CUDA C编程：cublas与cudnn介绍与使用

课程目标

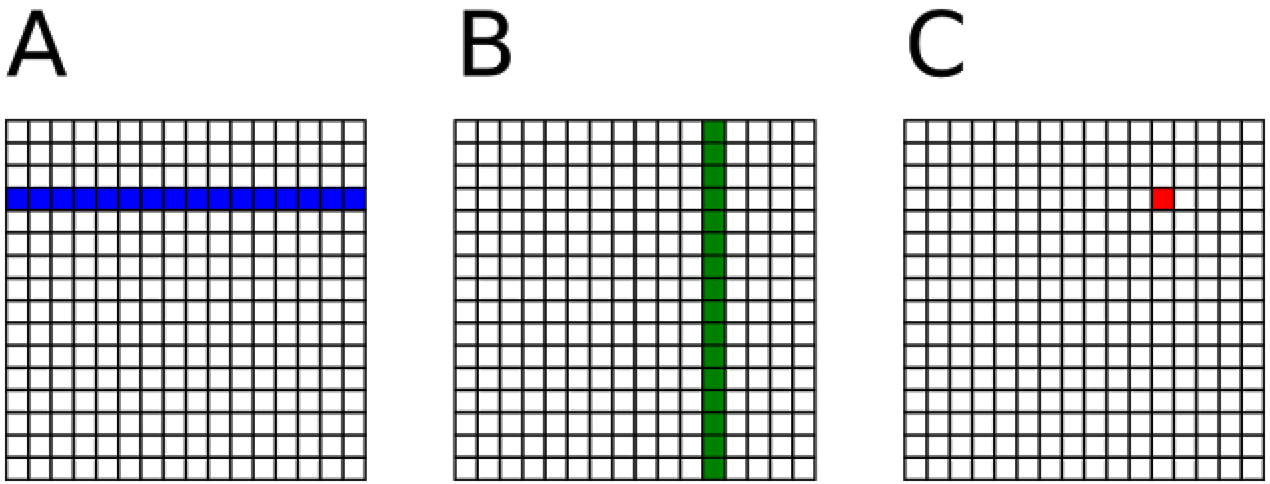
理论部分

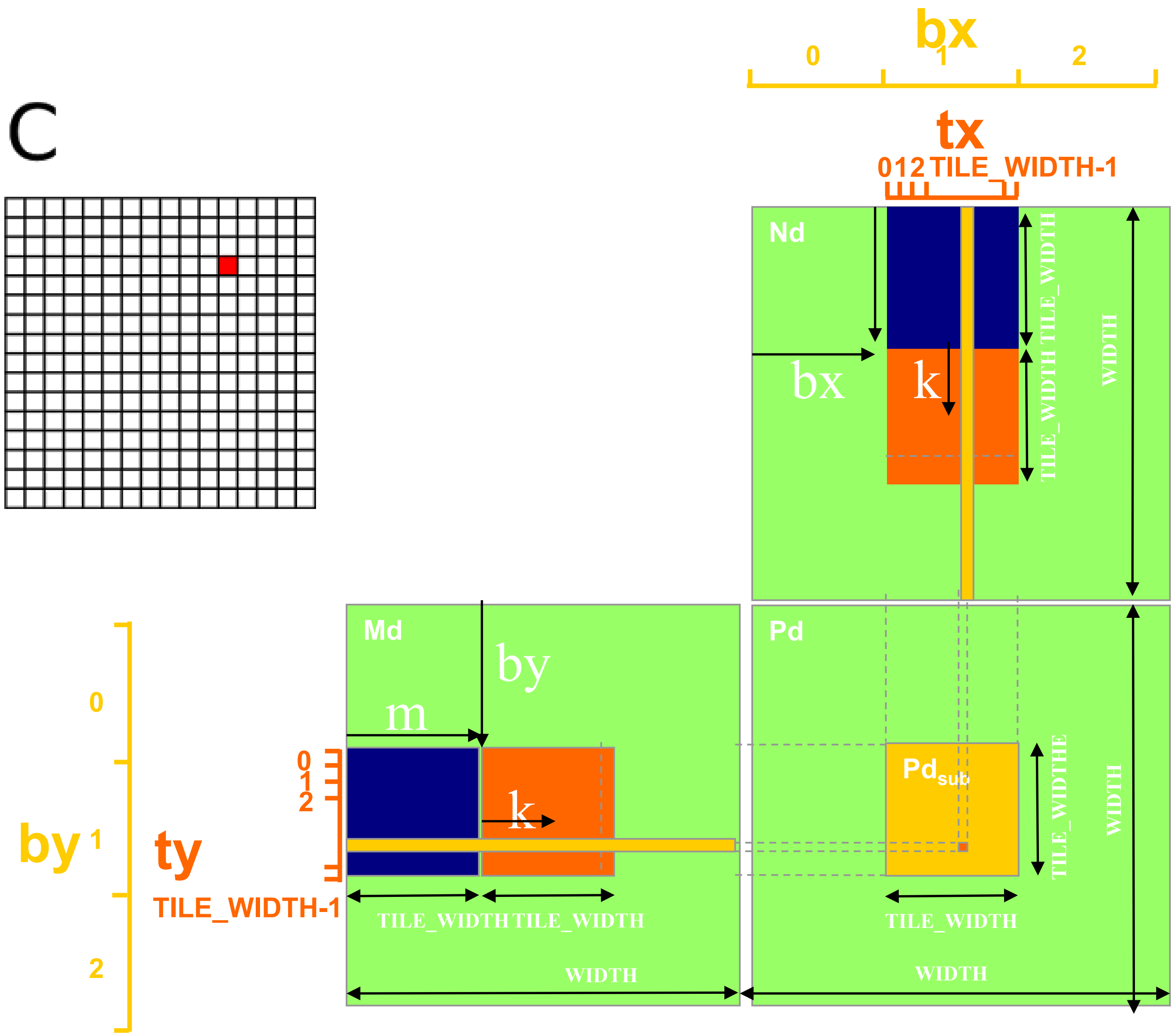
- 进一步认识GPU并行原理

技能部分

- Cublas常用函数
- Cudnn常用函数

并行计算实例：矩阵乘法


A
B
C


bx
0
1
2
tx
0 1 2 TILE_WIDTH-1
Nd
bx
k
TILE_WIDTH
TILE_WIDTH
WIDTH
Md
by
m
k
0
1
2
by
0
1
2
ty
TILE_WIDTH-1
TILE_WIDTH
TILE_WIDTH
WIDTH
Pd
Pdsub
TILE_WIDTHE
TILE_WIDTH
WIDTH
WIDTH

平铺矩阵乘法核函数（伪代码）

```
for tileIdx = 0 to (K/blockDim.x - 1) do

    /* Load one tile of A and one tile of B into shared mem */

    i <= blockIdx.y * blockDim.y + threadIdx.y // Row i of matrix A

    j <= tileIdx * blockDim.x + threadIdx.x // Column j of matrix A

    A_tile(threadIdx.y, threadIdx.x) <= A_gpu(i,j) // Load A(i,j) to shared mem

    B_tile(threadIdx.x, threadIdx.y) <= B_gpu(j,i) // Load B(j,i) to shared mem

    __sync() // Synchronize before computation

    /* Accumulate one tile of C from tiles of A and B in shared mem */

    for k = 0 to threadDim.x do

        accu <= accu + A_tile(threadIdx.y,k) * B_tile(k,threadIdx.x)

    end

    __sync()

end
```

Cublas实现矩阵乘法

- cuBLAS背景：是一个BLAS的实现，允许用户使用NVIDIA的GPU的计算资源。使用cuBLAS 的时候，应用程序应该分配矩阵或向量所需的GPU内存空间，并加载数据，调用所需的cuBLAS函数，然后从GPU的内存空间上传计算结果至主机，cuBLAS API也提供一些帮助函数来写或者读取数据从GPU中。
- 列优先的数组，索引以1为基准
- 头文件 include "cublas_v2.h“
- 三类函数（向量标量、向量矩阵、矩阵矩阵）
- 学习网站： https://docs.nvidia.com/cuda/cublas/index.html
- 30个数：12, 9, 8, 23, 3, 40, 60, 9, 6, 8, 29, 87, 0, 2, 3, 8, 4, 0, 9, 5, 7, 3, 0, 6, 56, 43, 11, 31, 89, 40

主机端：$\begin{bmatrix} 12 & 9 & 8 & 23 & 3 \\ 40 & 60 & 9 & 6 & 8 \\ 29 & 87 & 0 & 2 & 3 \\ 8 & 4 & 0 & 9 & 5 \\ 7 & 3 & 0 & 6 & 56 \\ 43 & 11 & 31 & 89 & 40 \end{bmatrix}$

cuBLAS：$\begin{bmatrix} 12 & 60 & 0 & 9 & 56 \\ 9 & 9 & 2 & 5 & 43 \\ 8 & 6 & 3 & 7 & 11 \\ 23 & 8 & 8 & 3 & 31 \\ 3 & 29 & 4 & 0 & 89 \\ 40 & 87 & 0 & 6 & 40 \end{bmatrix}$

Cublas实现矩阵乘法

- …// 准备 A, B, C 以及使用的线程网格、线程块的尺寸

- // 创建句柄
- cublasHandle_t handle;
- cublasCreate(&handle);

- // 调用计算函数
- cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, m, n, k, &alpha, *B, n, *A, k, &beta, *C, n);

- // 销毁句柄
- cublasDestroy(handle);

- …// 回收计算结果，顺序可以和销毁句柄交换

Cublas实现矩阵乘法

cuBLAS 辅助函数

- 句柄管理函数cublasCreate( ), cublasDestroy( )
- cublasStatus_t cublasCreate(cublasHandle_t *handle)
- cublasStatus_t cublasDestroy(cublasHandle_t handle)
- 初始化CUBLAS库，并为保存CUBLAS库上下文创建一个句柄。 它在主机和设备上分配硬件资源，并且在进行任何其他CUBLAS库调用时必须使用它。 CUBLAS库上下文绑定到当前CUDA设备。要在多个设备上使用该库， 需要为每个设备创建一个CUBLAS句柄。
- 创建句柄的函数 cublasCreate( ) 会返回一个 cublasStatus_t 类型的值， 用来判断句柄是否创建成功
- 流管理函数cublasSetStream(), cublasGetStream()
- cublasStatus_t cublasSetStream(cublasHandle_t handle, cudaStream_t streamId)
- cublasStatus_t cublasGetStream(cublasHandle_t handle, cudaStream_t *streamId)

Cublas level1函数：标量

- cublasStatus_t cublasIsamax(cublasHandle_t handle, int n, const float *x, int incx, int *result)
- cublasStatus_t cublasIsamin(cublasHandle_t handle, int n, const float *x, int incx, int *result)
- 实现功能： result = max/min(x)
- 参数意义
  - Incx：x的存储间隔

Cublas level2函数：矩阵向量

- cublasStatus_t cublasSgemv(cublasHandle_t handle, cublasOperation_t trans, int m, int n, const float *alpha, const float *A, int lda, const float *x, int incx, const float *beta, float *y, int incy)
- 实现功能： y = alpha * op (A) * x + beta * y
- 参数意义
  - Lda：A的leading dimension，若转置按行优先，则leading dimension为A的列数
  - Incx/incy：x/y的存储间隔

Cublas level3函数：矩阵矩阵

- cublasStatus_t cublasSgemm(cublasHandle_t handle,

  cublasOperation_t transa, cublasOperation_t transb,

  int m, int n, int k,

  const float *alpha, const float *A, int lda, const float *B, int ldb,

  const float *beta, float*C, int ldc)
- 实现功能： C = alpha * op (A) * op (B) + beta * C
- 参数意义
  - alpha和beta是标量， A B C是以列优先存储的矩阵
  - 如果 transa的参数是CUBLAS_OP_N 则op(A) = A ， 如果是CUBLAS_OP_T 则op(A)=A的转置
  - 如果 transb的参数是CUBLAS_OP_N 则op(B) = B ， 如果是CUBLAS_OP_T 则op(B)=B的转置
  - Lda/Ldb:A/B的leading dimension，若转置按行优先，则leading dimension为A/B的列数
  - Ldc：C的leading dimension，C矩阵一定按列优先，则leading dimension为C的行数

Cublas实现矩阵乘法

```
 float *d_A, *d_B, *d_C;
unsigned int size_C = ms.wc * ms.hc;
unsigned int mem_size_C = sizeof(float) * size_C;
float *h_CUBLAS = (float *) malloc(mem_size_C);
cudaMalloc((void **) &d_A, mem_size_A);
cudaMalloc((void **) &d_B, mem_size_B);
cudaMemcpy(d_A, h_A, mem_size_A, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, mem_size_B, cudaMemcpyHostToDevice);
cudaMalloc((void **) &d_C, mem_size_C);

dim3 threads(1,1);
dim3 grid(1,1);
```

Cublas实现矩阵乘法

```
//cuBLAS代码
const float alpha = 1.0f;
const float beta  = 0.0f;
int m = A.row, n = B.col, k = A.col;

cublasHandle_t handle;
cublasCreate(&handle);

cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, n, m, k, &alpha, d_B, n,
 d_A, k, &beta, d_C, n);//C=AB->C_T=B_T*A_T

cublasDestroy(handle);

cudaMemcpy(h_CUBLAS, d_C, mem_size_C, cudaMemcpyDeviceToHost);
```

Cublas实现矩阵乘法

- cublasStatus_t cublasSgemmBatched(cublasHandle_t handle,

  cublasOperation_t transa, cublasOperation_t transb,

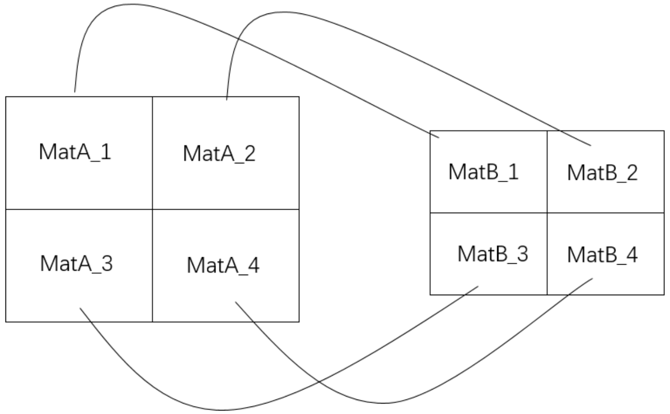
  int m, int n, int k,

  const float *alpha, const float*Aarray[], int lda,

  const float *Barray[], int ldb, const float *beta,

  float*Carray[], int ldc,

  int batchCount)



- 实现功能： C[i] = alpha * op (A[i]) * op (B[i]) + beta * C[i]
- 参数意义
  - Lda/Ldb:A/B的leading dimension，若转置按行优先，则leading dimension为A/B的列数
  - Ldc：C的leading dimension，C矩阵一定按列优先，则leading dimension为C的行数
  - Batchcount:批处理数量

Cublas实现矩阵乘法

- cublasStatus_t cublasSgemmStridedBatched(cublasHandle_t handle,

  cublasOperation_t transa, cublasOperation_t transb,

  int m, int n, int k,

  const float *alpha, const float *A, int lda, long long int strideA,

  const float *B, int ldb, long long int strideB, const float *beta,

  float *C, int ldc, long long int strideC,

  int batchCount)

- 实现功能： C +i*strideC= alpha * op (A+i*strideA) * op (B +i*strideB) + beta * (C +i*strideC)
- 参数意义
  - alpha和beta是标量， A B C是以列优先存储的矩阵
  - 如果 transa的参数是CUBLAS_OP_N 则op(A) = A， 如果是CUBLAS_OP_T 则op(A)=A的转置
  - 如果 transb的参数是CUBLAS_OP_N 则op(B) = B， 如果是CUBLAS_OP_T 则op(B)=B的转置

Cublas实现矩阵乘法

- cublasStatus_t cublasGemmEx(cublasHandle_t handle,

  cublasOperation_t transa, cublasOperation_t transb,

  int m, int n, int k,

  const void *alpha, const void *A, cudaDataType_t Atype, int lda,

  const void *B, cudaDataType_t Btype, int ldb, const void *beta,

  void *C, cudaDataType_t Ctype, int ldc, cudaDataType_t computeType,

  cublasGemmAlgo_t algo)

- 实现功能： C = alpha * op (A) * op (B) + beta * C
- 参数意义
  - alpha和beta是标量， A B C是以列优先存储的矩阵
  - 如果 transa的参数是CUBLAS_OP_N 则op(A) = A ， 如果是CUBLAS_OP_T 则op(A)=A的转置
  - 如果 transb的参数是CUBLAS_OP_N 则op(B) = B ， 如果是CUBLAS_OP_T 则op(B)=B的转置

Cublas实现矩阵乘法

- cublasStatus_t cublasGemmEx(cublas

  cublasOperation_t tra

  int m, int n, int k,

  const void *alpha, co

  const void *B, cudaDa

  void *C, cudaDataTyp

  cublasGemmAlgo_t a

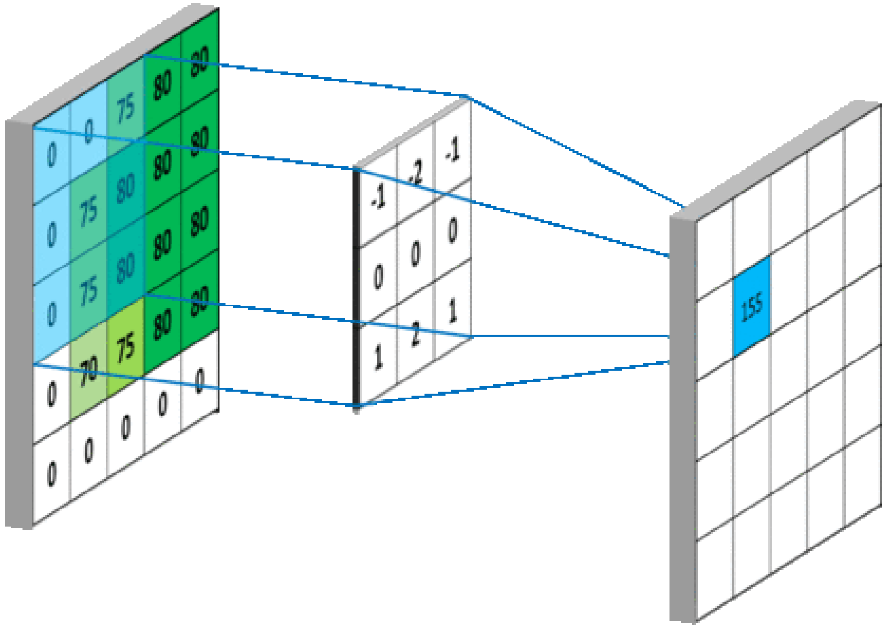
- 实现功能： C = alpha * op (A) * op
- 参数意义
  - alpha和beta是标量， A B C是以
  - 如果 transa的参数是CUBLAS_O
  - 如果 transb的参数是CUBLAS_C

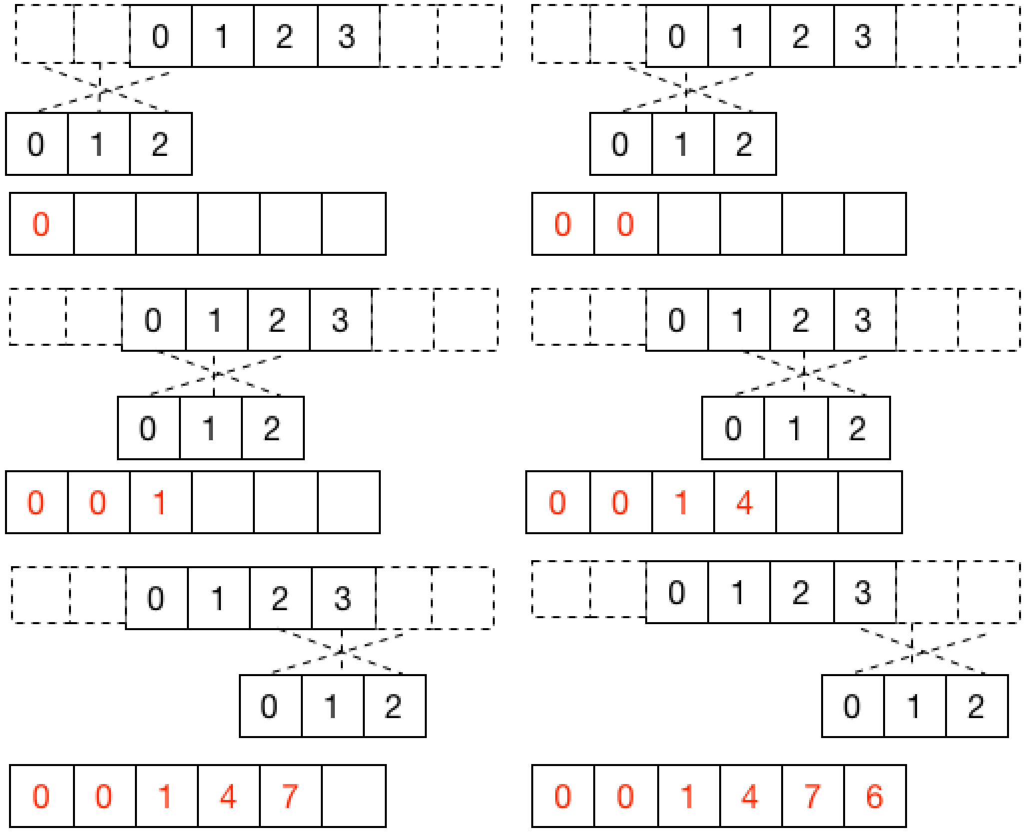
Compute Type	Atype/Btype	Ctype
CUBLAS_COMPUTE_16F or CUBLAS_COMPUTE_16F_PEDANTIC	CUDA_R_16F	CUDA_R_16F
CUBLAS_COMPUTE_32I or CUBLAS_COMPUTE_32I_PEDANTIC	CUDA_R_8I	CUDA_R_32I
CUBLAS_COMPUTE_32F or CUBLAS_COMPUTE_32F_PEDANTIC	CUDA_R_16BF	CUDA_R_16BF
	CUDA_R_16F	CUDA_R_16F
	CUDA_R_8I	CUDA_R_32F
	CUDA_R_16BF	CUDA_R_32F
	CUDA_R_16F	CUDA_R_32F
	CUDA_R_32F	CUDA_R_32F
	CUDA_C_8I	CUDA_C_32F
	CUDA_C_32F	CUDA_C_32F
CUBLAS_COMPUTE_32F_FAST_16F or CUBLAS_COMPUTE_32F_FAST_16BF or CUBLAS_COMPUTE_32F_FAST_TF32	CUDA_R_32F	CUDA_R_32F
	CUDA_C_32F	CUDA_C_32F
CUBLAS_COMPUTE_64F or CUBLAS_COMPUTE_64F_PEDANTIC	CUDA_R_64F	CUDA_R_64F
	CUDA_C_64F	CUDA_C_64F

卷积、常量内存与cudnn

(f*g)(m,n)

$$= \sum_{k=0}^{2} \sum_{h=0}^{2} f(h,k)g(m-h,n-k)$$





https://mlnotebook.github.io/post/CNN1/

Cudnn实现卷积神经网络

- NVIDIA cuDNN是用于深度神经网络的GPU加速库。它强调性能、易用性和低内存开销
- NVIDIA cuDNN可以集成到更高级别的机器学习框架中
- 常用神经网络组件
  - 常用语前向后向卷积网络
  - 前像后向pooling
  - 前向后向softmax
  - 前向后向神经元激活
  - Rectified linear (ReLU)、Hyperbolic tangent (TANH)
  - Tensor transformation functions
  - LRN, LCN and batch normalization forward and backward
- 头文件 include "cudnn.h“
- 学习网站： https://docs.nvidia.com/deeplearning/cudnn/

Cudnn实现卷积神经网络

创建cuDNN句柄

- cudnnStatus_t cudnnCreate(cudnnHandle_t *handle)

以Host方式调用在Device上运行的函数

- 比如卷积运算：cudnnConvolutionForward等

释放cuDNN句柄

- cudnnStatus_t cudnnDestroy(cudnnHandle_t handle)

将CUDA流设置&返回成cudnn句柄

- cudnnStatus_t cudnnSetStream(cudnnHandle_t handle, cudaStream_t streamId)
- cudnnStatus_t cudnnGetStream(cudnnHandle_t handle, cudaStream_t *streamId)

Cudnn实现卷积神经网络

```
#include <iostream>

#include <cudnn.h>

int main(int argc, const char **argv){

    cudnnStatus_t cudnn_re;

    cudnnHandle_t h_cudnn;

    cudnn_re = cudnnCreate(&h_cudnn);

    if(cudnn_re != CUDNN_STATUS_SUCCESS){

        std::cout << "创建cuDNN上下文失败！" << std::endl;

    }

    // =========================================cudnn操作

    // 释放cuDNN

    cudnnDestroy(h_cudnn);

}
```

Cudnn实现卷积神经网络

```
cudnnStatus_t cudnnConvolutionForward(
    cudnnHandle_t                      handle,
    const void                        *alpha,
    const cudnnTensorDescriptor_t      xDesc,
    const void                        *x,
    const cudnnFilterDescriptor_t      wDesc,
    const void                        *w,
    const cudnnConvolutionDescriptor_t  convDesc,
    cudnnConvolutionFwdAlgo_t          algo,
    void                              *workSpace,
    size_t                             workSpaceSizeInBytes,
    const void                        *beta,
    const cudnnTensorDescriptor_t      yDesc,
    void                              *y)
```

Cudnn实现卷积神经网络

```
void cudnn_conv(){
    cudnnStatus_t status; cudnnHandle_t h_cudnn; cudnnCreate(&h_cudnn);
    cudnnTensorDescriptor_t  ts_in, ts_out; // 1. 定义一个张量对象
    status = cudnnCreateTensorDescriptor(&ts_in); // 2. 创建输入张量
    if(CUDNN_STATUS_SUCCESS == status){  std::cout << "创建输入张量成功!" << std::endl; }
    status = cudnnSetTensor4dDescriptor(// 3. 设置输入张量数据
        ts_in,                  // 张量对象
        CUDNN_TENSOR_NHWC,      // 张量的数据布局
        CUDNN_DATA_FLOAT,       // 张量的数据类型
        1,                      // 图像数量
        3,                      // 图像通道
        1080,                   // 图像高度
        1920                    // 图像宽度);
    if(CUDNN_STATUS_SUCCESS == status) std::cout << "创建输出张量成功!" << std::endl;
```

```
cudnnStatus_t cudnnConvolutionForward(
    cudnnHandle_t                       handle,
    const void                         *alpha,
    const cudnnTensorDescriptor_t       xDesc,
    const void                         *x,
    const cudnnFilterDescriptor_t       wDesc,
    const void                         *w,
    const cudnnConvolutionDescriptor_t  convDesc,
    cudnnConvolutionFwdAlgo_t           algo,
    void                               *workSpace,
    size_t                              workSpaceSizeInBytes,
    const void                         *beta,
    const cudnnTensorDescriptor_t       yDesc,
    void                               *y)
```

Cudnn实现卷积神经网络

```
cudnnCreateTensorDescriptor(&ts_out); // 设置输出张量
status = cudnnSetTensor4dDescriptor(ts_out, CUDNN_TENSOR_NHWC, CUDNN_DATA_FLOAT, 1, 3, 1080,
1920);
cudnnFilterDescriptor_t kernel;
cudnnCreateFilterDescriptor(&kernel); // 创建卷积核
status = cudnnSetFilter4dDescriptor(kernel, CUDNN_DATA_FLOAT, CUDNN_TENSOR_NHWC, 3, 3, 3, 3);
cudnnConvolutionDescriptor_t conv; // 创建卷积
status = cudnnCreateConvolutionDescriptor(&conv); // 设置卷积
status = cudnnSetConvolution2dDescriptor(conv, 1, 1, 1, 1, 1, 1, CUDNN_CROSS_CORRELATION,
CUDNN_DATA_FLOAT);
cudnnConvolutionFwdAlgo_t algo;
status = cudnnGetConvolutionForwardAlgorithm(h_cudnn, ts_in, kernel, conv, ts_out,
CUDNN_CONVOLUTION_FWD_PREFER_FASTEST, 0, &algo); // 设置算法
```

Cudnn实现卷积神经网络

```
cudnnCreateTensorDescriptor(&ts_out); // 设置输出张量
status = cudnnSetTensor4dDescriptor(ts_out, CUDNN_T
1920);
cudnnFilterDescriptor_t kernel;
cudnnCreateFilterDescriptor(&kernel); // 创建卷积核
status = cudnnSetFilter4dDescriptor(kernel, CUDNN_DATA_FLOAT, CUDNN_TENSOR_NHWC, 3, 3, 3, 3);
cudnnConvolutionDescriptor_t conv; // 创建卷积
status = cudnnCreateConvolutionDescriptor(&conv); // 设置卷积
status = cudnnSetConvolution2dDescriptor(conv, 1, 1, 1, 1, 1, 1, CUDNN_CROSS_CORRELATION,
CUDNN_DATA_FLOAT);
cudnnConvolutionFwdAlgo_t algo;
status = cudnnGetConvolutionForwardAlgorithm(h_cudnn, ts_
CUDNN_CONVOLUTION_FWD_PREFER_FASTEST, 0, &algo); // 设置算法
```

```
cudnnStatus_t cudnnSetFilterNdDescriptor(
    cudnnFilterDescriptor_t filterDesc,
    cudnnDataType_t         dataType,
    cudnnTensorFormat_t     format,
    int                     nbDims,
    const int               filterDimA[])
```

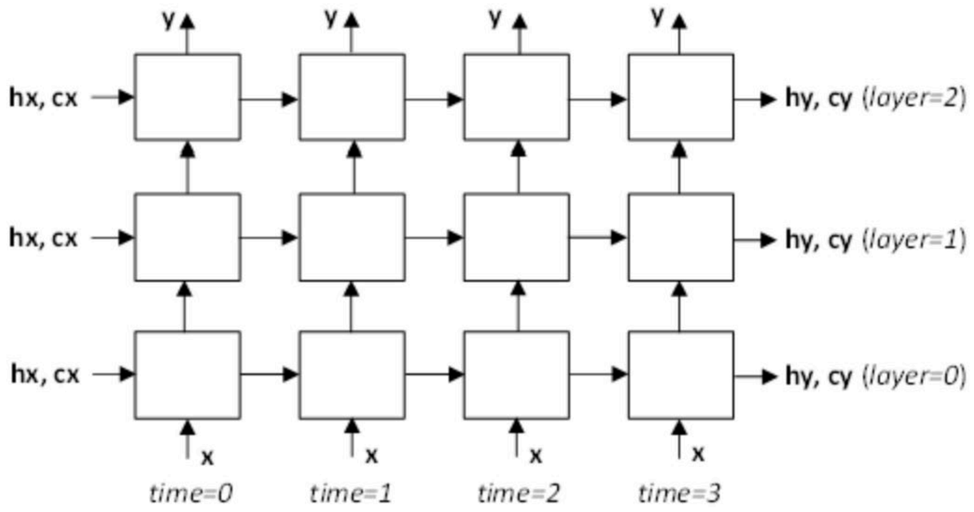
```
cudnnStatus_t cudnnSetConvolution2dDescriptor(
    cudnnConvolutionDescriptor_t    convDesc,
    int                             pad_h,
    int                             pad_w,
    int                             u,
    int                             v,
    int                             dilation_h,
    int                             dilation_w,
    cudnnConvolutionMode_t          mode,
    cudnnDataType_t                 computeType)
```

Cudnn实现卷积神经网络

```
    size_t workspace_size = 0;
    status = cudnnGetConvolutionForwardWorkspaceSize(h_cudnn, ts_in, kernel, conv, ts_out, algo, &workspace_size);
    void * workspace;
    cudaMalloc(&workspace, workspace_size);
    float alpha = 1.0f; float beta  = -100.0f;
    status =  cudnnConvolutionForward(// 卷积执行
        h_cudnn, &alpha, ts_in,
        img_gpu,                      // 输入
        kernel,
        kernel_gpu,                   // 核
        conv, algo, workspace, workspace_size, &beta,
        ts_out, conv_gpu                       // 输出
    );
    cudnnDestroy(h_cudnn);
}
```

Cudnn实现递归神经网络

```
cudnnStatus_t cudnnRNNForward(

    cudnnHandle_t handle, cudnnRNNDescriptor_t rnnDesc, cudnnForwardMode_t fwdMode,

    const int32_t devSeqLengths[],

    cudnnRNNDataDescriptor_t xDesc, const void *x,

    cudnnRNNDataDescriptor_t yDesc, void *y,

    cudnnTensorDescriptor_t hDesc,

    const void *hx, void *hy,

    cudnnTensorDescriptor_t cDesc,

    const void *cx, void *cy,

    size_t weightSpaceSize, const void *weightSpace, size_t workSpaceSize, void *workSpace,

    size_t reserveSpaceSize, void *reserveSpace);
```



Cudnn实现不同激活函数

Softmax

```
cudnnStatus_t cudnnActivationForward( cudnnHandle_t handle, cudnnActivationDescriptor_t
  activationDesc, const void *alpha, const cudnnTensorDescriptor_t xDesc, const void *x,
  const void *beta, const cudnnTensorDescriptor_t yDesc, void *y)
```

Batchnorm

```
cudnnStatus_t cudnnBatchNormalizationForwardInference( cudnnHandle_t handle,
  cudnnBatchNormMode_t mode, const void *alpha, const void *beta, const
  cudnnTensorDescriptor_t xDesc, const void *x, const cudnnTensorDescriptor_t yDesc, void
  *y, const cudnnTensorDescriptor_t bnScaleBiasMeanVarDesc, const void *bnScale, const
  void *bnBias, const void *estimatedMean, const void *estimatedVariance, double epsilon)
```

课程目标

理论部分

- 进一步认识GPU并行原理

技能部分

- Cublas常用函数
- Cudnn常用函数

感谢聆听！

Thanks for Listening