



CUDA 编程基础——利用CUDA实现卷积操作

NVIDIA企业级开发者社区 何琨

AGENDA

流 & CUDA 库

- CUDA流的基本概念
- 利用CUDA流重叠计算和数据传输
- CUDA加速工具库

CUDA 流的概念

- CUDA流在加速应用程序方面起到重要的作用，他表示一个GPU的操作队列，操作在队列中按照一定的顺序执行，也可以向流中添加一定的操作如核函数的启动、内存的复制、事件的启动和结束等，添加的顺序也就是执行的顺序。
- 一个流中的不同操作有着严格的顺序。但是不同流之间是没有任何限制的。多个流同时启动多个内核，就形成了网格级别的并行。
- CUDA流中排队的操作和主机都是异步的，所以排队的过程中并不耽误主机运行其他指令，所以这就隐藏了执行这些操作的开销。

CUDA 流的概念

- 基于流的异步内核启动和数据传输支持以下类型的粗粒度并发
 - 重叠主机和设备计算
 - 重叠主机计算和主机设备数据传输
 - 重叠主机设备数据传输和设备计算
 - 并发设备计算（多个设备）
- 不支持并发：
 - a page-locked host memory allocation,
 - a device memory allocation,
 - a device memory set,
 - a memory copy between two addresses to the same device memory,
 - any CUDA command to the NULL stream

CUDA 流的概念

- 流的创建与销毁
- `cudaError_t cudaMemcpyAsync(void* dst, const void* src, size_t count, cudaMemcpyKind kind, cudaStream_t stream = 0);`
- `cudaError_t cudaStreamCreate(cudaStream_t* pStream);`
- `cudaStream_t a;`
- `kernel_name<<<grid, block, sharedMemSize, stream>>>(argument list);`
- `cudaError_t cudaStreamDestroy(cudaStream_t stream);`

使用CUDA流来加速应用程序

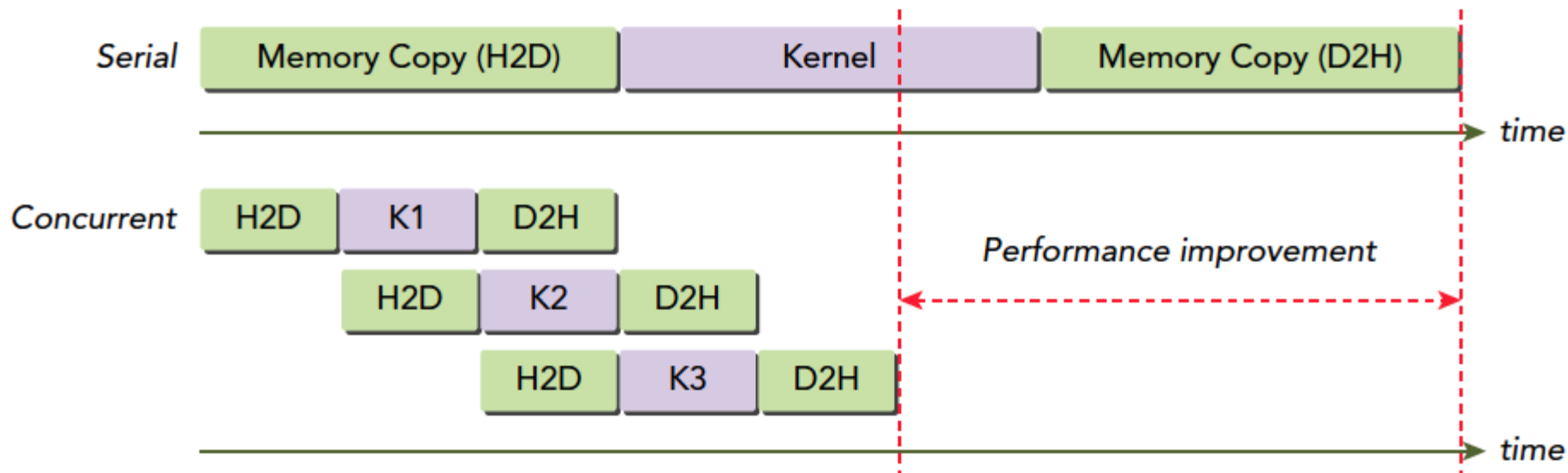







FIGURE 6-1

CUDA加速工具库

GPU Computing Applications						
Libraries and Middleware						
cuDNN TensorRT	cuFFT cuBLAS cuRAND cuSPARSE	CULA MAGMA	Thrust NPP	VSIPL SVM OpenCurrent	PhysX OptiX iRay	MATLAB Mathematica
Programming Languages						
C	C++	Fortran	Java Python Wrappers	DirectCompute	Directives (e.g. OpenACC)	
<div>CUDA-Enabled NVIDIA GPUs</div>						
NVIDIA Ampere Architecture (compute capabilities 8.x)					Tesla A Series	
NVIDIA Turing Architecture (compute capabilities 7.x)			GeForce 2000 Series	Quadro RTX Series	Tesla T Series	
NVIDIA Volta Architecture (compute capabilities 7.x)	DRIVE/JETSON AGX Xavier			Quadro GV Series	Tesla V Series	
NVIDIA Pascal Architecture (compute capabilities 6.x)	Tegra X2		GeForce 1000 Series	Quadro P Series	Tesla P Series	
<div>Embedded</div>		<div>Consumer Desktop/Laptop</div>		<div>Professional Workstation</div>		<div>Data Center</div>

cuBLAS

cuBLAS库是基于NVIDIA®CUDA™运行时的BLAS(Basic Linear Algebra Subprograms)实现

cuBLAS库用于进行向量/矩阵运算，它包含两套API:

- cuBLAS API，需要用户自己分配GPU内存空间，按照规定格式填入数据
- cuBLASXT API，可以分配数据在CPU端，然后调用函数，它会自动管理内存、执行计算

Pyculib是一个包，它提供对几个数值库的访问，这些数值库针对NVidia gpu的性能进行了优化。

Bindings to the following [CUDA libraries](#):

- [cuBLAS](#)
- [cuFFT](#)
- [cuSPARSE](#)
- [cuRAND](#)
- [CUDA Sorting](#) algorithms from the CUB and Modern GPU libraries

cuBLAS

DATA Layout

为了最大限度地兼容现有的Fortran环境，cuBLAS库使用列主存储和基于1的索引

$$\text{IDX2F}(i, j, ld) = (((j) - 1) * (ld)) + ((i) - 1)$$

cuBLAS

Error status

All cuBLAS library function calls return the error status [cublasStatus_t](#)

cuBLAS

cuBLAS context

- 应用程序必须通过调用cublasCreate（）函数初始化cuBLAS库上下文的句柄。
- 这种方法允许用户在使用多个主机线程和多个GPU时显式控制库设置。

cuBLAS

Thread Safety

- 这个库是线程安全的，它的函数可以从多个主机线程调用，即使使用相同的句柄。
- 当多个线程共享同一个句柄时，在更改句柄配置时需要格外小心，因为该更改可能会影响所有线程中后续的CUBLAS调用。
- 因此，不建议多个线程共享相同的CUBLAS handle

cuBLAS

Results reproducibility

- 按照设计，来自给定工具包版本的所有CUBLAS API例程在每次运行时在具有相同架构和相同SMs数量的gpu上执行时都生成相同的位结果。
- 然而，由于实现可能会因一些实现更改而有所不同，因此不能保证跨工具包版本的逐位重现性。

cuBLAS

Parallelism with Streams

- 如果应用程序使用多个独立任务计算的结果，则可以使用CUDA™streams来重叠这些任务中执行的计算。

`cudaStreamCreate()`

`cublasSetStream()`

cuBLAS

Cache configuration

在某些设备上，L1缓存和共享内存使用相同的硬件资源。可以使用CUDA运行时函数`cudaDeviceSetCacheConfig`直接设置缓存配置。还可以使用例程`cudaFuncSetCacheConfig`为某些函数专门设置缓存配置

cuBLAS

cuBLAS Level-1 Function Reference

执行基于标量和向量的操作

cusblas<t>asum()

```
cusblasStatus_t cusblasSasum(cusblasHandle_t handle, int n, const float *x,  
int incx, float *result)
```

```
cusblasStatus_t cusblasDasum(cusblasHandle_t handle, int n, const double *x,  
int incx, double *result)
```

```
cusblasStatus_t cusblasScasum(cusblasHandle_t handle, int n, const cuComplex  
*x, int incx, float *result)
```

```
cusblasStatus_t cusblasDzasum(cusblasHandle_t handle, int n, const  
cuDoubleComplex *x, int incx, double *result)
```

cusblas<t>amax()

cusblas<t>amin()

cusblas<t>asum()

cusblas<t>axpy()

cusblas<t>copy()

cusblas<t>dot()

cusblas<t>nrm2()

cusblas<t>rot()

cusblas<t>rotg()

. cusblas<t>rotm()

. cusblas<t>rotmg()

. cusblas<t>scal()

. cusblas<t>swap()

cuBLAS

cuBLAS Level-2 Function Reference

执行基于矩阵和向量的操作

```
cublasStatus_t cublasSgbmv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n, int kl, int ku,
                           const float *alpha,
                           const float *A, int lda,
                           const float *x, int incx,
                           const float *beta,
                           float *y, int incy)
cublasStatus_t cublasDgbmv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n, int kl, int ku,
                           const double *alpha,
                           const double *A, int lda,
                           const double *x, int incx,
                           const double *beta,
                           double *y, int incy)
cublasStatus_t cublasCgbmv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n, int kl, int ku,
                           const cuComplex *alpha,
                           const cuComplex *A, int lda,
                           const cuComplex *x, int incx,
                           const cuComplex *beta,
                           cuComplex *y, int incy)
cublasStatus_t cublasZgbmv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n, int kl, int ku,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *x, int incx,
                           const cuDoubleComplex *beta,
                           cuDoubleComplex *y, int incy)
```

```
cublas<t>gbmv()
cublas<t>gemv()
cublas<t>ger()
cublas<t>sbmv()
cublas<t>spmv()
cublas<t>spr()
cublas<t>spr2()
cublas<t>symv()
cublas<t>syr()
). cublas<t>syr2()
. cublas<t>tbmvm()
. cublas<t>tbsv()
. cublas<t>tpmv()
. cublas<t>tpsv()
. cublas<t>trmv()
. cublas<t>trsv()
. cublas<t>hemv()
. cublas<t>hbmvm()
. cublas<t>hpmvm()
. cublas<t>her()
. cublas<t>her2()
. cublas<t>hpr()
. cublas<t>hpr2()
```

cuBLAS

cuBLAS Level-3 Function Reference

```
cublasStatus_t cublasSgeam(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n,
                           const float *alpha,
                           const float *A, int lda,
                           const float *beta,
                           const float *B, int ldb,
                           float *C, int ldc)
cublasStatus_t cublasDgeam(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n,
                           const double *alpha,
                           const double *A, int lda,
                           const double *beta,
                           const double *B, int ldb,
                           double *C, int ldc)
cublasStatus_t cublasCgeam(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n,
                           const cuComplex *alpha,
                           const cuComplex *A, int lda,
                           const cuComplex *beta,
                           const cuComplex *B, int ldb,
                           cuComplex *C, int ldc)
cublasStatus_t cublasZgeam(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *beta,
                           const cuDoubleComplex *B, int ldb,
                           cuDoubleComplex *C, int ldc)
```

```
cublas<t>geam()
cublas<t>dgemm()
cublas<t>getrfBatched()
cublas<t>getrsBatched()
cublas<t>getriBatched()
cublas<t>matinvBatched()
cublas<t>geqrfBatched()
cublas<t>gelsBatched()
cublas<t>tptr()
. cublas<t>trttp()
. cublas<t>gemmEx()
. cublasGemmEx()
. cublasGemmBatchedEx()
. cublasGemmStridedBatch
. cublasCsyrrkEx()
. cublasCsyrrk3mEx()
. cublasCherkEx()
. cublasCherk3mEx()
. cublasNrm2Ex()
. cublasAxyEx()
. cublasDotEx()
. cublasScalEx()
```

cuBLAS

让我们一起来看一下实例

cuBLAS

```
cublasStatus_t cublasSgemm(cublasHandle_t handle, //句柄, 无含义
cublasOperation_t transa, //是否对A转置, 即是否更换优先方式行/列
cublasOperation_t transb, //是否对B转置, 即是否更换优先方式行/列
int m, int n, int k,
const float *alpha,
const float *A, int lda, //leading dimension of two-dimensional array
used to store the matrix A.
const float *B, int ldb, //leading dimension of two-dimensional array
used to store the matrix B.
const float *beta,
float *C, int ldc //leading dimension of two-dimensional array used
to store the matrix C.
)
```

m:这代表op(A)的行或是c的行;n, 这个代表的是op(B)的列或是C的列, 其中k, 代表的是op(A)的列或是op(B)的行;其中alpha和beta如上面的公式可见, beta是修正偏差, 只需要将alpha=1, beta=0, 即可;比较难以理解的是lda, ldb, ldc这三个参数。参考api表示两维矩阵的leading dimension(主维度)

cuBLAS

```
cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, m, n, k, &alpha, d_A, m, d_B, k, &\beta, d_C, m);
```

$$A = \begin{bmatrix} 0 & 5 \\ 1 & 6 \\ 2 & 7 \\ 3 & 8 \\ 4 & 9 \end{bmatrix}_{5 \times 2} * \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}_{2 \times 3} = \begin{bmatrix} 5 & 15 & 25 \\ 6 & 20 & 34 \\ 7 & 25 & 43 \\ 8 & 30 & 52 \\ 9 & 35 & 61 \end{bmatrix}_{5 \times 3}$$

cuBLAS

```
cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, m, n, K, &alpha, d_A, k, d_B, k, &\beta, d_C, m);
```

$$A = \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \\ 6 & 7 \\ 8 & 9 \end{bmatrix}_{5 \times 2} * \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}_{2 \times 3} = \begin{bmatrix} 1 & 3 & 5 \\ 3 & 13 & 23 \\ 5 & 23 & 41 \\ 7 & 33 & 59 \\ 9 & 43 & 77 \end{bmatrix}_{5 \times 3}$$

cuBLAS

```
cublasSetMatrix(int rows, int cols, int  
elemSize, const void *A, int lda, void *B, int  
ldb)
```

可以把CPU上的矩阵A拷贝到GPU上的矩阵B，两个矩阵都是列优先存储的格式。`lda`是A的leading dimension，既然是列优先，那么就是A的行数（A的一列有多少个元素）

```
cublasGetMatrix()
```

与cublasSetMatrix作用相反

cuBLAS

```
cublasCreate(cublasHandle_t *handle)
```

```
cublasDestroy(cublasHandle_t handle)
```

使用cuBLAS库函数，必须初始化一个句柄来管理cuBLAS上下文。函数为cublasCreate()，销毁的函数为cublasDestroy()。这些操作全部需要显式的定义，这样用户同时创建多个handle，绑定到不同的GPU上去，执行不同的运算，这样多个handle之间的计算工作就可以互不影响。

官方手册建议尽可能少的创建handle，并且在运行任何cuBLAS库函数之前创建好handle。handle会和当前的device（当前的GPU显卡）绑定，如果有多块GPU，你可以为每一块GPU创建一个handle。或者只有一个GPU，你也可以创建多个handle与之绑定。

```
cublasDeviceSynchronize()
```

cuBLAS

```
cublasSetVector(int n, int elemSize, const void *x, int incx,  
void *y, int incy)
```

这个函数也是把数据从CPU复制到GPU。CPU上包含n个元素的向量x，数据复制到GPU矩阵y上，每个元素的字节数为elemSize，incx是x的主维的长度，如果是列优先格式，那么就是x的行数，incy同理。

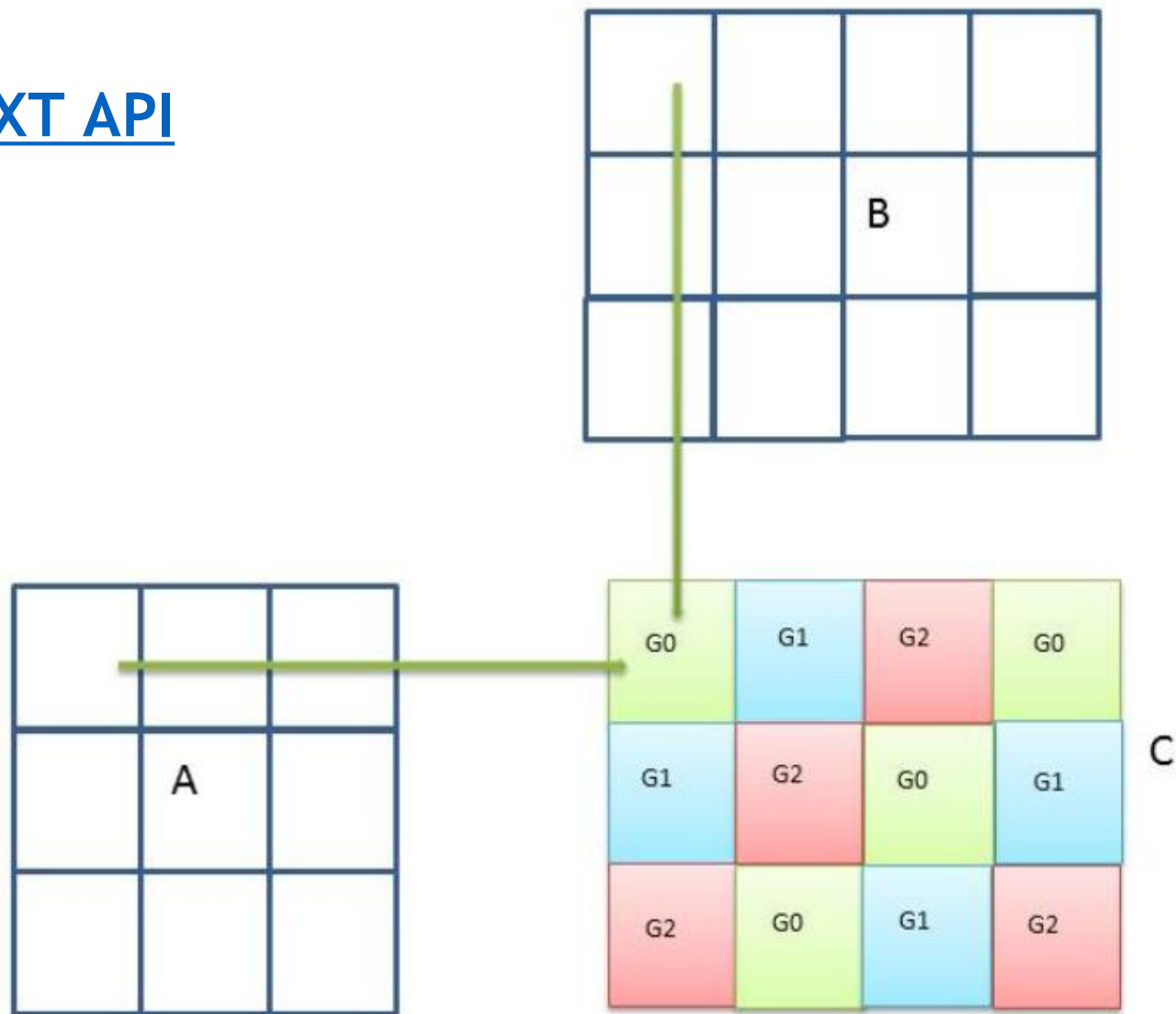
cuBLAS

CUBLASXT API

cuBLAS的cublasXt API公开了一个支持多gpu的主机接口:当使用这个API时, 应用程序只需要在主机内存空间上分配所需的矩阵

cuBLAS

CUBLASXT API



cuDNN

1. 在GPU上分配空间
2. 初始化句柄
3. 描述Tensor
4. 描述操作并设置相关参数
5. 描述算法
6. 申请工作空间
7. 将计算需要的数据传输到GPU
8. 开始计算
9. 将计算结果传回CPU内存
10. 释放资源

更多资源：

<https://developer.nvidia-china.com>



何琨-Ken

北京 密云



扫一扫上面的二维码图案，加我微信

<https://www.nvidia.cn/developer/community-training/>

