + B[i];
```

```
i = blockIdx.x * blockDim.x +
threadIdx.x;
C[i] = A[i] + B[i];
```

# Estrategias e implementación

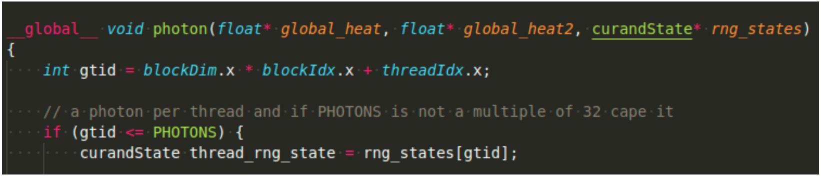
- Un seed por thread
- Un fotón por thread
- heats en unified memory

# Estrategias e implementación



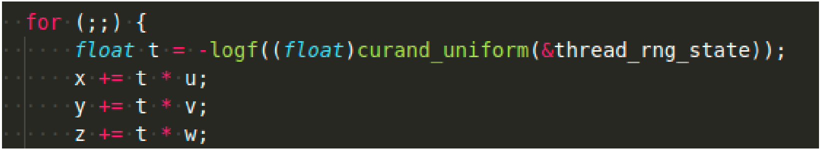
```
__global__ void init_curand(curandState* rng_states)
{
    int gtid = blockDim.x * blockIdx.x + threadIdx.x;
    // setup a seed for every thread
    curand_init((unsigned long long)clock(), gtid, 0, &rng_states[gtid]);
}
```

# Estrategias e implementación



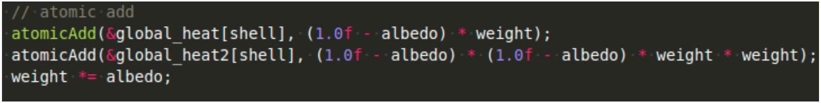
```
__global__ void photon(float* global_heat, float* global_heat2, curandState* rng_states)
{
    int gtid = blockDim.x * blockIdx.x + threadIdx.x;

    // a photon per thread and if PHOTONS is not a multiple of 32 cape it
    if (gtid <= PHOTONS) {
        curandState thread_rng_state = rng_states[gtid];
```

# Estrategias e implementación



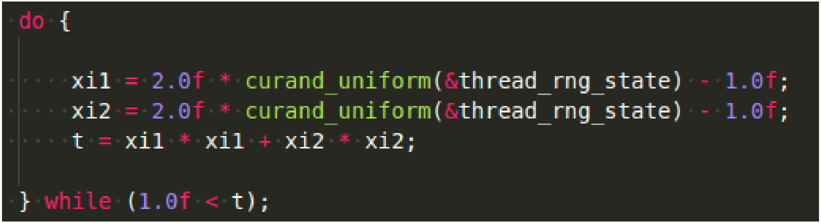
```
  for (;;) {
      float t = -logf((float)curand_uniform(&thread_rng_state));
      x += t * u;
      y += t * v;
      z += t * w;
```

# Estrategias e implementación



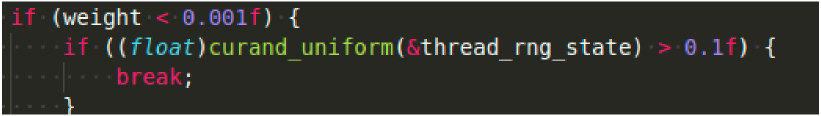
```
// atomic add
atomicAdd(&global_heat[shell], (1.0f - albedo) * weight);
atomicAdd(&global_heat2[shell], (1.0f - albedo) * (1.0f - albedo) * weight * weight);
weight *= albedo;
```

# Estrategias e implementación



```
do {

    xi1 = 2.0f * curand_uniform(&thread_rng_state) - 1.0f;
    xi2 = 2.0f * curand_uniform(&thread_rng_state) - 1.0f;
    t = xi1 * xi1 + xi2 * xi2;

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

# Estrategias e implementación

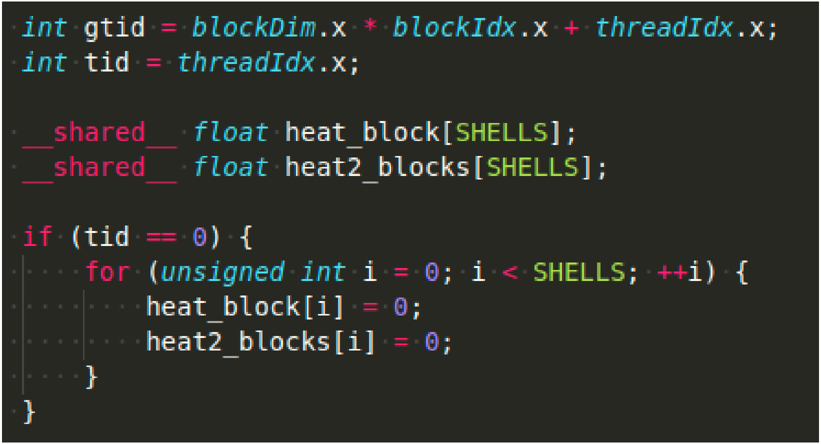


```
if (weight < 0.001f) {
    if ((float)curand_uniform(&thread_rng_state) > 0.1f) {
        break;
    }
```

# Optimización al kernel

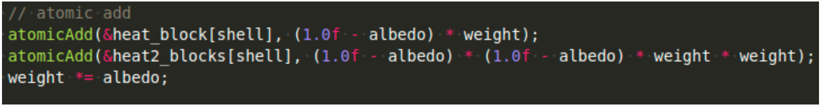
- Arrays heat de memoria __shared__

# Optimización al kernel



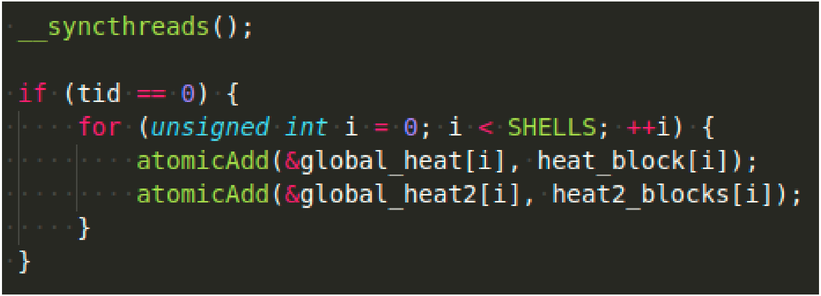
```
int gtid = blockDim.x * blockIdx.x + threadIdx.x;
int tid = threadIdx.x;

__shared__ float heat_block[SHELLS];
__shared__ float heat2_blocks[SHELLS];

if (tid == 0) {
    for (unsigned int i = 0; i < SHELLS; ++i) {
        heat_block[i] = 0;
        heat2_blocks[i] = 0;
    }
}
```

# Optimización al kernel



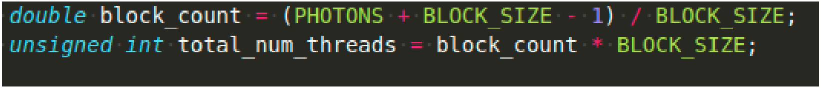
```
// atomic add
atomicAdd(&heat_block[shell], (1.0f - albedo) * weight);
atomicAdd(&heat2_blocks[shell], (1.0f - albedo) * (1.0f - albedo) * weight * weight);
weight *= albedo;
```

# Optimización al kernel



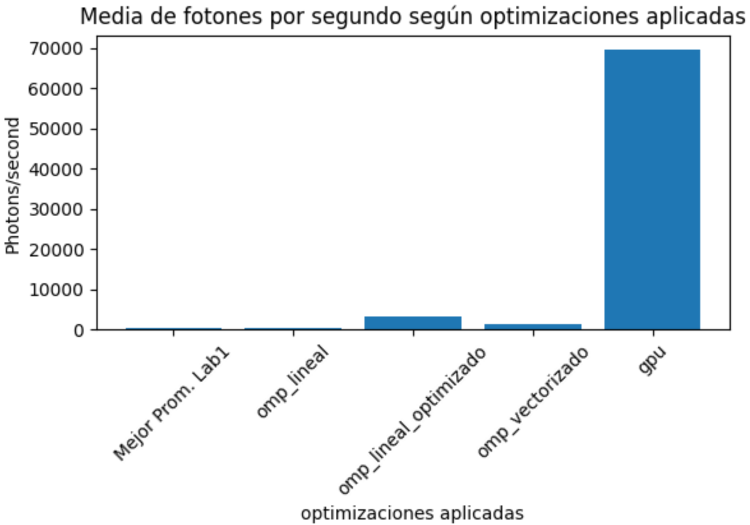
```
__syncthreads();

if (tid == 0) {
    for (unsigned int i = 0; i < SHELLS; ++i) {
        atomicAdd(&global_heat[i], heat_block[i]);
        atomicAdd(&global_heat2[i], heat2_blocks[i]);
    }
}
```

# Optimización al kernel

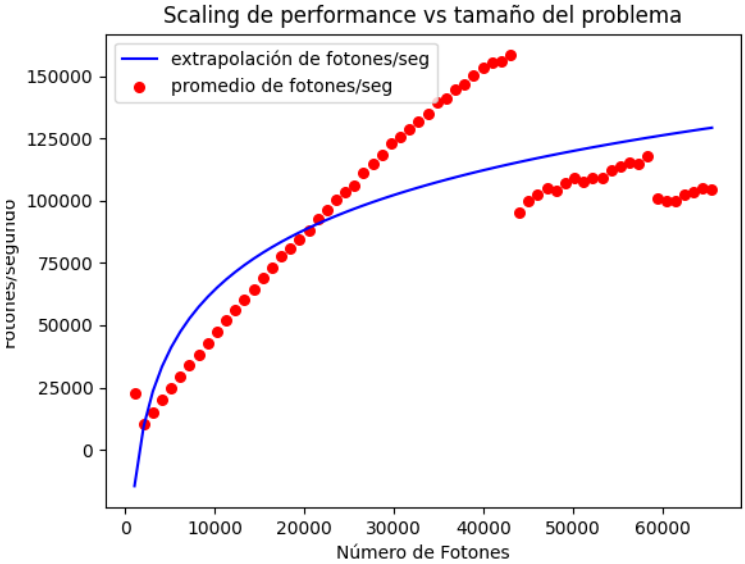


```
 double block_count = (PHOTONS + BLOCK_SIZE -  1) / BLOCK_SIZE;
 unsigned int total_num_threads = block_count * BLOCK_SIZE;
```

# Comparación con el resto de los labs



# Scaling



Scaling de performance vs tamaño del problema
extrapolación de fotones/seg
promedio de fotones/seg
Fotones/segundo
150000
125000
100000
75000
50000
25000
0
0
10000
20000
30000
40000
50000
60000
Número de Fotones

# Posibles mejores

- Uso de arrays a nivel de warp
- Escalar el problema