OpenCL介绍

什么是OpenCL

OpenCL（Open Computing Language），即开放运算语言，是一个统一的开放式的开发平台。

OpenCL是首个提出的并行开发的开放式的、兼容的、免费的标准。

它的目的是为异构系统通用提供统一开发平台。

OpenCL最初是由苹果公司设想和开发，并在与AMD，IBM，英特尔和NVIDIA技术团队的合作之下初步完善。随后，苹果将这一草案提交至Khronos Group。

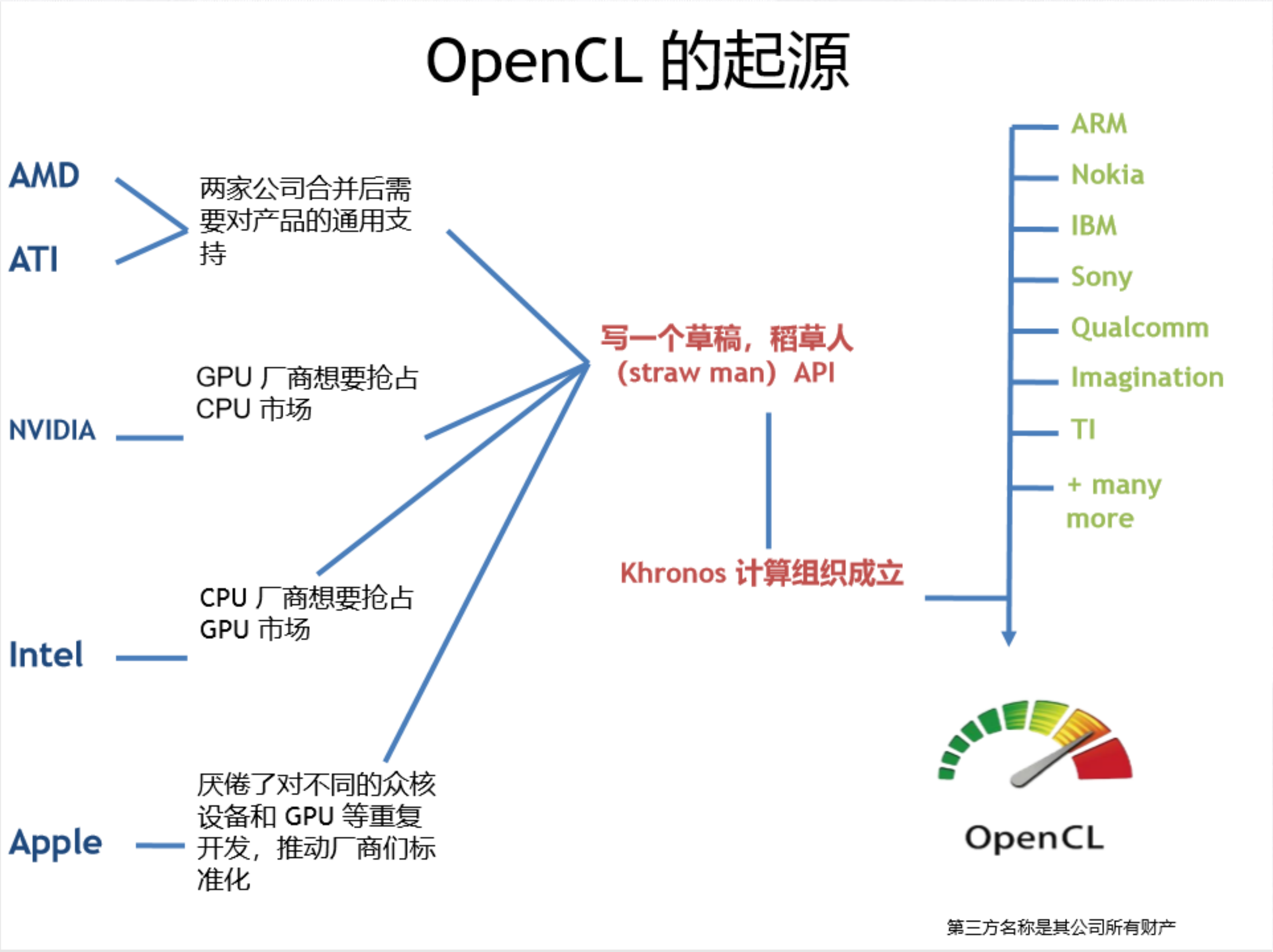
参见

https://www.jianshu.com/p/b55552ee61ac

https://www.cnblogs.com/wangshide/archive/2012/01/07/2315830.html

https://blog.csdn.net/eric41050808/article/details/10210025

OpenCL



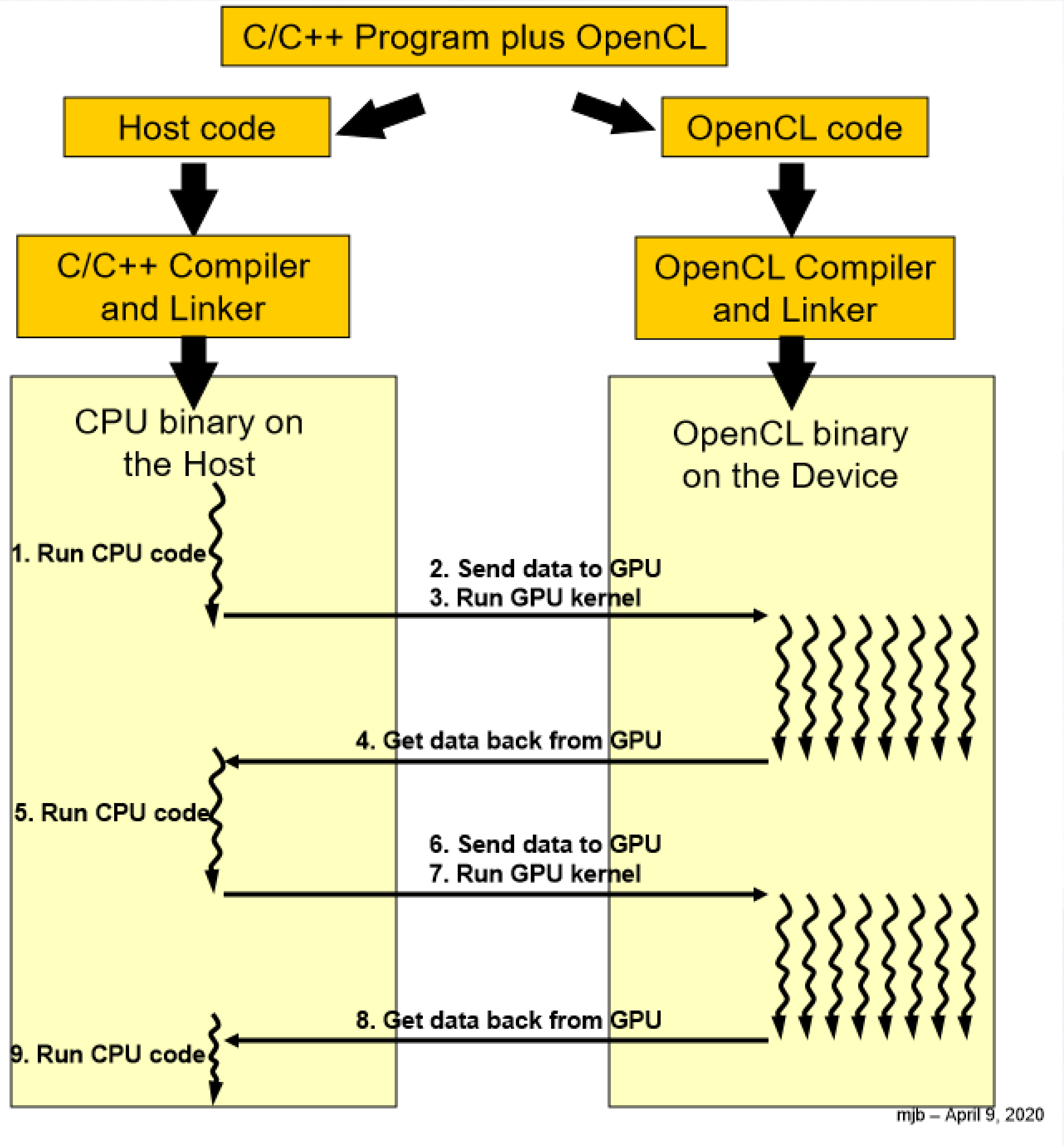
现代的计算平台包括

- 一个或多个 CPU
- 一个或多个 GPU
- DSP 处理器
- 加速器(accelerator)
- ... 其他的

OpenCL 允许开发者写一份可以移植的代码，在不同平台上都可以使用所有的资源。

OpenCL

OpenCL的例子



什么是CUDA

CUDA（Compute Unified Device Architecture），即计算统一设备架构

显卡厂商NVIDIA推出的运算平台，也是一种由NVIDIA推出的通用并行计算架构

用于NVIDIA设备，能够解决复杂的计算问题

参见

https://www.cnblogs.com/skyfsm/p/9673960.html

OpenCL

与CUDA的对比

	CUDA	OpenCL
是什么	硬件架构，指令集架构（ISA），编程语言，API，SDK，工具	开放的API和语言标准
是否自由	非自由	自由，买断式授权
引入时间	2006	2008
SDK供应商	NVIDIA	具体根据企业
实现企业	NVIDIA	Apple、NVIDIA、AMD、IBM
支持系统	Windows, Linux, Mac OS X; 32 and 64-bit	依赖具体企业
支持设备类型	NVIDIA GPU	多种类型
支持嵌入式设备	不支持	支持

OpenCL

CPU

```
// function to add the elements of two arrays
void add(int n, double *x, double *y, double *z)
{
  for (int i = 0; i < n; i++)
      z[i] = x[i] + y[i];
}
```

CUDA

```
__global__
void add(int n, double *x, double *y, double *z)
{
    // compute using the whole grid at once
    // the reuse the threads in the same grid
    // assume we have 1k elements to compute, and a stride of 100
    // thread 0 is responsible for c[0] = a[0] + b[0]
    //                             c[100] = a[100] + b[100]...
    // thread 1 is responsible for c[1] = a[1] + b[1]
```

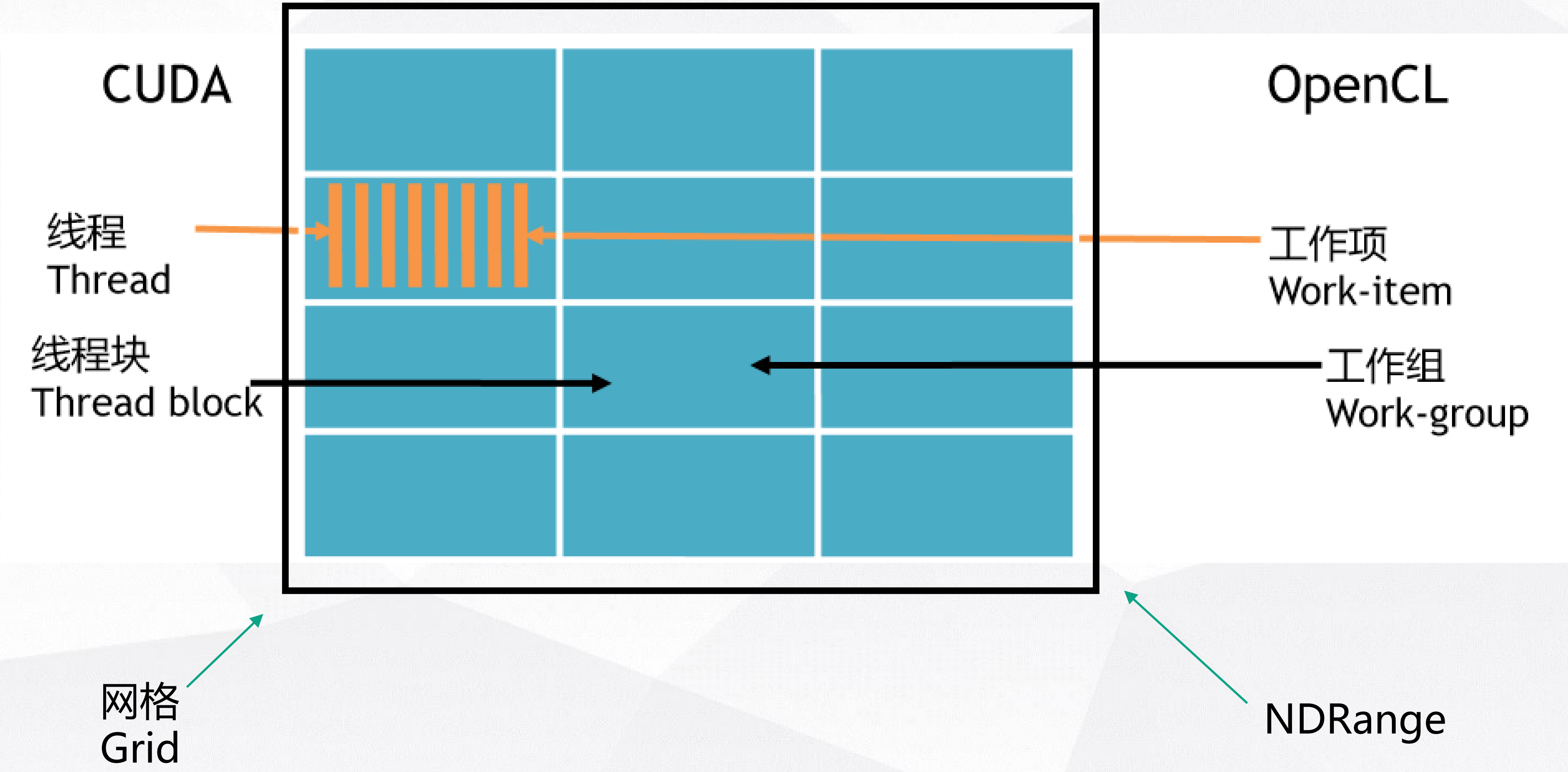
```
    //                             c[101] = a[101] + b[101]...
    // and so on
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    int stride = blockDim.x * gridDim.x;
    for(int i=index;i<n;i+=stride)
        z[i] = x[i] + y[i];
}
```

OpenCL

```
__kernel void adder(__global const float* a, __global const float* b,
__global float* result)
{
    int idx = get_global_id(0);
    result[idx] = a[idx] + b[idx];
}
```

OpenCL

与CUDA的一些对比



OpenCL

CUDA

```
int main(void)
{
  int N = 1<<20;
  double *x, *y,*z;

  // Allocate Unified Memory – accessible from CPU or GPU
  cudaMallocManaged(&x, N*sizeof(double));
  cudaMallocManaged(&y, N*sizeof(double));
  cudaMallocManaged(&z, N*sizeof(double));

  // initialize x and y arrays on the host
  for (int i = 0; i < N; i++) {
    x[i] = std::rand();
    y[i] = std::rand();
  }

  // Run kernel on 1M elements on the GPU
  int blockSize = 256;
  int numBlocks = (N+blockSize-1)/blockSize;
  add<<<numBlocks, blockSize>>>(N, x, y, z);

  // Wait for GPU to finish before accessing on host
  cudaDeviceSynchronize();

  // Check for errors
  double maxError = 0;
  for (int i = 0; i < N; i++)
    maxError = fmax(maxError, fabs(z[i]-x[i]-y[i]));
  std::cout << "Max error: " << maxError << std::endl;

  // Free memory
  cudaFree(x);
  cudaFree(y);
  cudaFree(z);

  return 0;
}
```

OpenCL

1. 引入头文件

```
#ifdef __APPLE__
#include <OpenCL/opencl.h>
#else
#include <CL/cl.h>
#endif
```

2. 获取系统上所有OpenCL平台

```
cl_int err;
cl_uint num;
err = clGetPlatformIDs(0, 0, &num);
if(err != CL_SUCCESS) {
    std::cerr << "Unable to get platforms\n";
    return 0;
}
```

3. 取得平台的ID，用于建立OpenCL 上下文（context）

```
std::vector<cl_platform_id> platforms(num);
err = clGetPlatformIDs(num, &platforms[0], &num);
if(err != CL_SUCCESS) {
    std::cerr << "Unable to get platform ID\n";
    return 0;
}
```

4. 建立OpenCL 上下文

```
cl_context_properties prop[] = { CL_CONTEXT_PLATFORM,
reinterpret_cast<cl_context_properties>(platforms[0]), 0 };
cl_context context = clCreateContextFromType(prop, CL_DEVICE_TYPE_DEFAULT,
NULL, NULL, NULL);
if(context == 0) {
    std::cerr << "Can't create OpenCL context\n";
    return 0;
}
```

5. 取得OpenCL设备列表

```
size_t cb;
clGetContextInfo(context, CL_CONTEXT_DEVICES, 0, NULL, &cb);
std::vector<cl_device_id> devices(cb / sizeof(cl_device_id));
clGetContextInfo(context, CL_CONTEXT_DEVICES, cb, &devices[0], 0);
```

6. 选定OpenCL设备

```
clGetDeviceInfo(devices[0], CL_DEVICE_NAME, 0, NULL, &cb);
std::string devname;
devname.resize(cb);
clGetDeviceInfo(devices[0], CL_DEVICE_NAME, cb, &devname[0], 0);
std::cout << "Device: " << devname.c_str() << "\n";
```

7. 建立命令队列（接收操作）

```
cl_command_queue queue = clCreateCommandQueue(context, devices[0], 0, 0);
if(queue == 0) {
    std::cerr << "Can't create command queue\n";
    clReleaseContext(context);
    return 0;
}
```

OpenCL

8. 配置内存并复制数据

```
const int DATA_SIZE = 1048576;
std::vector<float> a(DATA_SIZE), b(DATA_SIZE), res(DATA_SIZE);
for(int i = 0; i < DATA_SIZE; i++) {
    a[i] = std::rand();
    b[i] = std::rand();
}

cl_mem cl_a = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, sizeof(cl_float) * DATA_SIZE, &a[0], NULL);
cl_mem cl_b = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, sizeof(cl_float) * DATA_SIZE, &b[0], NULL);
cl_mem cl_res = clCreateBuffer(context, CL_MEM_WRITE_ONLY, sizeof(cl_float) * DATA_SIZE, NULL, NULL);
if(cl_a == 0 || cl_b == 0 || cl_res == 0) {
    std::cerr << "Can't create OpenCL buffer\n";
    clReleaseMemObject(cl_a);
    clReleaseMemObject(cl_b);
    clReleaseMemObject(cl_res);
    clReleaseCommandQueue(queue);
    clReleaseContext(context);
    return 0;
}
```

创建program object

```
cl_program program = load_program(context, "shader.cl");
if(program == 0) {
    std::cerr << "Can't load or build program\n";
    clReleaseMemObject(cl_a);
    clReleaseMemObject(cl_b);
    clReleaseMemObject(cl_res);
    clReleaseCommandQueue(queue);
    clReleaseContext(context);
    return 0;
}
```

取得程序中函数的进入点

```
cl_kernel adder = clCreateKernel(program, "adder", 0);
if(adder == 0) {
    std::cerr << "Can't load kernel\n";
    clReleaseProgram(program);
    clReleaseMemObject(cl_a);
    clReleaseMemObject(cl_b);
    clReleaseMemObject(cl_res);
    clReleaseCommandQueue(queue);
    clReleaseContext(context);
    return 0;
}
```

10. 执行OpenCL kernel

设定参数

```
clSetKernelArg(adder, 0, sizeof(cl_mem), &cl_a);
clSetKernelArg(adder, 1, sizeof(cl_mem), &cl_b);
clSetKernelArg(adder, 2, sizeof(cl_mem), &cl_res);
```

开始执行

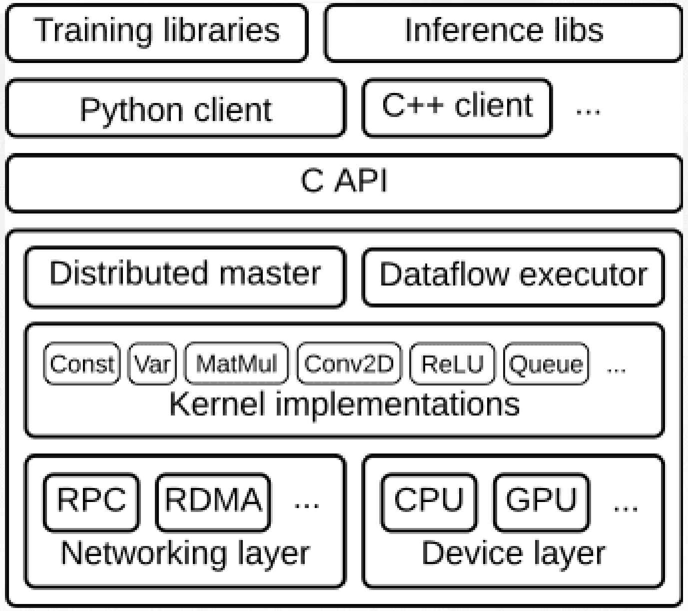
```
size_t work_size = DATA_SIZE;
err = clEnqueueNDRangeKernel(queue, adder, 1, 0, &work_size, 0, 0, 0, 0);
if(err == CL_SUCCESS) {
    err = clEnqueueReadBuffer(queue, cl_res, CL_TRUE, 0, sizeof(float) * DATA_SIZE, &res[0], 0, 0, 0);
}
```

OpenCL

与CUDA的一些对比

CUDA	OpenCL
gridDim	get_num_groups()
blockIdx	get_group_id()
blockDim	get_local_size()
gridDim * blockDim	get_global_size()
threadIdx	get_local_id()
blockIdx * blockdim + threadIdx	get_global_id()

Tensorflow kernel



kernel是Tensorflow中操作的具体实现

例如图中的Matmul（矩阵乘法）、Conv2D（二维卷积）

这些操作可以在CPU上进行，也可以在GPU上运行

https://blog.csdn.net/u013510838/article/details/84103503

相关资料

CUDA与OpenCL

https://www.sharcnet.ca/help/index.php/Porting_CUDA_to_OpenCL（API对比）

https://www.cnblogs.com/huliangwen/p/5003504.html （概念对比）

https://blog.csdn.net/ijuliet/article/details/4631214（代码对比）

OpenCL代码详情

https://blog.csdn.net/breakawayroad/article/details/8227450（修饰符）

https://www.khronos.org/registry/OpenCL/sdk/1.2/docs/man/xhtml（API）

Thank you for watching