

# Семинар 16

## Технология CUDA

### Атомарные операции

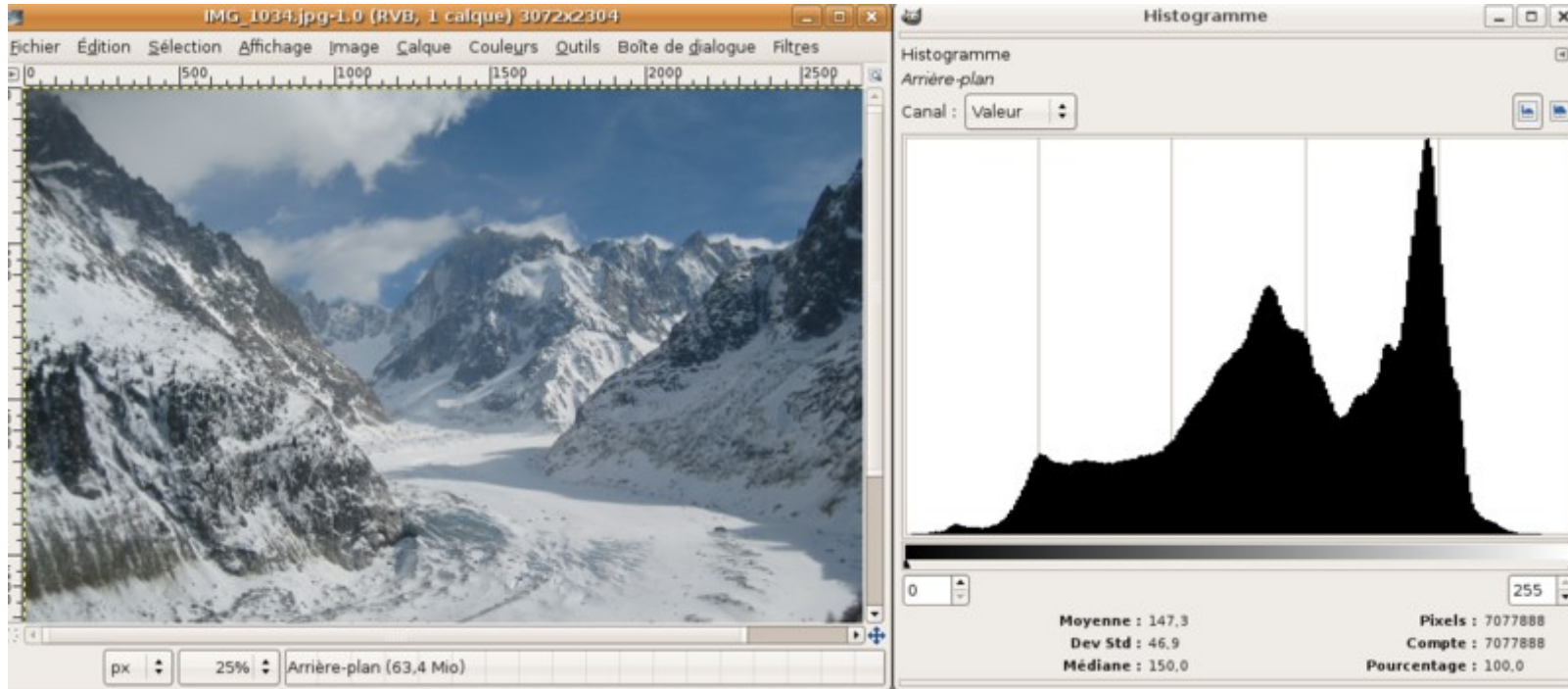
**Михаил Курносов**

E-mail: [mkurnosov@gmail.com](mailto:mkurnosov@gmail.com)

WWW: [www.mkurnosov.net](http://www.mkurnosov.net)

Цикл семинаров «Основы параллельного программирования»  
Институт физики полупроводников им. А. В. Ржанова СО РАН  
Новосибирск, 2015

# Гистограмма цветов изображения



- Задано изображение – двумерный массив целых чисел из интервала  $[0..255]$
- Необходимо построить гистограмму, таблицу  $hist[0..255]$ , элемент  $hist[i]$  которой равен числу пикселей с цветом  $i$

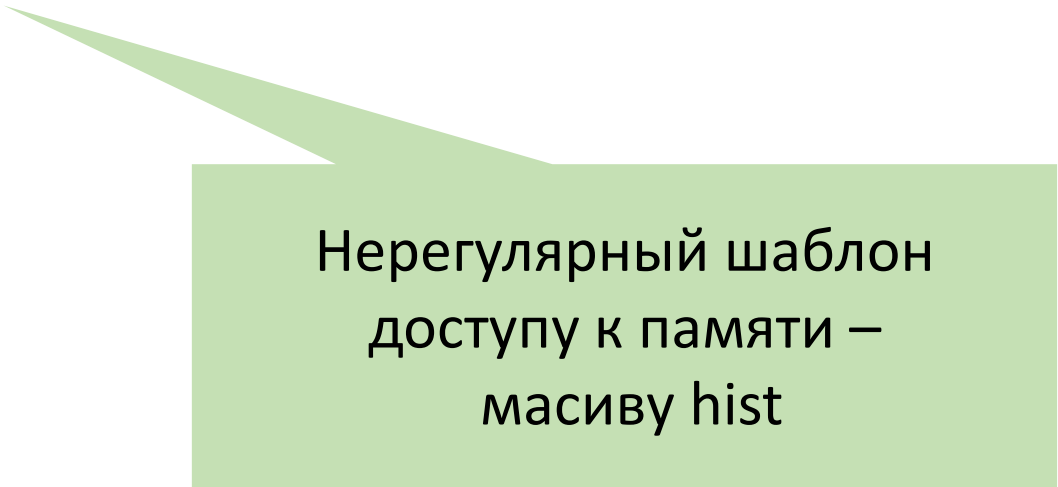
# Гистограмма цветов изображения (CPU)

```
// Выделение памяти под изображение
size_t size = sizeof(uint8_t) * width * height;
uint8_t *image = (uint8_t *)malloc(size);
if (image == NULL) {
    fprintf(stderr, "Allocation error.\n");
    exit(EXIT_FAILURE);
}
// Инициализация изображения
srand(0);
for (size_t i = 0; i < size; i++)
    image[i] = (rand() / (double)RAND_MAX) * 255;

// Инициализация гистограммы
int hist[256];
memset(hist, 0, sizeof(*hist) * 256);
```

# Гистограмма цветов изображения (CPU)

```
void hist_host(uint8_t *image, int width, int height,  
              int *hist)  
{  
    for (int i = 0; i < height; i++)  
        for (int j = 0; j < width; j++)  
            hist[image[i * width + j]]++;  
}
```



Нерегулярный шаблон  
доступу к памяти –  
массиву hist

# Гистограмма цветов изображения (GPU)

Каждый поток обрабатывает один пиксель изображения

```
// Гистограмма в памяти GPU
```

```
int *d_hist = NULL;
```

```
cudaMalloc((void **)&d_hist, sizeof(*d_hist) * 256);
```

```
cudaMemset(d_hist, 0, sizeof(*d_hist) * 256);
```

```
uint8_t *d_image = NULL;
```

```
cudaMalloc((void **)&d_image, size);
```

```
cudaMemcpy(d_image, image, size, cudaMemcpyHostToDevice);
```

```
int threadsPerBlock = 1024;
```

```
int blocksPerGrid = (size + threadsPerBlock - 1) / threadsPerBlock;
```

```
hist_gpu<<<blocksPerGrid, threadsPerBlock>>>(d_image, width,  
                                              height, d_hist);
```

```
cudaMemcpy(hist, d_hist, sizeof(*d_hist) * 256, cudaMemcpyDeviceToHost);
```

# Гистограмма цветов изображения (GPU)

Каждый поток обрабатывает один пиксель изображения

```
__global__ void hist_gpu(uint8_t *image, int width, int height,
                        int *hist)
{
    size_t size = width * height;
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < size) {
        hist[image[i]]++;
    }
}
```

# Гистограмма цветов изображения: тест

```
$ ./hist
```

```
Sum (CPU) = 104857600.000000
```

```
CUDA kernel launch with 102400 blocks of 1024 threads
```

```
Sum (GPU) = 3928061.000000
```

```
CPU version (sec.): 0.052969
```

```
GPU version (sec.): 0.068165
```

```
Memory ops. (sec.): 0.000019
```

```
Speedup: 0.78
```

```
Speedup (with mem ops.): 0.78
```

```
$ ./hist
```

```
Sum (CPU) = 104857600.000000
```

```
CUDA kernel launch with 102400 blocks of 1024 threads
```

```
Sum (GPU) = 3931478.000000
```

```
CPU version (sec.): 0.052965
```

```
GPU version (sec.): 0.068171
```

```
Memory ops. (sec.): 0.000020
```

```
Speedup: 0.78
```

```
Speedup (with mem ops.): 0.78
```

# Гистограмма цветов изображения (GPU)

Каждый поток обрабатывает один пиксель изображения

```
__global__ void hist_gpu(uint8_t *image, int width, int height,
                        int *hist)
{
    size_t size = width * height;
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < size) {
        hist[image[i]]++;
    }
}
```

Потоки одновременно читают и записывают данные  
в одни и те же ячейки таблицы hist[0..255] – data race



# CUDA Atomic Functions

- **Атомарная функция** (CUDA atomic function) – функция, выполняющая операцию *read-modify-write* (RMW) над 32 или 64 битным значением в глобальной или разделяемой памяти GPU
- `int atomicAdd(int* address, int val);`
- `unsigned long long int atomicAdd(unsigned long long int* address,`  
▪ `unsigned long long int val);`
- `float atomicAdd(float* address, float val);   // compute capability >= 2.0`
- `int atomicSub(int* address, int val);`
- `int atomicMin(int* address, int val);`
- `int atomicAnd(int* address, int val);`
- `int atomicXor(int* address, int val);`
- `int atomicCAS(int* address, int compare, int val);`
- `...`

# atomicAdd для double (CAS-based)

```
__device__ double atomicAdd(double* address, double val)
{
    unsigned long long int* addr = (unsigned long long int*)address;
    unsigned long long int old = *addr, assumed;

    do {
        assumed = old;
        old = atomicCAS(addr, assumed,
                        __double_as_longlong(val +
                        __longlong_as_double(assumed)));
    } while (assumed != old);

    return __longlong_as_double(old);
}
```

**int atomicCAS(int\* address, int compare, int val);**

1. Загружает в old значение по адресу address
2. Записывает обратно значение:  
(old == compare) ? val : old
3. Возвращает old

# Гистограмма цветов изображения (GPU)

```
__global__ void hist_gpu_atomic(uint8_t *image, int width,  
                                int height, int *hist)  
{  
    size_t size = width * height;  
    int i = blockIdx.x * blockDim.x + threadIdx.x;  
  
    if (i < size) {  
        atomicAdd(&hist[image[i]], 1);  
    }  
}
```

# Гистограмма цветов изображения (GPU)

```
__global__ void hist_gpu_atomic(uint8_t *image, int width,  
                                int height, int *hist)  
{  
    size_t size = width * height;  
    int i = blockIdx.x * blockDim.x + threadIdx.x;  
    int stride = blockDim.x * gridDim.x;  
  
    while (i < size) {  
        atomicAdd(&hist[image[i]], 1);  
        i += stride;  
    }  
}
```

Если пикселей больше  
числа потоков

# Задание

- **Histogram calculation in CUDA**

[http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86\\_website/projects/histogram64/doc/histogram.pdf](http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/histogram64/doc/histogram.pdf)

- Идея: локальные гистограммы и их слияние + оптимизация работы с памятью