1. 第一个 gpu 程序

* 平方和

\_\_global\_\_ voidsquare(float\* d\_out, float\* d\_in){ int idx = threadIdx.x; float f = d\_in[idx]; d\_out[idx] = f\*f; } intmain(int argc, char\*\* argv){ constint ARRAY\_SIZE = 64; constint ARRAY\_BYTES = ARRAY\_SIZE \* sizeof(float); float h\_in{ARRAY\_SIZE}; for(int i=0; i<ARRAY\_SIZE ; i++){ h\_in[i] = float(i); } float h\_out{ARRAY\_SIZE}; // declare GPU memory pointersfloat\* d\_in; float\* d\_out; cudaMalloc((void\*\*) &d\_in, ARRAY\_SIZE); cudaMalloc((void\*\*) &d\_out, ARRAY\_SIZE); cudaMemcpy(d\_int, h\_in, ARRAY\_BYTES, cudaMemcpyHostToDevice); square<<<1, ARRAY\_SIZE>>>(d\_out, d\_in); cudaMemcpy(h\_out, d\_out, ARRAY\_SIZE, cudaMemcpyDeviceToHost); for(int i =0; i<ARRAY\_SIZE; i++){ } cudaFree(d\_in); cudaFree(d\_out); }

* kernel 加载 - 1d 模式

1. 网格 (grid) 是 1d 的，线程(block)是1D int idx = blockIdx.x\*blockDim.x+threadIdx.x; //加载方式 Kernel<<<numBlock, threadxPerBlock>>>(argv) 2. 网格是 1 D，线程块是2D int idx = blockIdx.x\*blockDim.x\*blockDim.y + threadIdx.y\*blockDim.x+threadIdx.x; //加载方式dim3 dmBlock(x,y); Kernel<<<numBlock, dimBlock>>>(argv); 3. 网格是1d的， 线程块是3d的 int idx = blockIdx.x\*blockDim.x\*blockDim.y\*blockDim.z+threadIdx.z\*blockDim.y\*blockDim.x+threadIdx.y\*blockDim.x+threadIdx.x 加载方式 dim3 dimBlock(x,y,z) Kernel<<<numBlock, dimBlock>>>(argv)

* kernel 加载 - 2d 模式

1.网格是2d,线程块是1d的 int blockId = blockIdx.y \* gridDim.x + blockIdx.x; int Idx = blockId \* blockDim.x + threadIdx.x; //加载dim3 dimGrid(x,y); Kernel<<<dimGrid, threadsPerBlock>>>(argv); 2. 网格是2d，线程块是2d int blockId = blockIdx.y \* gridDim.x + blockIdx.x; int Idx = blockId \* (blockDim.x \* blockDim.y) + (threadIdx.y \* blockDim.x) + threadIdx.x; //加载dim3 dimGrid(x1,y1), dimBlock(x2, y2); Kernel<<<dimGrid, dimBlock>>>(argv); 3. 网格是 2d 线程块是3d int blockId = blockIdx.y \* gridDim.x + blockIdx.x; int Idx = blockId \* (blockDim.x \* blockDim.y \* blockDim.z) + (threadIdx.z \* (blockDim.x\*blockDim.y) )+ (threadIdx.y\*blockDim.x) + threadIdx.x; //加载dim3 dimGrid(x1, y1), dimBlock(x2, y2, z2); Kernel<<<dimGrid, dimBlock>>>(argv);

## 3.cuda 中的各种内存代码的使用

### 3.1 本地变量

\_\_global\_\_ voiduse\_local\_memory\_GPU(float in){ float f; f = in; } use\_local\_memory\_GPU<<<1,128>>>(2.0f);

### 3.2 全局变量

\_\_global\_\_ voiduse\_global\_memory\_GPU(float \*array){ array[threadIdx.x] = 2.0f \* (float) threadIdx.x; } float h\_arr[128]; float \*d\_arr; cudaMolloc((void\*\*) &d\_arr, sizeof(float)\*128)； cudaMemcpy((void\*)d\_arr, (void \*)h\_arr, sizeof(float)\*128, cudaMemcpyHostToDevice); use\_global\_memory\_GPU<<<1,128>>>(d\_arr); cudaMemcpy((void\*)h\_arr, (void\*)d\_arr,sizeof(float)\*128, cudaMemcpyDeviceToHost);

### 3.3 共享变量

\_\_global\_\_ voiduse\_shared\_memory\_GPU(float \*array){ int i, index = threadIdx.x; float average , sum = 0.0f; //lifetime as the thread block sh\_arr[index] = array[index]; \_\_syncthreads(); for(i=0; i<index; i++){sum+=sh\_arr[i];} average = sum /(index + 1.0f); if(array[index] > average){array[index] = average;} sh\_arr[index] = 3.14; } use\_shared\_memory\_GPU<<<1,128>>>(d\_arr); cudaMemcpy((void\*)h\_arr, (void\*)d\_arr, sieof(float)\*128, cudaMemcpyHostToDevice);