CUDA, Multi-GPU

Вычисления на видеокартах. Лекция 12

1. CUDA и как поддерживать кодовую базу с несколькими API
2. Шаблоны на C99 (для OpenCL 1.2)
3. Multi-GPU + CPU вычисления (а так же SLI и Crossfire)
4. pyOpenCL

Полярный Николай

polarnick239@gmail.com

CUDA

1) **NVCC** компилятор накладывает ограничение на используемые **компиляторы** (в т.ч. ограничение на версии снизу и сверху)
2) Так же есть ограничение на **runtime** окружение:
   чем новее **NVCC** компилятор - тем выше версия драйвера требуется:
   см. таблицу сопоставления:

```
CUDA 10.0: 410.48
CUDA  9.2: 396.xx
CUDA  9.1: 390.xx (update)
CUDA  9.0: 384.xx
CUDA  8.0  375.xx (GA2)
CUDA  8.0: 367.4x
CUDA  7.5: 352.xx
CUDA  7.0: 346.xx
CUDA  6.5: 340.xx
CUDA  6.0: 331.xx
CUDA  5.5: 319.xx
CUDA  5.0: 304.xx
```

CUDA: Example

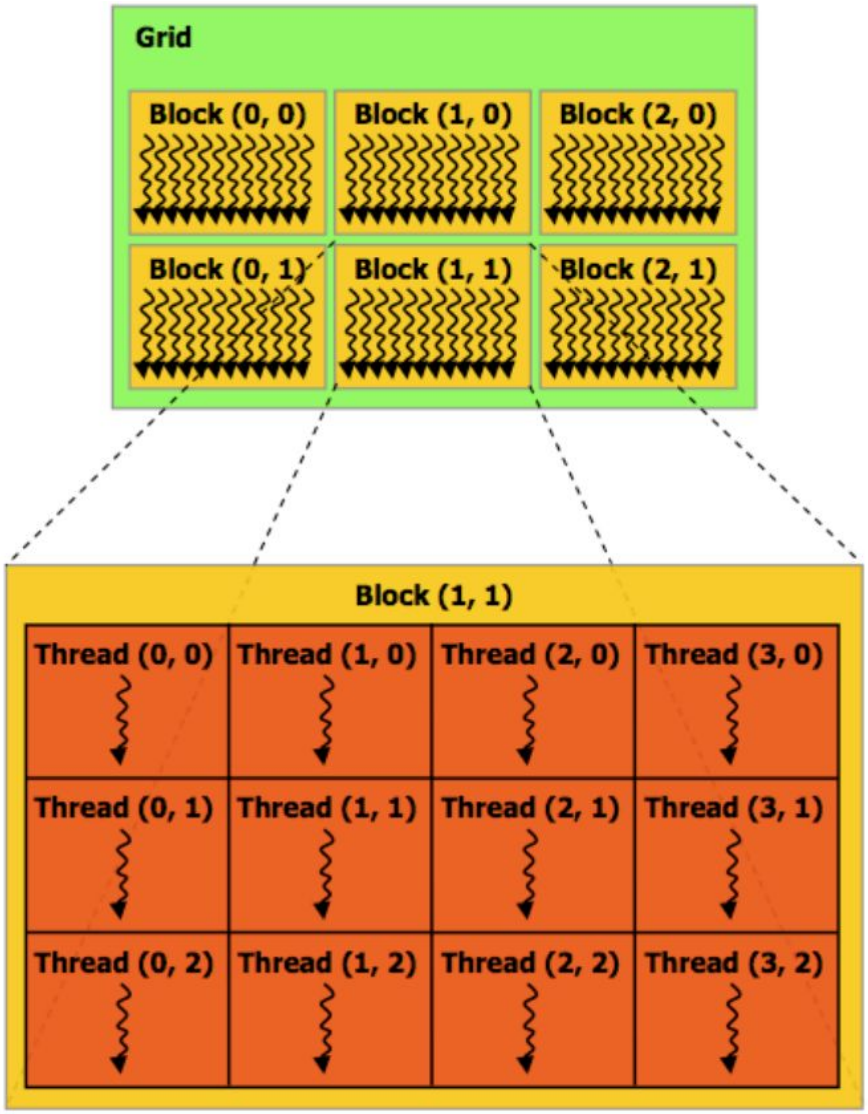
```
__global__ void aplusb(const float* a, const float* b, float* c, unsigned int n) {
    //     get_global_id(0) == get_group_id(0) * get_local_size(0) + get_local_id(0)
    const unsigned int index = blockIdx.x      * blockDim.x        + threadIdx.x;

    if (index < n)
        c[index] = a[index] + b[index];
}

void main(...) {
    ...
    void* gpu_data;
    cudaError_t err = cudaMalloc(&gpu_data, size);
    if (err != cudaSuccess) {
        ...
    }
    ...
    cudaError_t err = cudaMemcpy(data, gpu_data, size, cudaMemcpyHostToDevice);
    ...
    aplusb<<<n/256, 256>>>(a, b, c, n);
    ...
}
```

CMakeLists.txt:

add_executable → cuda_add_executable

add_library → cuda_add_library

CUDA: Device side



OpenCL⇔CUDA

`get_local_id(0)`	`threadIdx.x`
`get_local_size(0)`	`blockDim.x`
`get_group_id(0)`	`blockIdx.x`
`get_global_id(0)`	`blockIdx.x * blockDim.x + threadIdx.x`
`get_num_groups(0)`	`gridDim.x`
`get_global_size(0)`	`gridDim.x * blockDim.x`

Подробнее: opencl to cuda header

OpenCL + CUDA: общая кодовая база

1) Исходники кернела:
   OpenCL - **aplusb.cl**
   CUDA - **aplusb.cu**:

Пример: a+b OpenCL and CUDA

```
#include <libgpu/cuda/cu/opencl_translator.cu>
#include "../cl/aplusb.cl"

void cuda_aplusb(const gpu::WorkSize &workSize,
                 const float* a, const float* b, float* c, unsigned int n) {
    aplusb<<<workSize.cuGridSize(), workSize.cuBlockSize()>>>(a, b, c, n);
}
```

2) API-вызовы (аллокации, копирования памяти) спрятаны внутри библиотеки

3) Вызов кернела:

```
if (device.supports_cuda)
    cuda_aplusb(workSize, as_gpu.cuptr(), bs_gpu.cuptr(), cs_gpu.cuptr(), n);
else
    aplusb.exec(workSize, as_gpu, bs_gpu, cs_gpu, n);
```

Шаблоны на C99 (OpenCL 1.2)

```
float adopt_color_uint(unsigned int intensity) {
    return intensity / 256.0f;
}

float adopt_color_float(float intensity) {
    return intensity;
}
// T_DEPENDENT(function_name) -> function_name_T
#define T_DEPENDENT2(fun, suffix) fun ## _ ## suffix
#define T_DEPENDENT1(fun, suffix) T_DEPENDENT2(fun, suffix)
#define T_DEPENDENT(fun) T_DEPENDENT1(fun, T)


__kernel void T_DEPENDENT(photometricGradient)(__global const T* image) {
    ...
    float value = T_DEPENDENT(adopt_color)(image[...]);
    ...
}
```

Шаблоны на C99 (OpenCL 1.2)

```
// T1_T2_DEPENDENT(function_name) -> function_name_T1_T2
#define T1_T2_DEPENDENT2(fun, suffix1, suffix2) fun ## _ ## suffix1 ## _ ## suffix2
#define T1_T2_DEPENDENT1(fun, suffix1, suffix2) T1_T2_DEPENDENT2(fun, suffix1, suffix2)
#define T1_T2_DEPENDENT(fun) T1_T2_DEPENDENT1(fun, T1, T2)

#define T2 float
#define T1 char
#include "convert.cl"
#undef T1
#undef T2
```

Multi-GPU rasterization: Crossfire & SLI

SLI (Scan-Line Interleave) - метод распределенной на несколько видеокарт растеризации, ввела **3dfx Interactive** в 1998 (позднее была куплена **NVIDIA**).

Multi-GPU rasterization: Crossfire & SLI

SLI (Scan-Line Interleave) - метод распределенной на несколько видеокарт растеризации, ввела **3dfx Interactive** в 1998 (позднее была куплена **NVIDIA**).

Проблема: обработка геометрии делается полностью на каждой видеокарте, т.е. нелинейная масштабируемость.

Поэтому **SLI (Scalable Link Interface, NVIDIA) и CrossFire (AMD)** используют другой метод - **AFR (Alternate Frame Rendering).**

PyOpenCL: пример

```
import pyopencl as cl
ctx = cl.create_some_context(interactive=True)
queue = cl.CommandQueue(ctx)

program = cl.Program(ctx, """
#line 20
__kernel void match(
__global const int *xs, int n, int k, __global int *res)
{
    int global_i = get_global_id(0);
    ...
mf = cl.mem_flags
xs_cl = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=xs)
res_cl = cl.Buffer(ctx, mf.COPY_HOST_PTR, hostbuf=np.int32([0]))
...
```