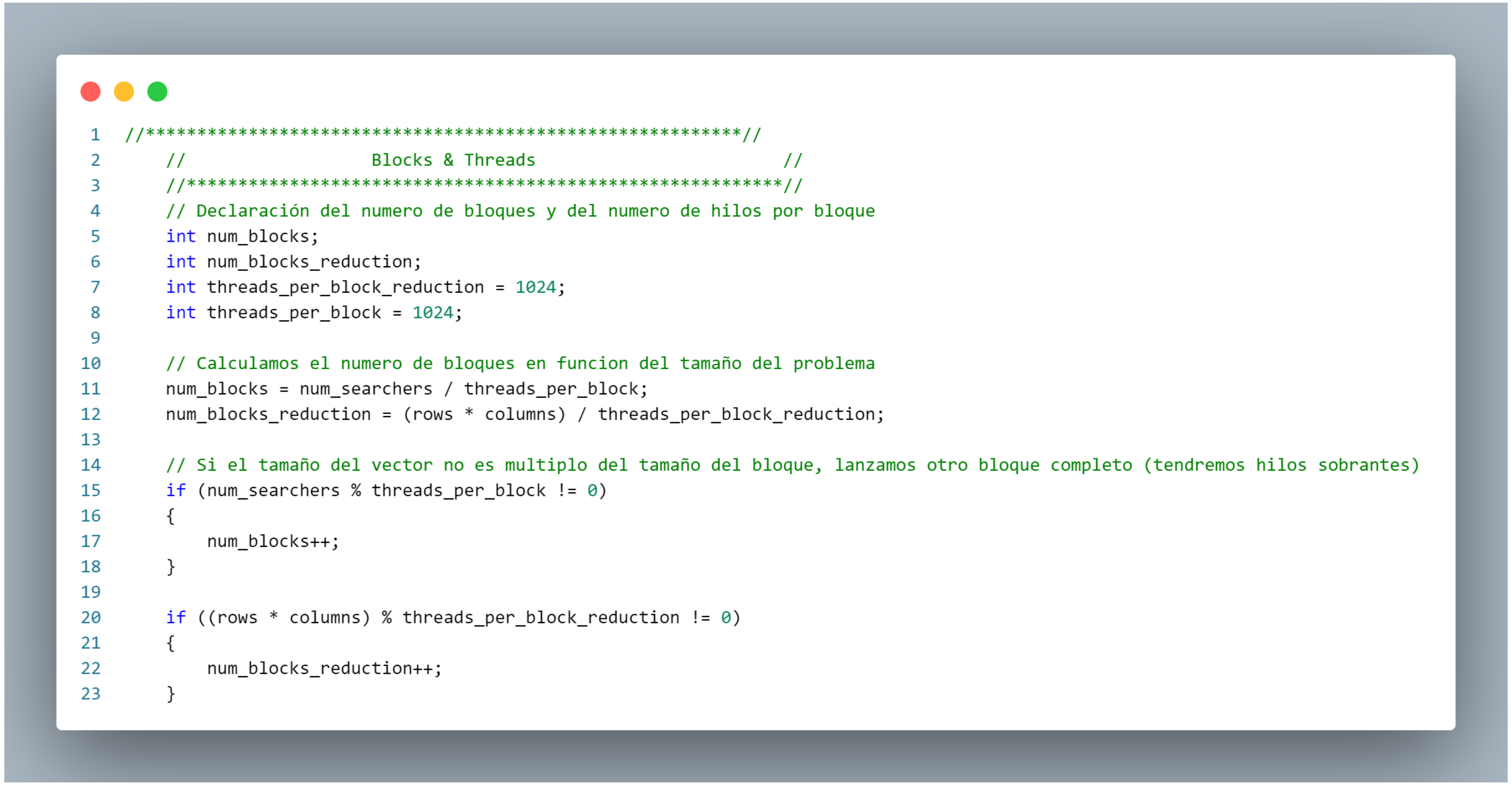
Computación Paralela: CUDA

JHON STEEVEN CABANILLA ALVARADO

MIGUEL CHAVEINTE GARCÍA

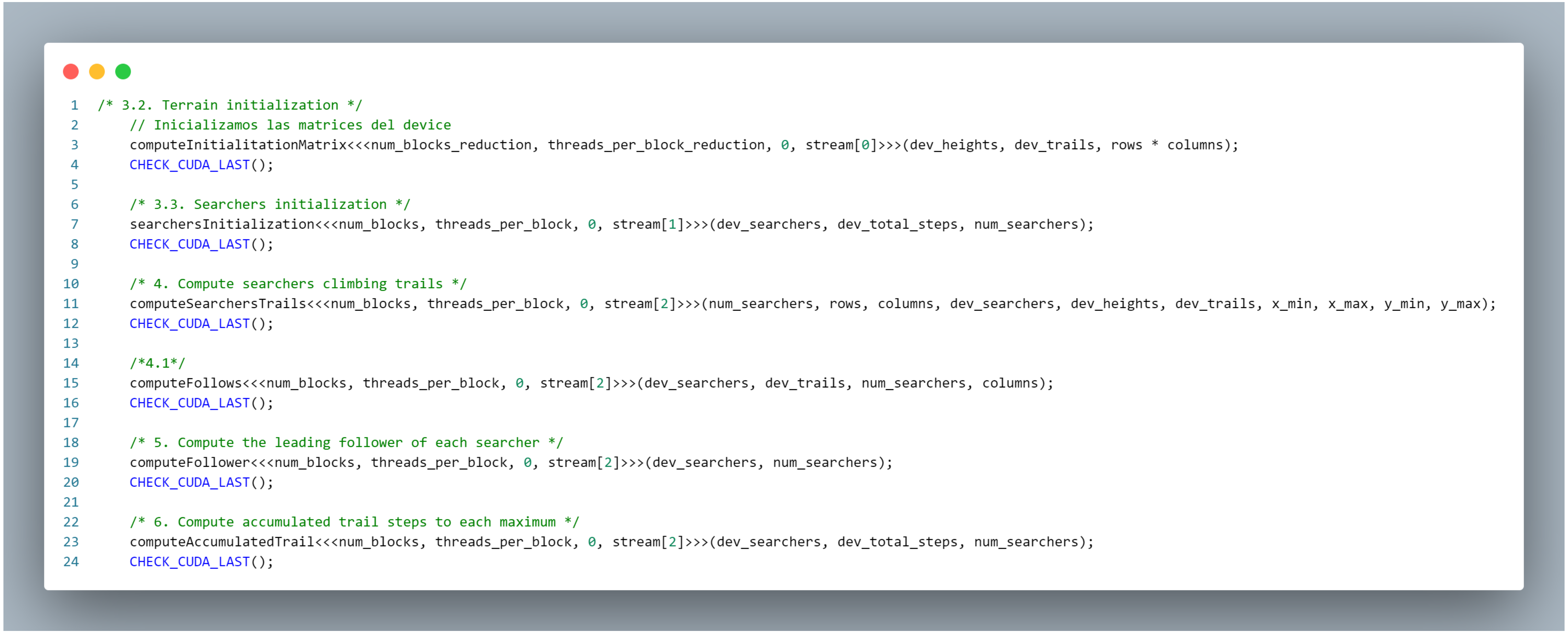
G01

Declaración de bloques e hilos



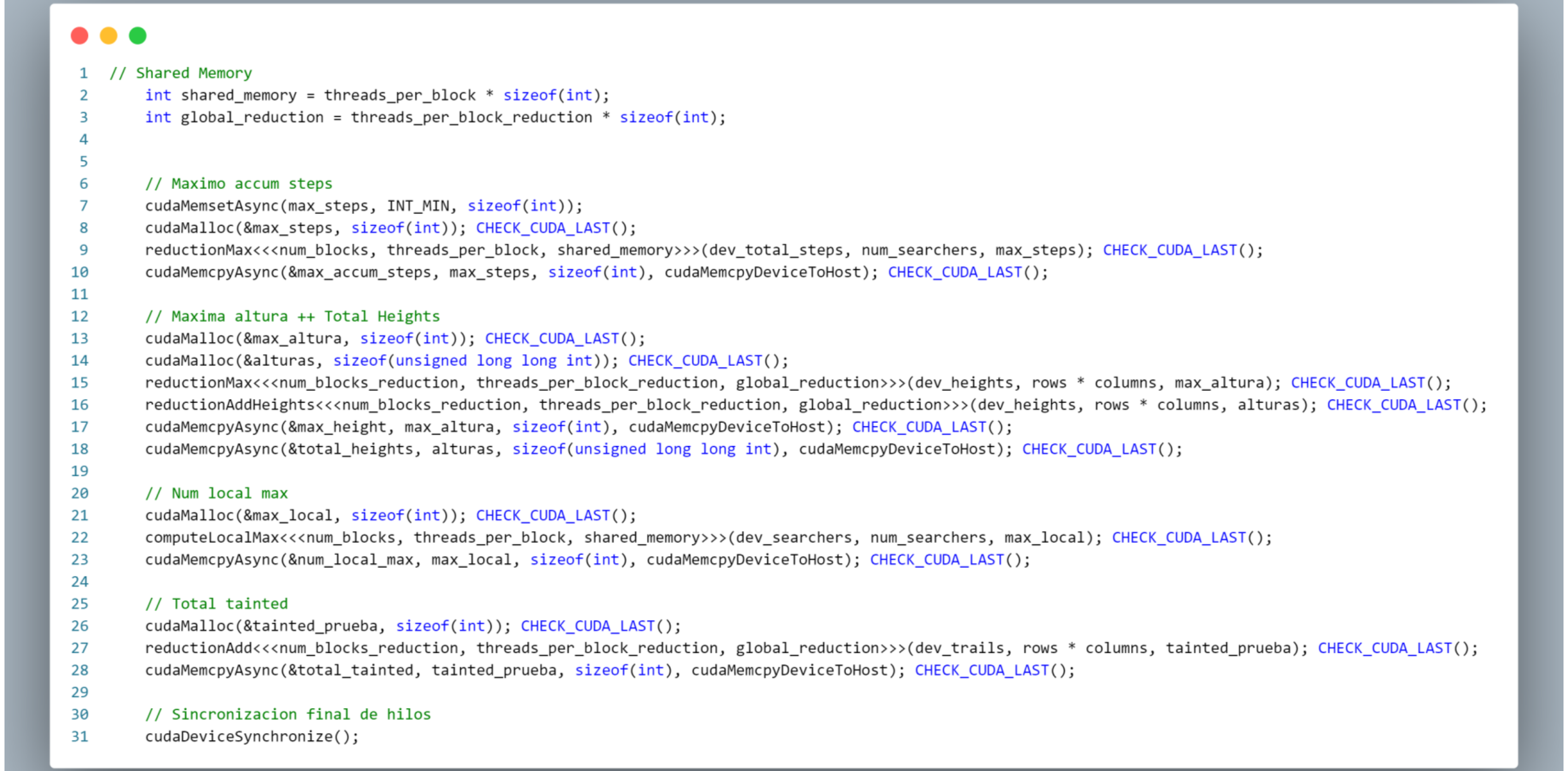
```
//**********************************************************//
    //                  Blocks & Threads                        //
    //**********************************************************//
    // Declaración del numero de bloques y del numero de hilos por bloque
    int num_blocks;
    int num_blocks_reduction;
    int threads_per_block_reduction = 1024;
    int threads_per_block = 1024;

    // Calculamos el numero de bloques en funcion del tamaño del problema
    num_blocks = num_searchers / threads_per_block;
    num_blocks_reduction = (rows * columns) / threads_per_block_reduction;

    // Si el tamaño del vector no es multiplo del tamaño del bloque, lanzamos otro bloque completo (tendremos hilos sobrantes)
    if (num_searchers % threads_per_block != 0)
    {
        num_blocks++;
    }

    if ((rows * columns) % threads_per_block_reduction != 0)
    {
        num_blocks_reduction++;
    }
```

Lanzamientos de Kernels



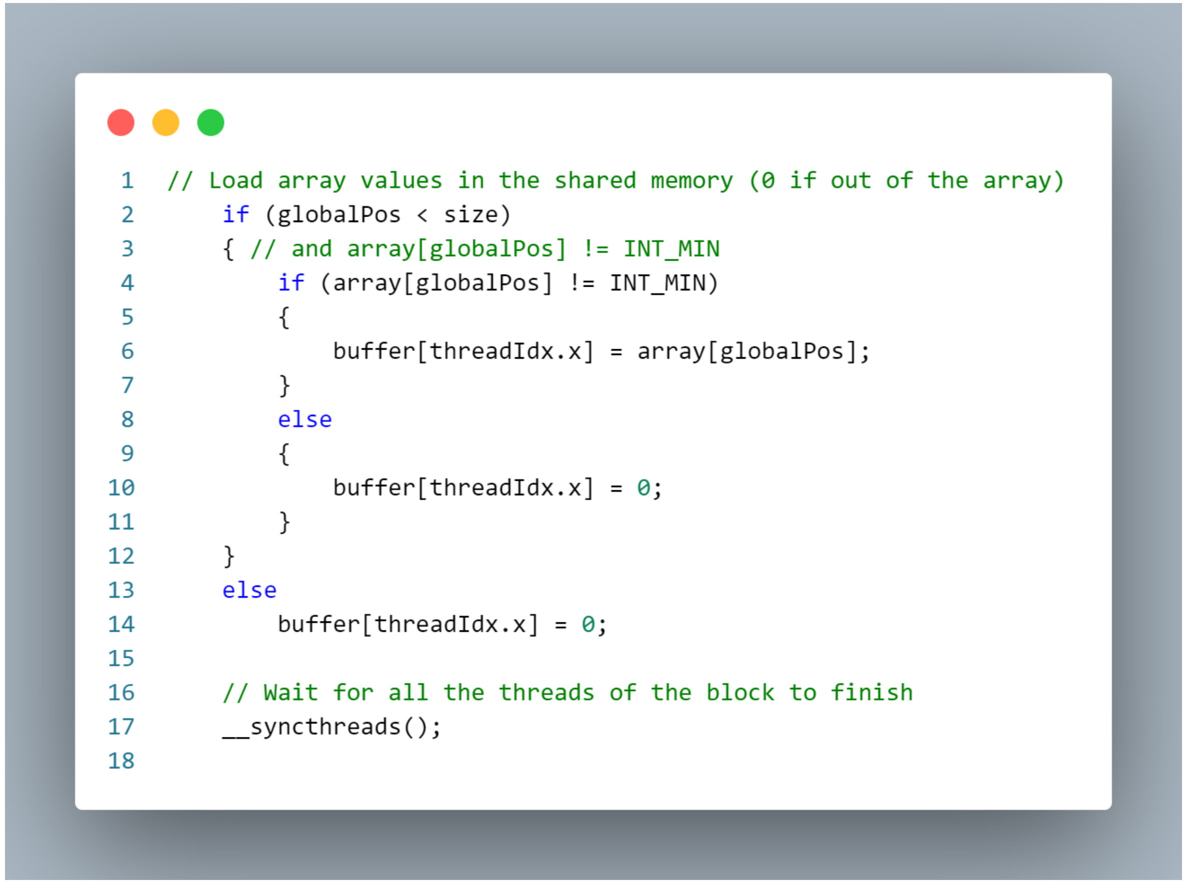
```
/* 3.2. Terrain initialization */
    // Inicializamos las matrices del device
    computeInitialitationMatrix<<<num_blocks_reduction, threads_per_block_reduction, 0, stream[0]>>>(dev_heights, dev_trails, rows * columns);
    CHECK_CUDA_LAST();

    /* 3.3. Searchers initialization */
    searchersInitialization<<<num_blocks, threads_per_block, 0, stream[1]>>>(dev_searchers, dev_total_steps, num_searchers);
    CHECK_CUDA_LAST();

    /* 4. Compute searchers climbing trails */
    computeSearchersTrails<<<num_blocks, threads_per_block, 0, stream[2]>>>(num_searchers, rows, columns, dev_searchers, dev_heights, dev_trails, x_min, x_max, y_min, y_max);
    CHECK_CUDA_LAST();

    /*4.1*/
    computeFollows<<<num_blocks, threads_per_block, 0, stream[2]>>>(dev_searchers, dev_trails, num_searchers, columns);
    CHECK_CUDA_LAST();

    /* 5. Compute the leading follower of each searcher */
    computeFollower<<<num_blocks, threads_per_block, 0, stream[2]>>>(dev_searchers, num_searchers);
    CHECK_CUDA_LAST();

    /* 6. Compute accumulated trail steps to each maximum */
    computeAccumulatedTrail<<<num_blocks, threads_per_block, 0, stream[2]>>>(dev_searchers, dev_total_steps, num_searchers);
    CHECK_CUDA_LAST();
```

Cómputo estadístico

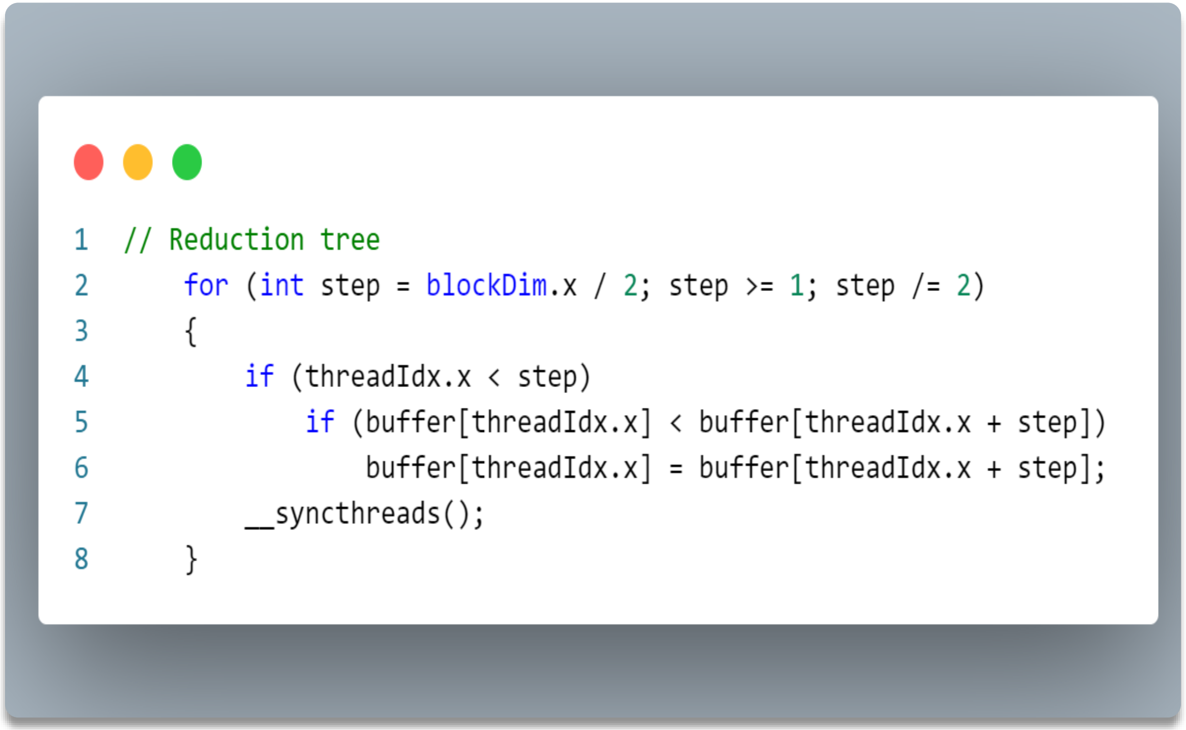


```
// Shared Memory
    int shared_memory = threads_per_block * sizeof(int);
    int global_reduction = threads_per_block_reduction * sizeof(int);


    // Maximo accum steps
    cudaMemsetAsync(max_steps, INT_MIN, sizeof(int));
    cudaMalloc(&max_steps, sizeof(int)); CHECK_CUDA_LAST();
    reductionMax<<<num_blocks, threads_per_block, shared_memory>>>(dev_total_steps, num_searchers, max_steps); CHECK_CUDA_LAST();
    cudaMemcpyAsync(&max_accum_steps, max_steps, sizeof(int), cudaMemcpyDeviceToHost); CHECK_CUDA_LAST();

    // Maxima altura ++ Total Heights
    cudaMalloc(&max_altura, sizeof(int)); CHECK_CUDA_LAST();
    cudaMalloc(&alturas, sizeof(unsigned long long int)); CHECK_CUDA_LAST();
    reductionMax<<<num_blocks_reduction, threads_per_block_reduction, global_reduction>>>(dev_heights, rows * columns, max_altura); CHECK_CUDA_LAST();
    reductionAddHeights<<<num_blocks_reduction, threads_per_block_reduction, global_reduction>>>(dev_heights, rows * columns, alturas); CHECK_CUDA_LAST();
    cudaMemcpyAsync(&max_height, max_altura, sizeof(int), cudaMemcpyDeviceToHost); CHECK_CUDA_LAST();
    cudaMemcpyAsync(&total_heights, alturas, sizeof(unsigned long long int), cudaMemcpyDeviceToHost); CHECK_CUDA_LAST();

    // Num local max
    cudaMalloc(&max_local, sizeof(int)); CHECK_CUDA_LAST();
    computeLocalMax<<<num_blocks, threads_per_block, shared_memory>>>(dev_searchers, num_searchers, max_local); CHECK_CUDA_LAST();
    cudaMemcpyAsync(&num_local_max, max_local, sizeof(int), cudaMemcpyDeviceToHost); CHECK_CUDA_LAST();

    // Total tainted
    cudaMalloc(&tainted_prueba, sizeof(int)); CHECK_CUDA_LAST();
    reductionAdd<<<num_blocks_reduction, threads_per_block_reduction, global_reduction>>>(dev_trails, rows * columns, tainted_prueba); CHECK_CUDA_LAST();
    cudaMemcpyAsync(&total_tainted, tainted_prueba, sizeof(int), cudaMemcpyDeviceToHost); CHECK_CUDA_LAST();

    // Sincronizacion final de hilos
    cudaDeviceSynchronize();
```

Reducciones

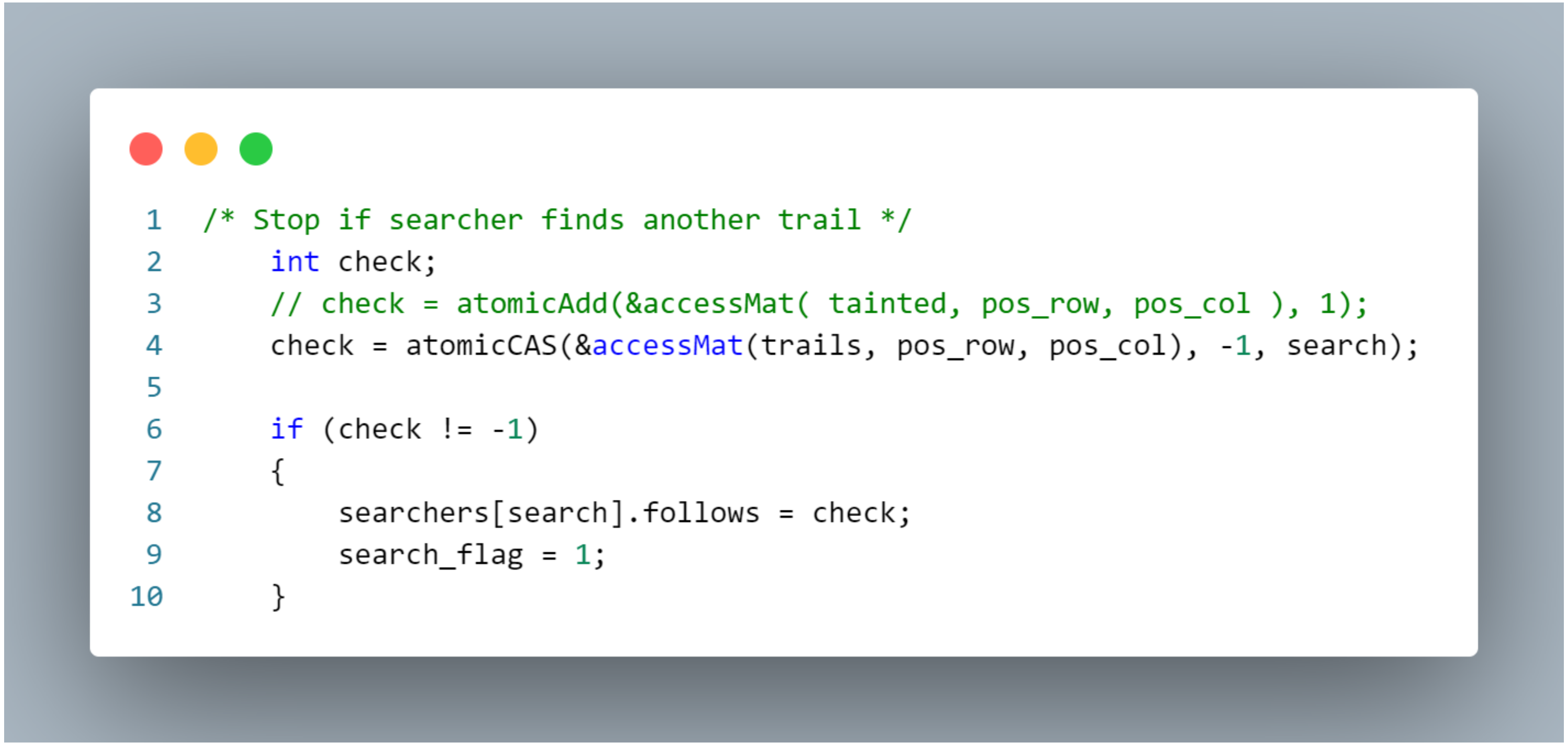


```
// Load array values in the shared memory (0 if out of the array)
    if (globalPos < size)
    { // and array[globalPos] != INT_MIN
        if (array[globalPos] != INT_MIN)
        {
            buffer[threadIdx.x] = array[globalPos];
        }
        else
        {
            buffer[threadIdx.x] = 0;
        }
    }
    else
        buffer[threadIdx.x] = 0;

    // Wait for all the threads of the block to finish
    __syncthreads();

```



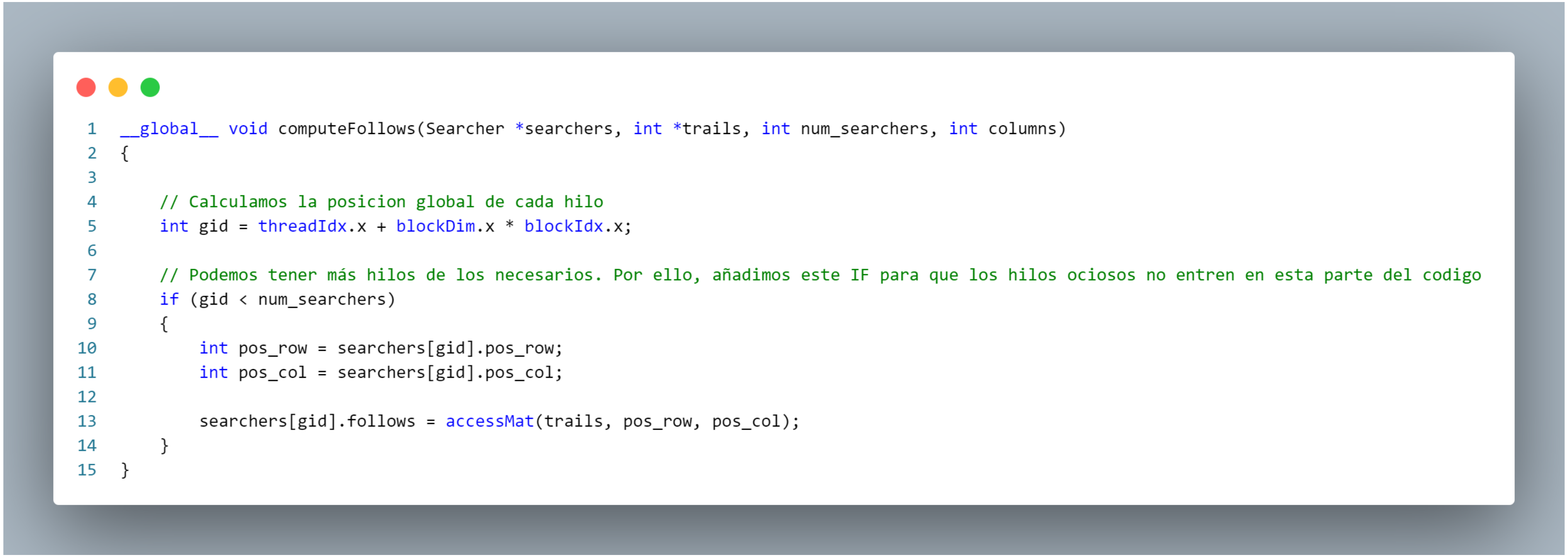
```
// Reduction tree
    for (int step = blockDim.x / 2; step >= 1; step /= 2)
    {
        if (threadIdx.x < step)
            if (buffer[threadIdx.x] < buffer[threadIdx.x + step])
                buffer[threadIdx.x] = buffer[threadIdx.x + step];
        __syncthreads();
    }
```

Condiciones de carrera



```
/* Stop if searcher finds another trail */
    int check;
    // check = atomicAdd(&accessMat( tainted, pos_row, pos_col ), 1);
    check = atomicCAS(&accessMat(trails, pos_row, pos_col), -1, search);

    if (check != -1)
    {
        searchers[search].follows = check;
        search_flag = 1;
    }
```

Ejemplo de kernel

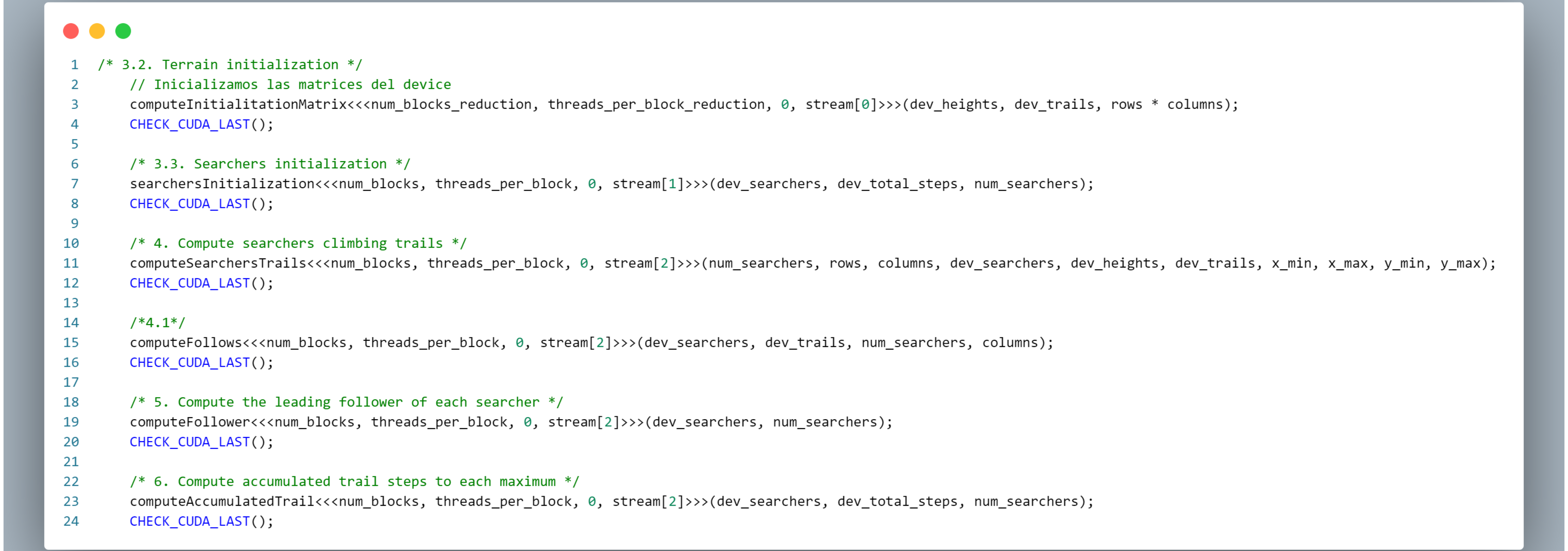


```
__global__ void computeFollows(Searcher *searchers, int *trails, int num_searchers, int columns)
{

    // Calculamos la posicion global de cada hilo
    int gid = threadIdx.x + blockDim.x * blockIdx.x;

    // Podemos tener más hilos de los necesarios. Por ello, añadimos este IF para que los hilos ociosos no entren en esta parte del codigo
    if (gid < num_searchers)
    {
        int pos_row = searchers[gid].pos_row;
        int pos_col = searchers[gid].pos_col;

        searchers[gid].follows = accessMat(trails, pos_row, pos_col);
    }
}
```

Optimizaciones

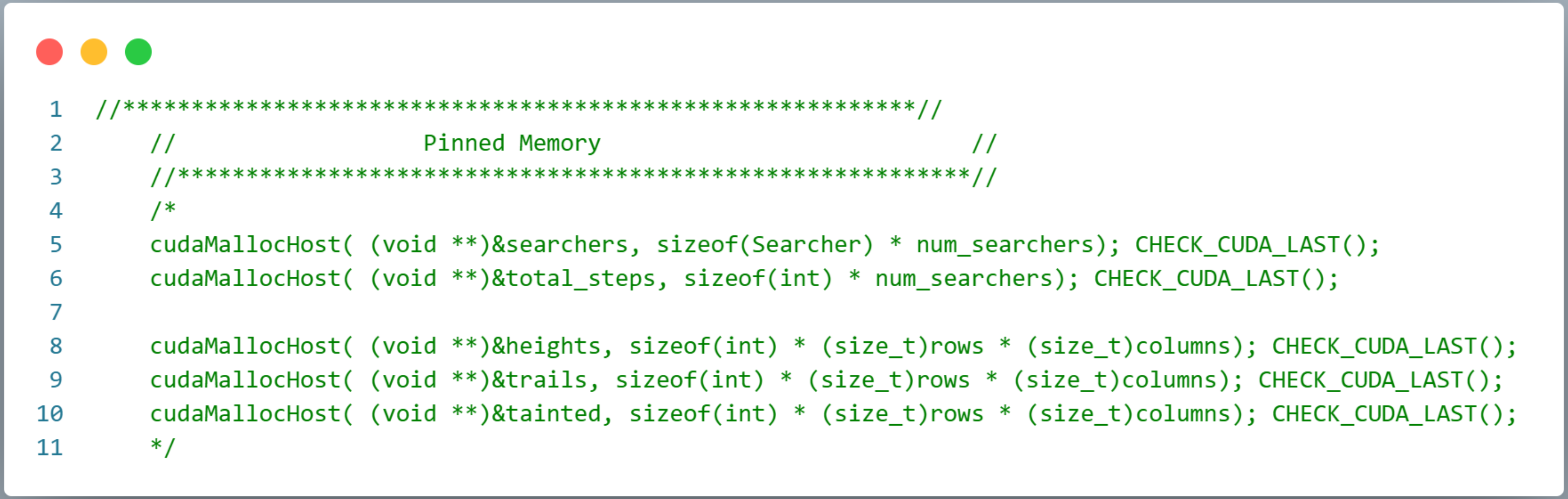
- Eliminar matriz tainted
- Eliminar el atributo id del struct Searchers
- Declaración de variables para ahorrar llamadas a funciones
- Uso de Streams

Streams

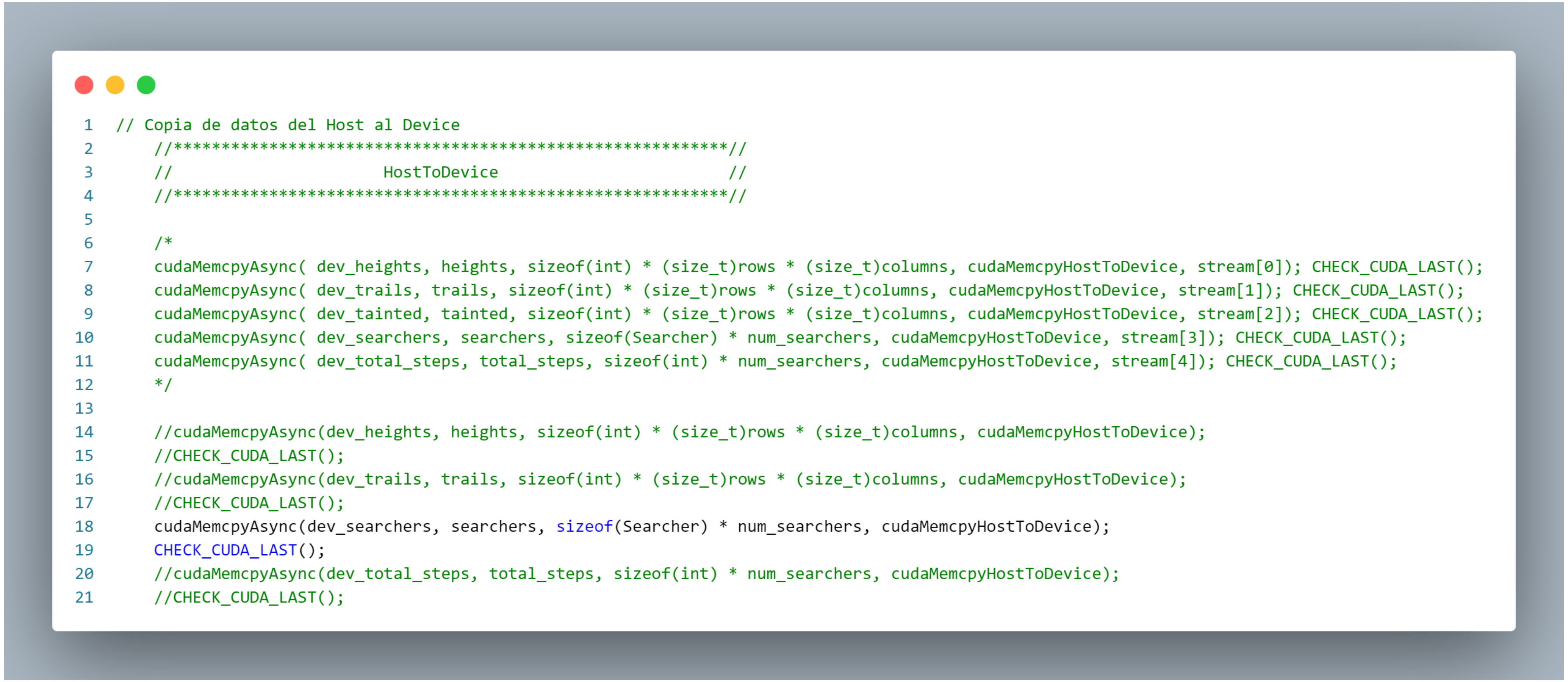


```
/* 3.2. Terrain initialization */
    // Inicializamos las matrices del device
    computeInitialitationMatrix<<<num_blocks_reduction, threads_per_block_reduction, 0, stream[0]>>>(dev_heights, dev_trails, rows * columns);
    CHECK_CUDA_LAST();

    /* 3.3. Searchers initialization */
    searchersInitialization<<<num_blocks, threads_per_block, 0, stream[1]>>>(dev_searchers, dev_total_steps, num_searchers);
    CHECK_CUDA_LAST();

    /* 4. Compute searchers climbing trails */
    computeSearchersTrails<<<num_blocks, threads_per_block, 0, stream[2]>>>(num_searchers, rows, columns, dev_searchers, dev_heights, dev_trails, x_min, x_max, y_min, y_max);
    CHECK_CUDA_LAST();

    /*4.1*/
    computeFollows<<<num_blocks, threads_per_block, 0, stream[2]>>>(dev_searchers, dev_trails, num_searchers, columns);
    CHECK_CUDA_LAST();

    /* 5. Compute the leading follower of each searcher */
    computeFollower<<<num_blocks, threads_per_block, 0, stream[2]>>>(dev_searchers, num_searchers);
    CHECK_CUDA_LAST();

    /* 6. Compute accumulated trail steps to each maximum */
    computeAccumulatedTrail<<<num_blocks, threads_per_block, 0, stream[2]>>>(dev_searchers, dev_total_steps, num_searchers);
    CHECK_CUDA_LAST();
```

Pinned Memory



```
//**********************************************************//
    //                  Pinned Memory                           //
    //**********************************************************//
    /*
    cudaMallocHost( (void **)&searchers, sizeof(Searcher) * num_searchers); CHECK_CUDA_LAST();
    cudaMallocHost( (void **)&total_steps, sizeof(int) * num_searchers); CHECK_CUDA_LAST();

    cudaMallocHost( (void **)&heights, sizeof(int) * (size_t)rows * (size_t)columns); CHECK_CUDA_LAST();
    cudaMallocHost( (void **)&trails, sizeof(int) * (size_t)rows * (size_t)columns); CHECK_CUDA_LAST();
    cudaMallocHost( (void **)&tainted, sizeof(int) * (size_t)rows * (size_t)columns); CHECK_CUDA_LAST();
    */
```



```
// Copia de datos del Host al Device
    //**********************************************************//
    //                      HostToDevice                        //
    //**********************************************************//

    /*
    cudaMemcpyAsync( dev_heights, heights, sizeof(int) * (size_t)rows * (size_t)columns, cudaMemcpyHostToDevice, stream[0]); CHECK_CUDA_LAST();
    cudaMemcpyAsync( dev_trails, trails, sizeof(int) * (size_t)rows * (size_t)columns, cudaMemcpyHostToDevice, stream[1]); CHECK_CUDA_LAST();
    cudaMemcpyAsync( dev_tainted, tainted, sizeof(int) * (size_t)rows * (size_t)columns, cudaMemcpyHostToDevice, stream[2]); CHECK_CUDA_LAST();
    cudaMemcpyAsync( dev_searchers, searchers, sizeof(Searcher) * num_searchers, cudaMemcpyHostToDevice, stream[3]); CHECK_CUDA_LAST();
    cudaMemcpyAsync( dev_total_steps, total_steps, sizeof(int) * num_searchers, cudaMemcpyHostToDevice, stream[4]); CHECK_CUDA_LAST();
    */

    //cudaMemcpyAsync(dev_heights, heights, sizeof(int) * (size_t)rows * (size_t)columns, cudaMemcpyHostToDevice);
    //CHECK_CUDA_LAST();
    //cudaMemcpyAsync(dev_trails, trails, sizeof(int) * (size_t)rows * (size_t)columns, cudaMemcpyHostToDevice);
    //CHECK_CUDA_LAST();
    cudaMemcpyAsync(dev_searchers, searchers, sizeof(Searcher) * num_searchers, cudaMemcpyHostToDevice);
    CHECK_CUDA_LAST();
    //cudaMemcpyAsync(dev_total_steps, total_steps, sizeof(int) * num_searchers, cudaMemcpyHostToDevice);
    //CHECK_CUDA_LAST();
```