CUDA ON ARM PLATFORM—CUDA 库

NVIDIA企业级开发者社区 何琨

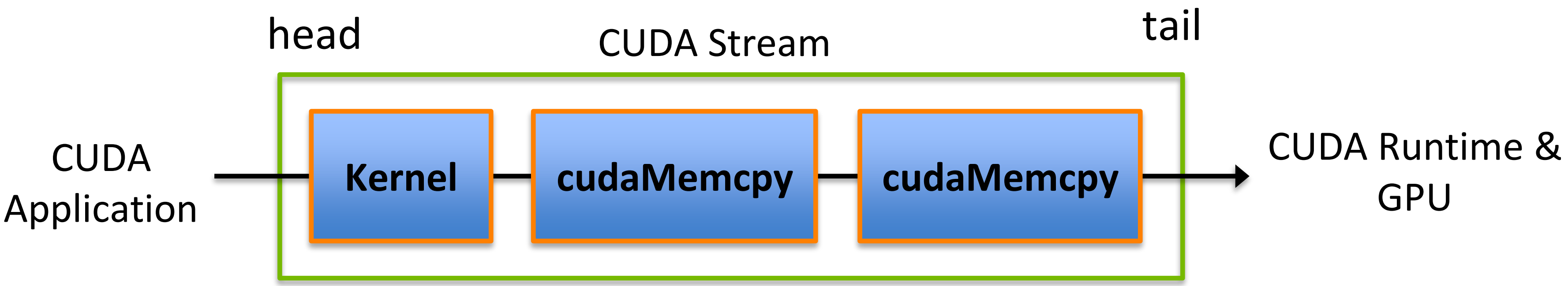
AGENDA

CUDA 库

- CUDA加速工具库
- 基本流程
- cuBLAS
- cuFFT

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

- Two types of streams in a CUDA program
  - The **implicitly** declared stream (**NULL stream**)
  - **Explicitly** declared streams (**non-NULL streams**)

- Up until now, all code has been using the NULL stream by default

```
cudaMemcpy(...);
kernel<<<...>>>(...);
cudaMemcpy(...);
```

- Non-NULL streams require manual allocation and management by the CUDA programmer

NVIDIA

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);

CUDA 流的概念

- Performing a cudaMemcpyAsync:

```
int *h_arr, *d_arr;
cudaStream_t stream;
cudaMalloc((void **)&d_arr, nbytes);
cudaMallocHost((void **)&h_arr, nbytes);
cudaStreamCreate(&stream);

cudaMemcpyAsync(d_arr, h_arr, nbytes, cudaMemcpyHostToDevice,
stream);
...
cudaStreamSynchronize(stream);
cudaFree(d_arr); cudaFreeHost(h_arr); cudaStreamDestroy(stream);
```

page-locked memory allocation

异步传输数据

Kernel

同步流

CUDA 流的概念

- Associate kernel launches with a non-NULL stream
  - Note that kernels are always asynchronous

```
kernel<<<nblocks, threads_per_block,
smem_size, stream>>>(...);
```

- The effects of `cudaMemcpyAsync` and kernel launching
  - Operations are put in the stream queue for execution
  - Actually operations may not happen yet
- Host-side timer to time those operations
  - Not the actual time of the operations

使用CUDA流来加速应用程序

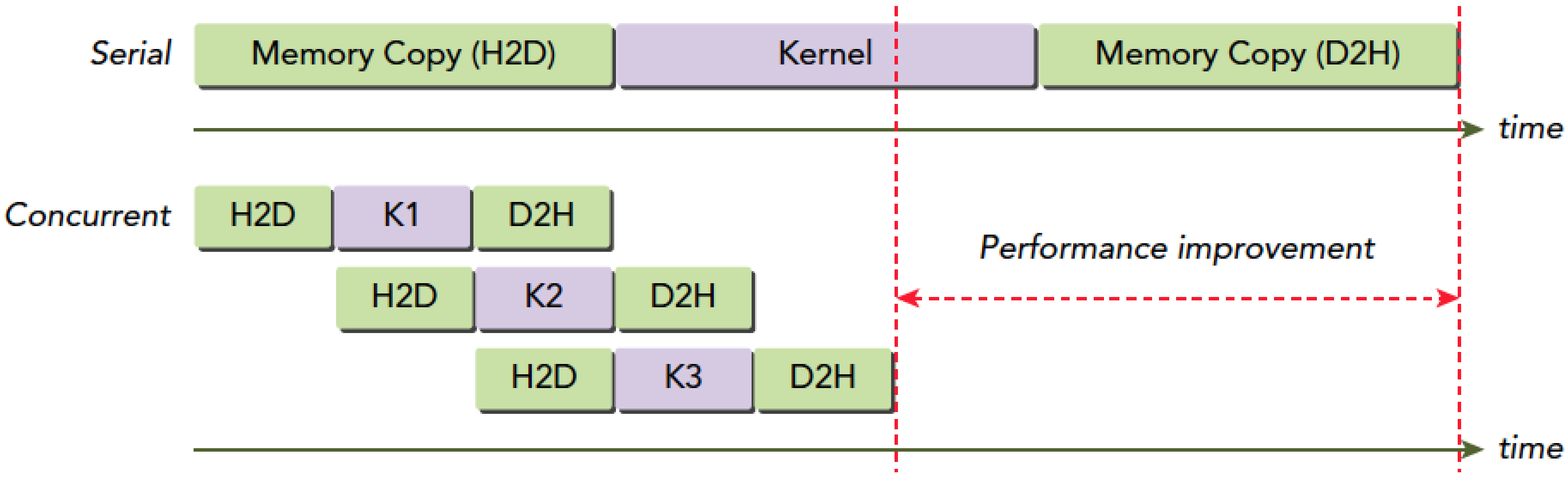
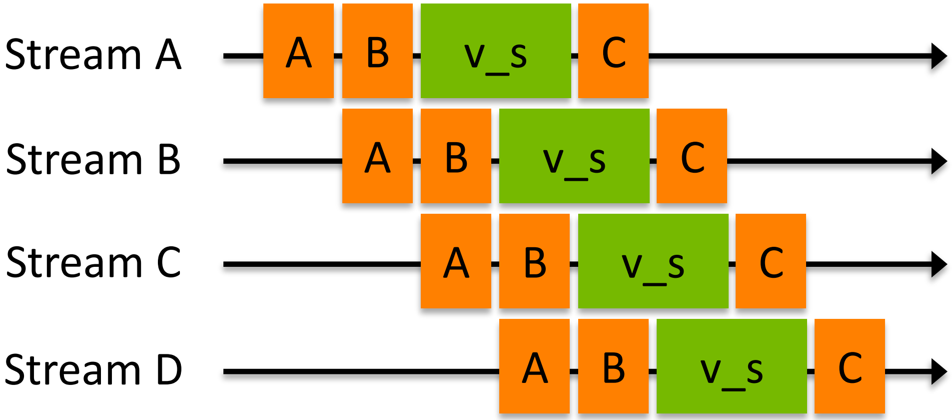


FIGURE 6-1

使用CUDA流来加速应用程序

- Vector sum example, A + B = C

- Partition the vectors and use CUDA streams to overlap copy and compute



使用CUDA流来加速应用程序

```
for (int i = 0; i < nstreams; i++) {
  int offset = i * eles_per_stream;
  cudaMemcpyAsync(&d_A[offset], &h_A[offset], eles_per_stream *
     sizeof(int), cudaMemcpyHostToDevice, streams[i]);
  cudaMemcpyAsync(&d_B[offset], &h_B[offset], eles_per_stream *
     sizeof(int), cudaMemcpyHostToDevice, streams[i]);
   ......
  vector_sum<<<..., streams[i]>>>(d_A + offset,
     d_B + offset, d_C + offset);
  cudaMemcpyAsync(&h_C[offset], &d_C[offset], eles_per_stream *
     sizeof(int), cudaMemcpyDeviceToHost, streams[i]);
}

for (int i = 0; i < nstreams; i++)
  cudaStreamSynchronize(streams[i]);
```

CUDA加速工具库

LIBRARY NAME	DOMAIN
NVIDIA cuFFT	Fast Fourier Transforms
NVIDIA cuBLAS	Linear Algebra (BLAS Library)
CULA Tools	Linear Algebra
MAGMA	Next-gen Linear Algebra
IMSL Fortran Numerical Library	Mathematics and Statistics
NVIDIA cuSPARSE	Sparse Linear Algebra
NVIDIA CUSP	Sparse Linear Algebra and Graph Computations
AccelerEyes ArrayFire	Mathematics, Signal and Image Processing, and Statistics
NVIDIA cuRAND	Random Number Generation
NVIDIA NPP	Image and Signal Processing
NVIDIA CUDA Math Library	Mathematics
Thrust	Parallel Algorithms and Data Structures
HiPLAR	Linear Algebra in R
Geometry Performance Primitives	Computational Geometry
Paralution	Sparse Iterative Methods
AmgX	Core Solvers

CUDA加速工具库---基本流程

1. Create a library-specific handle that manages contextual information useful for the library’s operation.
   - Many CUDA Libraries have the concept of a handle which stores opaque library-specific information on the host which many library functions access
   - Programmer’s responsibility to manage this handle
   - For example: cublasHandle_t, cufftHandle, cusparseHandle_t, curandGenerator_t

2. Allocate device memory for inputs and outputs to the library function.
   - Use cudaMalloc as usual

CUDA加速工具库---基本流程

3. If inputs are not already in a library-supported format, convert them to be accessible by the library.
   - Many CUDA Libraries only accept data in a specific format
   - For example: column-major vs. row-major arrays

4. Populate the pre-allocated device memory with inputs in a supported format.
   - In many cases, this step simply implies a cudaMemcpy or one of its variants to make the data accessible on the GPU
   - Some libraries provide custom transfer functions, for example: cublasSetVector optimizes strided copies for the CUBLAS library

CUDA加速工具库---基本流程

5. Configure the library computation to be executed.
   - In some libraries, this is a no-op
   - Others require additional metadata to execute library computation correctly
   - In some cases this configuration takes the form of extra parameters passed to library functions, others set fields in the library handle

6. Execute a library call that offloads the desired computation to the GPU.
   - No GPU-specific knowledge required

CUDA加速工具库---基本流程

7. Retrieve the results of that computation from device memory, possibly in a library-determined format.
   - Again, this may be as simple as a cudaMemcpy or require a library-specific function

8. If necessary, convert the retrieved data to the application’s native format.
   - If a conversion to a library-specific format was necessary, this step ensures the application can now use the calculated data
   - In general, it is best to keep the application format and library format the same, reducing overhead from repeated conversions

CUDA加速工具库---基本流程

9. Release CUDA resources.
   - Includes the usual CUDA cleanup (cudaFree, cudaStreamDestroy, etc) plus any library-specific cleanup

10. Continue with the remainder of the application.

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

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

[cublas<t>asum()]()

```
cublasStatus_t cublasSasum(cublasHandle_t handle, int n, const float *x, int incx, float *result)
cublasStatus_t cublasDasum(cublasHandle_t handle, int n, const double *x, int incx, double *result)
cublasStatus_t cublasScasum(cublasHandle_t handle, int n, const cuComplex *x, int incx, float *result)
cublasStatus_t cublasDzasum(cublasHandle_t handle, int n, const cuDoubleComplex *x, int incx, double *result)
```

- cublasI<t>amax()
- cublasI<t>amin()
- cublas<t>asum()
- cublas<t>axpy()
- cublas<t>copy()
- cublas<t>dot()
- cublas<t>nrm2()
- cublas<t>rot()
- cublas<t>rotg()
- cublas<t>rotm()
- cublas<t>rotmg()
- cublas<t>scal()
- cublas<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

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

让我们一起来看一下实例

cuBLAS Example

- Matrix-vector multiplication
  - Uses 6 of the 10 steps in the common library workflow:

1. Create a cuBLAS handle using **`cublasCreateHandle`**
2. Allocate device memory for inputs and outputs using **`cudaMalloc`**
3. Populate device memory using **`cublasSetVector`**, **`cublasSetMatrix`**
4. Call **`cublasSgemv`** to run matrix-vector multiplication on the GPU
5. Retrieve results from the GPU using **`cublasGetVector`**
6. Release CUDA and cuBLAS resources using **`cudaFree`**, **`cublasDestroy`**

cuBLAS Example

```
cublasCreate(&handle);
cudaMalloc((void **)&dA, sizeof(float) * M * N);
cudaMalloc((void **)&dX, sizeof(float) * N);
cudaMalloc((void **)&dY, sizeof(float) * M);

cublasSetVector(N, sizeof(float), X, 1, dX, 1);
cublasSetVector(M, sizeof(float), Y, 1, dY, 1);
cublasSetMatrix(M, N, sizeof(float), A, M, dA, M);

cublasSgemv(handle, CUBLAS_OP_N, M, N, &alpha, dA, M, dX, 1,
&beta, dY, 1);

cublasGetVector(M, sizeof(float), dY, 1, Y, 1);

/* for sgemm */
```

cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, matrix_size.uiWB, matrix_size.uiHA, matrix_size.uiWA, &alpha, d_B, matrix_size.uiWB, d_A, matrix_size.uiWA, &beta, d_C, matrix_size.uiWA)

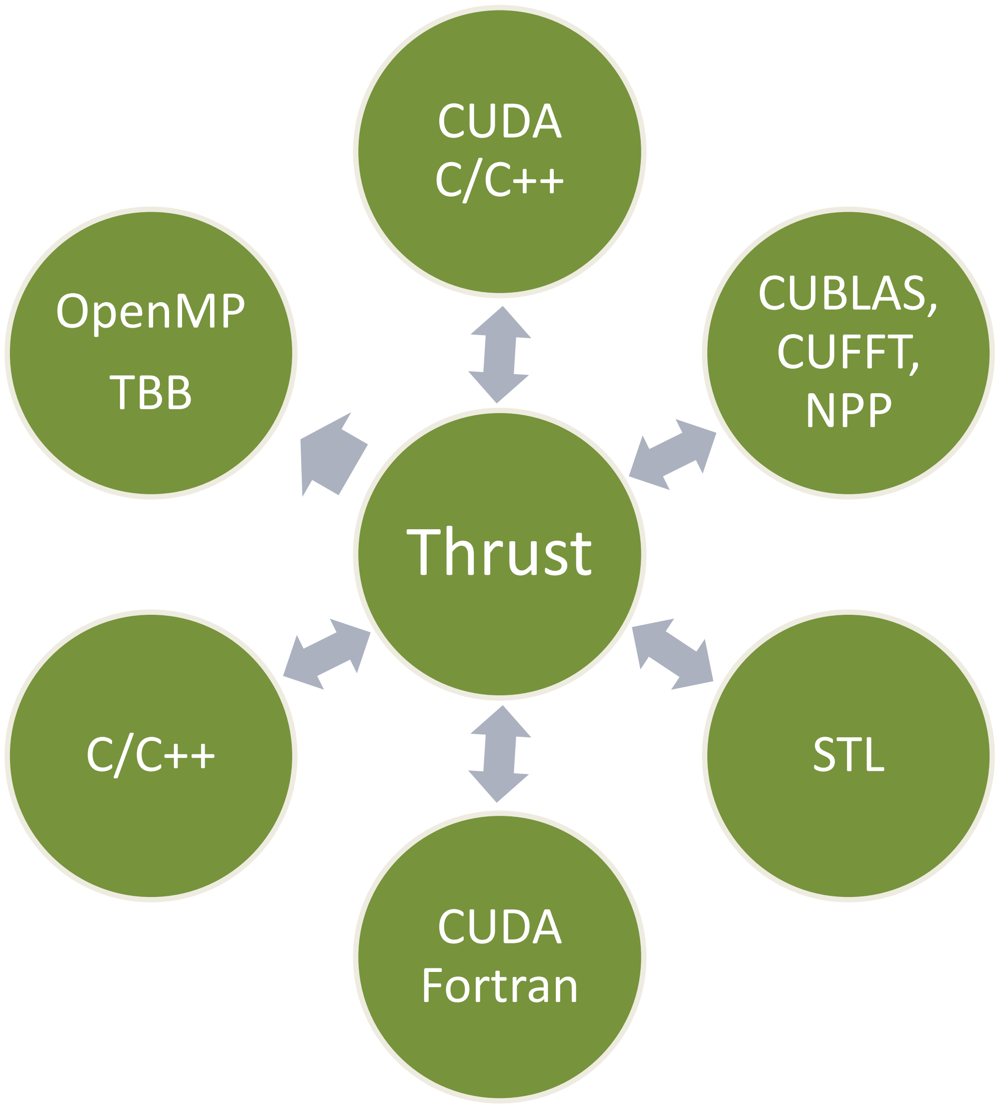
Thrust

- Thrust 是基于标准模板库 (STL) 的 CUDA 的 C++ 模板库。
- Thrust 允许您通过与 CUDA C 完全互操作的高级接口，以最少的编程工作实现高性能并行应用程序。
- Thrust 提供了丰富的数据并行原语集合，例如扫描、排序和归约，它们可以组合在一起，以简洁、可读的源代码实现复杂的算法。
- 通过用这些高级抽象描述您的计算，您可以让 Thrust 自由地自动选择最有效的实现。 因此，Thrust 可用于 CUDA 应用程序的快速原型设计（其中程序员的生产力最为重要），也可用于生产（其中稳健性和绝对性能至关重要）。

Thrust

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/sort.h>
#include <cstdlib>

int main(void)
{
    // generate 32M random numbers on the host
    thrust::host_vector<int> h_vec(32 << 20);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

    // transfer data to the device
    thrust::device_vector<int> d_vec = h_vec;

    // sort data on the device
    thrust::sort(d_vec.begin(), d_vec.end());

    // transfer data back to host
    thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

    return 0;
}
```

Thrust


CUDA C/C++
OpenMP
TBB
CUBLAS, CUFFT, NPP
Thrust
C/C++
STL
CUDA Fortran

Thrust

- Containers

  `host_vector`

  `device_vector`

- Memory Mangement
  - Allocation
  - Transfers
- Algorithm Selection
  - Location is implicit

```
// allocate host vector with two elements
thrust::host_vector<int> h_vec(2);

// copy host data to device memory
thrust::device_vector<int> d_vec = h_vec;

// write device values from the host
d_vec[0] = 27;
d_vec[1] = 13;

// read device values from the host
int sum = d_vec[0] + d_vec[1];

// invoke algorithm on device
thrust::sort(d_vec.begin(), d_vec.end());

// memory automatically released
```

Thrust

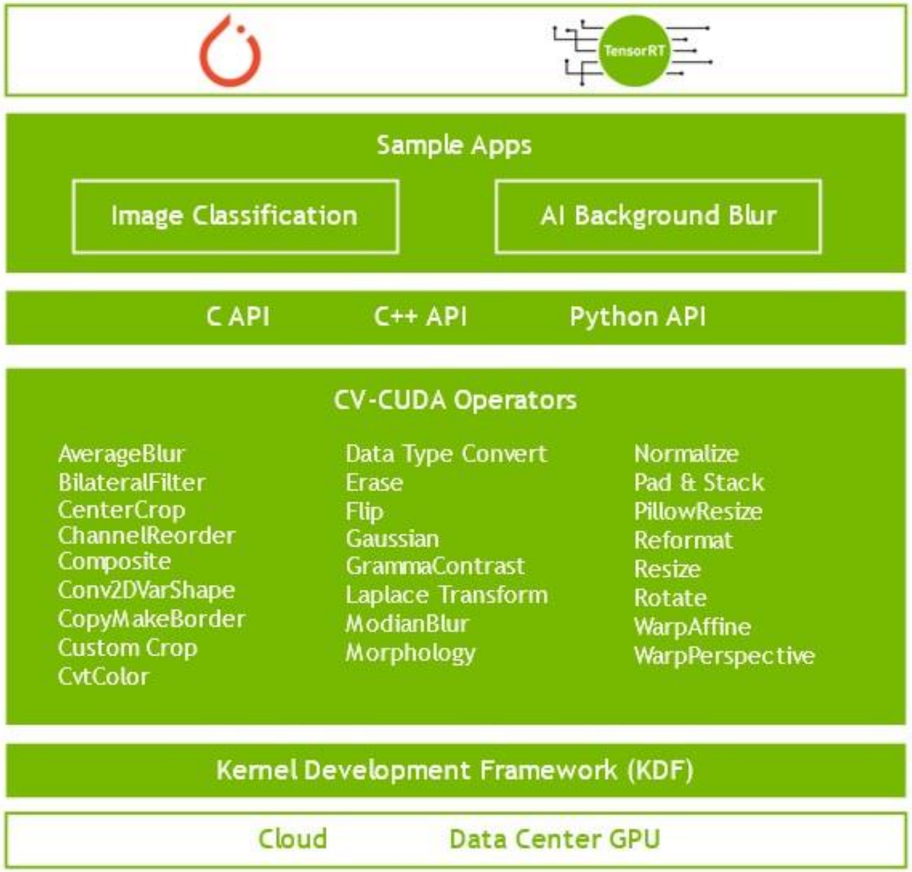
- Large set of algorithms
  - ~75 functions
  - ~125 variations
- Flexible
  - User-defined types
  - User-defined operators

Algorithm	Description
`reduce`	Sum of a sequence
`find`	First position of a value in a sequence
`mismatch`	First position where two sequences differ
`inner_product`	Dot product of two sequences
`equal`	Whether two sequences are equal
`min_element`	Position of the smallest value
`count`	Number of instances of a value
`is_sorted`	Whether sequence is in sorted order
`transform_reduce`	Sum of transformed sequence

CV-CUDA

- CV-CUDA 是一个开源项目，使开发人员能够在云规模的人工智能 (AI) 成像和计算机视觉 (CV) 工作负载中构建高效、GPU 加速的预处理和后处理管道。
- 借助一组针对数据中心 GPU 性能进行手动优化的专用 CV 和图像处理内核，CV-CUDA 可确保使用这些内核构建的处理管道得到执行，从而在整个复杂工作负载中提供更高的吞吐量。
- CV-CUDA 可以提供超过 10 倍的吞吐量改进和更低的云计算成本。 CV-CUDA 将提供与 C/C++、Python 的轻松集成，以及与 PyTorch 等常见深度学习 (DL) 框架的接口。

CV-CUDA


TensorRT
Sample Apps
Image Classification
AI Background Blur
C API
C++ API
Python API
CV-CUDA Operators
AverageBlur
BilateralFilter
CenterCrop
ChannelReorder
Composite
Conv2DVarShape
CopyMakeBorder
Custom Crop
CvtColor
Data Type Convert
Erase
Flip
Gaussian
GrammaContrast
Laplace Transform
ModianBlur
Morphology
Normalize
Pad & Stack
PillowResize
Reformat
Resize
Rotate
WarpAffine
WarpPerspective
Kernel Development Framework (KDF)
Cloud
Data Center GPU

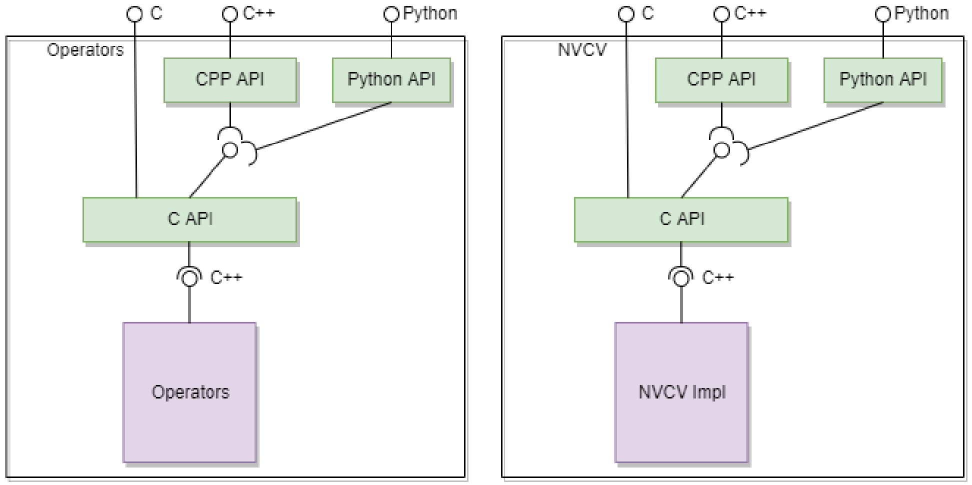
CV-CUDA

Key Features:

- A unified, specialized set of highly performant CV kernels
- C, C++, and Python APIs
- Kernel development framework
- Batching support, with variable shape images
- Zero-copy interfaces to PyTorch and TensorFlow
- End-to-end reference samples

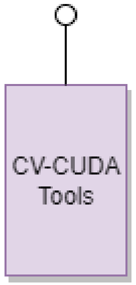
Public APIs

Two types of use cases for public APIs



- Using Operators
  - C, C++, Python APIs exposed.
    - Operator API
    - NVCV API
- Creating Operators

  C++ header only, API will be used on host/device
  - CV-CUDA tools API



CV-Toolkit

NVIDIA offers a number of products for accelerating computer vision and image processing applications. In addition to CV-CUDA, some of the others include:

- DALI (Data Loading Library), a portable, holistic framework for accelerated data loading and augmentation in deep learning workflows involving images, videos, and audio data.
- VPI (Vision Programming Interface), an accelerated computer vision and image processing software library primarily for embedded/edge applications.
- cuCIM (Compute Unified Device Architecture Clara Image), an open source, accelerated computer vision and image processing library for multidimensional images in biomedical, geospatial, material life science, and remote sensing use cases.
- NPP (NVIDIA Performance Primitives), an image, signal, and video processing library that accelerates and performs domain-specific functions.

NVIDIA cuNumeric

将 GPU 加速的超级计算引入 NumPy 生态系统Python 已成为数据科学、机器学习和高效数值计算中使用最广泛的语言。

NumPy 是事实上的标准数学和矩阵库，提供简单易用的编程模型，其接口与科学应用的数学需求密切相关，使其成为许多最广泛使用的数据科学和机器学习的基础 构建学习编程环境。

随着数据集规模的不断扩大和程序的复杂性不断增加，越来越需要通过利用远远超出单个 CPU 节点所能提供的计算资源来解决这些问题。

NVIDIA TensorRT

高性能深度学习推理SDK

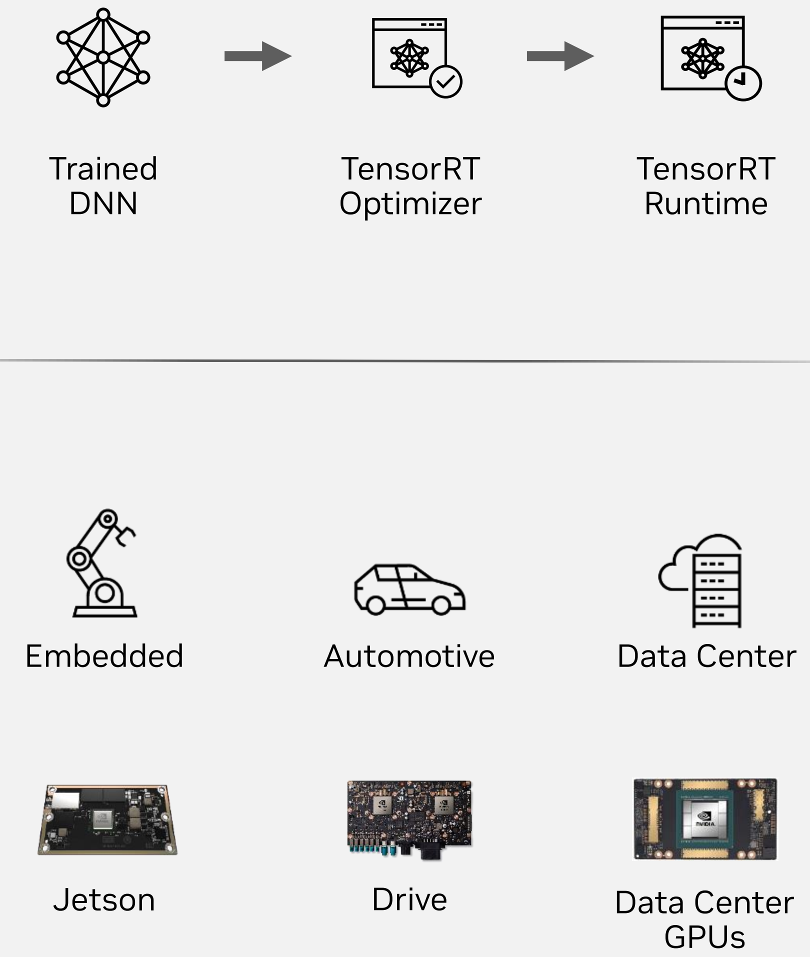
优化和部署产品级的神经网络

利用编译器和运行时最大化吞吐量

优化各种网络模型，包括CNN， RNN和Transformers

1. 多种精度数据类型: FP32, TF32, FP16, and INT8.
2. 网络层的融合: 优化GPU存储带宽的使用率.
3. 内核调整: 选择适合您要部署平台的最佳算法.
4. 动态Tensor memory: 部署高内存效率的应用.
5. 多流执行可以轻松扩展处理多个执行流.
6. 时间序列融合: 优化RNN.

https://developer.nvidia.com/tensorrt



TensorRT Integrated with Pytorch and Tensorflow

Up to 4X faster inference with 1 line of code

Torch-TensorRT

PyTorch

```python
import torch
import torch_tensorrt as torchtrt

# SET trained model to evaluation mode
model = model.eval()

# COMPILE TRT module using Torch-TensorRT
trt_module = torchtrt.compile(model,
inputs=[example_input],enabled_precisions={torch.half})

# RUN optimized inference with Torch-TensorRT
trt_module(x)
```

Available in PyTorch & NGC Container

TensorFlow-TensorRT

TensorFlow

```python
import tensorflow as tf
from tf.python.compiler.tensorrt import trt_convert as tftrt

# COMPILE TRT module using TensorFlow-TensorRT
trt_module =
tftrt.TrtGraphConverterV2(saved_model_pth).convert()

# RUN optimized inference with TensorFlow-TensorRT
trt_module(x)
```

Available in TensorFlow & NGC Container

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/

NVIDIA