Лекция 5

- PTX (*Parallel Thread eXecution*) ISA (*Instruction Set Architecture*).
- CUDA Driver API.

```cpp
#include <cuda.h>
#include <cuda_runtime.h>
#include <stdio.h>
#include <malloc.h>

int main(){
    cuInit(0);
    CUdevice cuDevice;
    CUresult res = cuDeviceGet(&cuDevice, 0);
    if (res != CUDA_SUCCESS){
        printf("cannot acquire device 0\n");
        exit(1);
    }
```

cda.cpp

```
CUcontext cuContext;
res = cuCtxCreate(&cuContext, 0, cuDevice);
if (res != CUDA_SUCCESS){
    printf("cannot create context\n");
    exit(1);
}
```

```
int N=2048;
float* a=(float*)calloc(N, sizeof(float));
float* b=(float*)calloc(N, sizeof(float));

for(int i=0; i<N; i++){
    a[i]=2*i;
    b[i]=2*i+1;
}

float *a_d, *b_d;
cudaMalloc((void**)&a_d, N*sizeof(float));
cudaMalloc((void**)&b_d, N*sizeof(float));

cudaMemcpy(a_d, a, N*sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(b_d, b, N*sizeof(float), cudaMemcpyHostToDevice);
```

```
//gStub<<<N/128,128>>>(a_d,b_d);
//cudaDeviceSynchronize();

CUmodule cuModule = (CUmodule)0;
cuModuleLoad(&cuModule, "cda.ptx");
CUfunction gStub;
cuModuleGetFunction(&gStub, cuModule, "gStub");

void* args[] = {&a_d, &b_d};
cuLaunchKernel(gStub, N/128, 1, 1, 128, 1, 1, 0, 0, args, 0);
```

```
cudaMemcpy(a, a_d, N*sizeof(float), cudaMemcpyDeviceToHost);

for(int i=0; i<N; i+=N/16)
    printf("%g\n",a[i]);


  cuCtxDestroy(cuContext);
  return 0;
}
```

```
/*
    __global__ void gStub(float* a, float* b){
        int tid=threadIdx.x+blockIdx.x*blockDim.x;
        a[tid]=a[tid]+b[tid];
    }
*/
```

cda.cu

```
.version 8.5
.target sm_52
.address_size 64


.visible .entry gStub(
    .param .u64 gStub_param_0,
    .param .u64 gStub_param_1
)
{
    .reg .f32    %f<4>;
    .reg .b32    %r<5>;
    .reg .b64    %rd<8>;


    ld.param.u64    %rd1, [gStub_param_0];
    ld.param.u64    %rd2, [gStub_param_1];
```

cda.ptx

```
        cvta.to.global.u64    %rd3, %rd2;
        cvta.to.global.u64    %rd4, %rd1;
        mov.u32      %r1, %tid.x;
        mov.u32      %r2, %ctaid.x;
        mov.u32      %r3, %ntid.x;
        mad.lo.s32  %r4, %r2, %r3, %r1;
        mul.wide.s32    %rd5, %r4, 4;
        add.s64      %rd6, %rd4, %rd5;
        ld.global.f32      %f1, [%rd6];
        add.s64      %rd7, %rd3, %rd5;
        ld.global.f32      %f2, [%rd7];
        add.f32      %f3, %f1, %f2;
        st.global.f32      [%rd6], %f3;
        ret;
}
```

```
tests/cudrapi> g++ -I/usr/local/cuda/include
-L/usr/local/cuda/lib64 -lcudart -lcuda cda.cpp -o cda
```