cuDNN

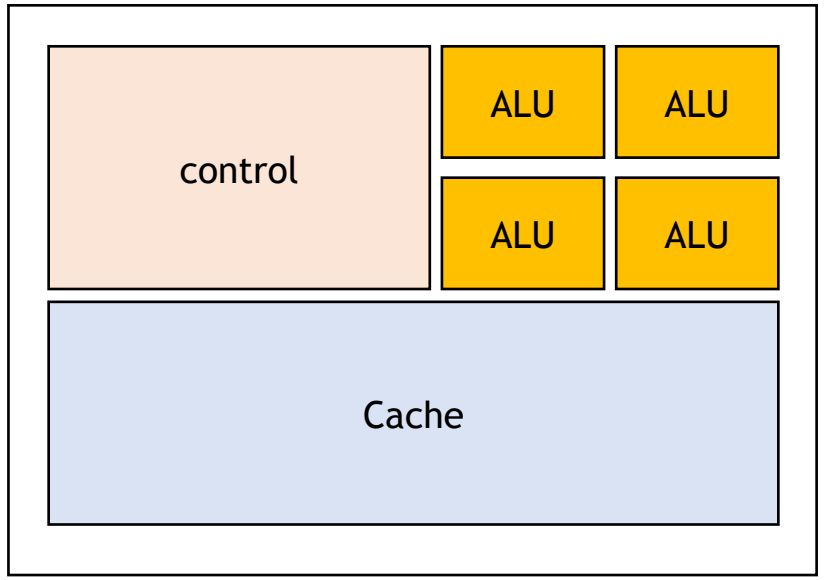
https://youtu.be/Y4Kyb7SDR18

Contents

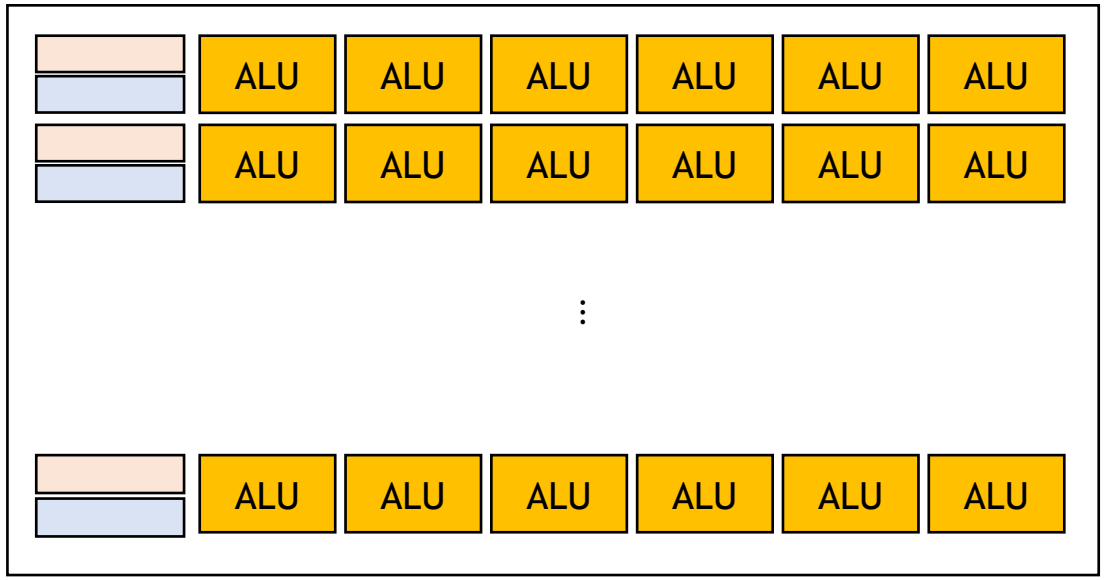
CryptoCraft LAB

Graphics Processing Unit (GPU)

- 3D 그래픽 처리 위한 하드웨어
- General-Purpose Computation on GPU (GPGPU) : 2006 ~ 현재
  - 그래픽 뿐만 아니라 범용 연산 분야에서 활용



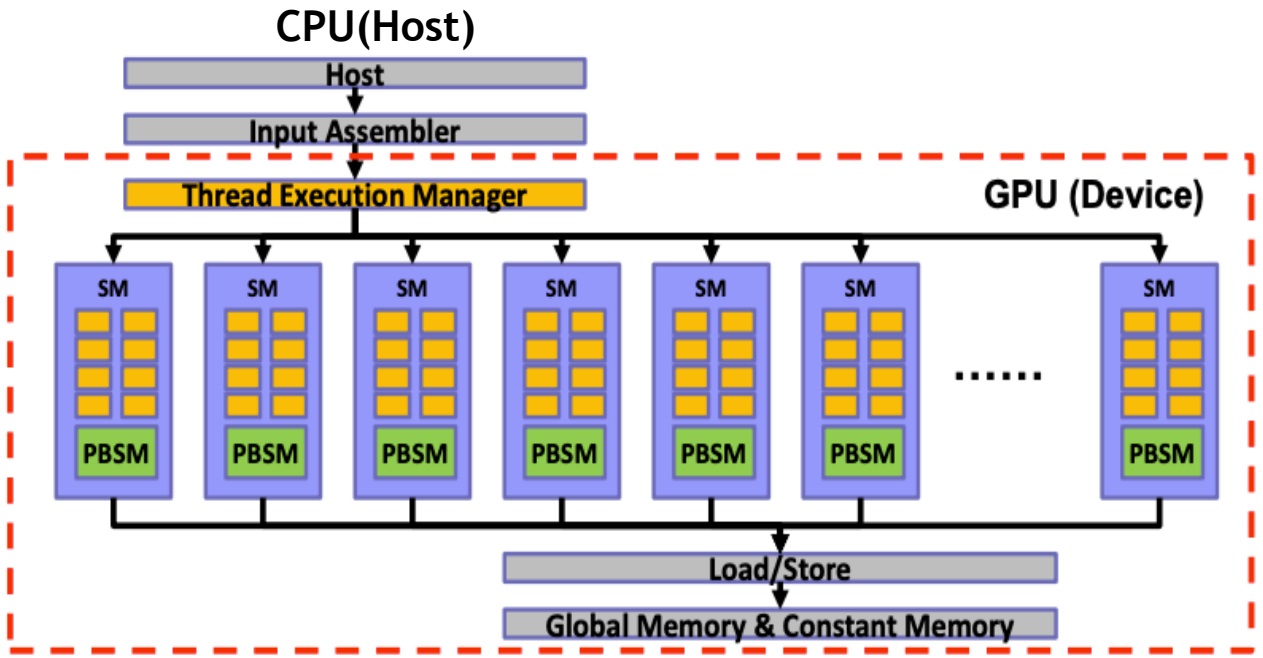
CPU

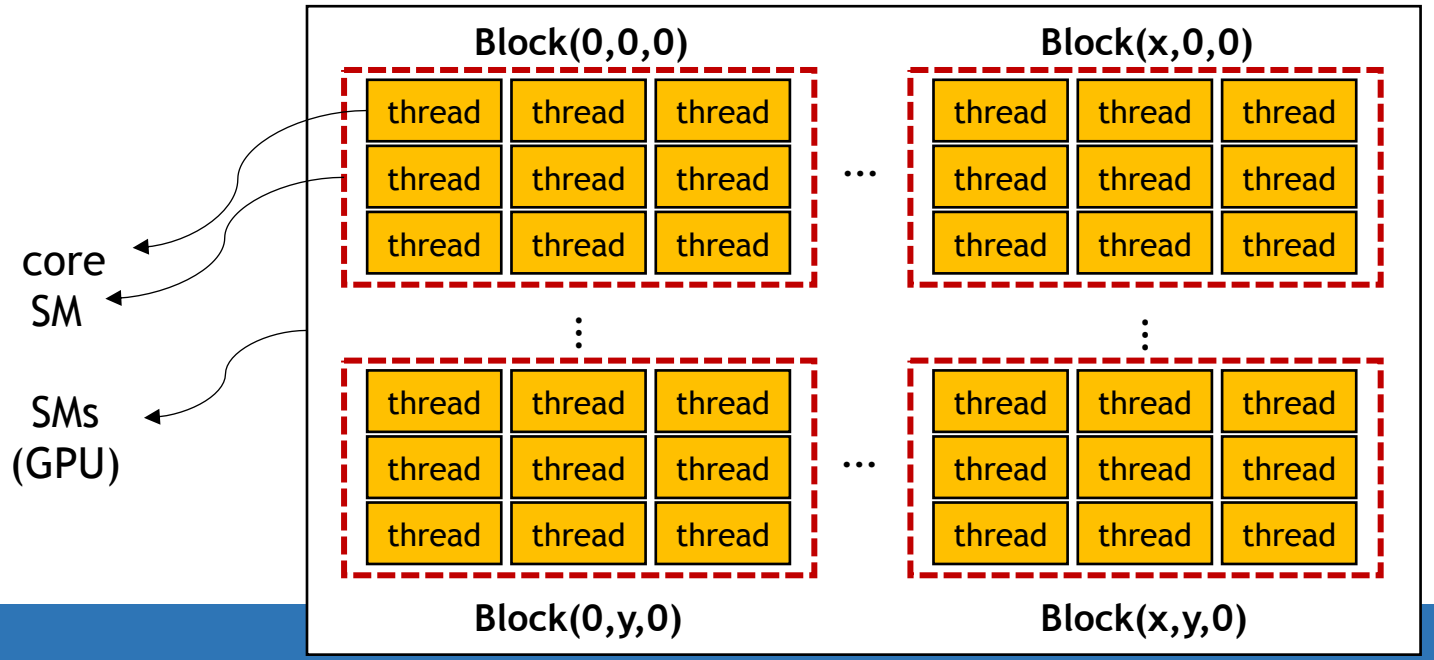


GPU

GPU architecture

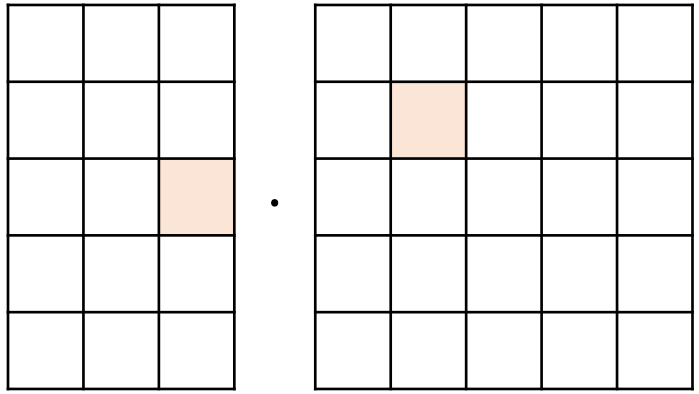
- Streaming Multi-processor
- Streaming Processor (core)
- thread → block → grid
  - thread → core가 수행
  - block → Streaming Multiprocessor
  - grid → SMs
- Shared Memory (per block share memory, PBSM
- Global Memory





CPU vs GPU vs TPU

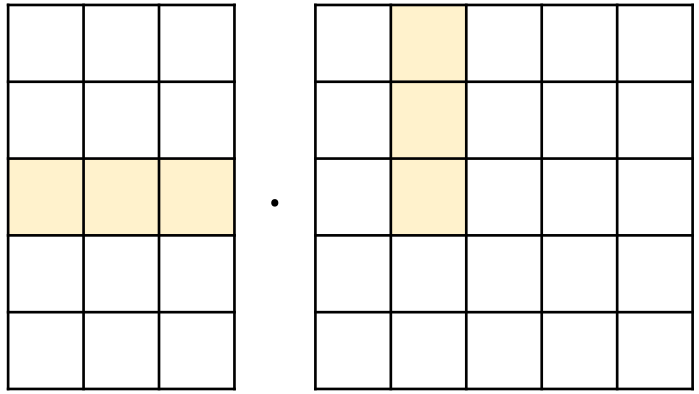
CPU



scalar

- scalar 단위 연산 → 느림

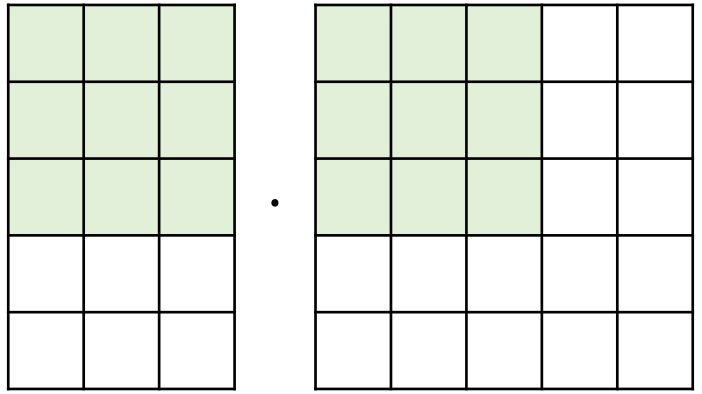
GPU



vector

- vector 단위 연산 → TPU보다 느림
- **TPU에 비해 유연** (data shape에 의한 영향 x)

TPU



tensor

- tensor 단위 연산 → 속도 빠름
- data와 weight matrix의 형태에 영향

  → 최적화하지 않을 경우 ,

  이 과정에서 시간이 더 소요되기도 함

Deep learning + GPU

- Neural Network main operation

  → floating point multiplication

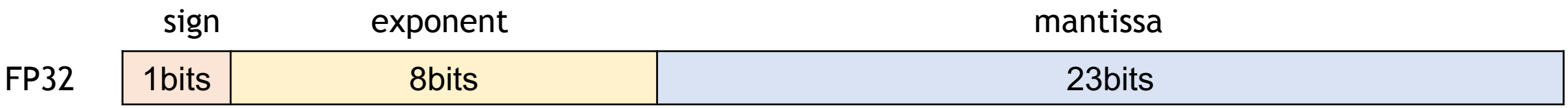
  matrix의 각 요소를 순차적 연산할 필요 없음 (**병렬 수행 가능한 연산**)

- Low control overhead

  → Single Instruction Multiple Threads

  GPU 통해 각 thread들이 **하나의 연산을 동시에 수행**

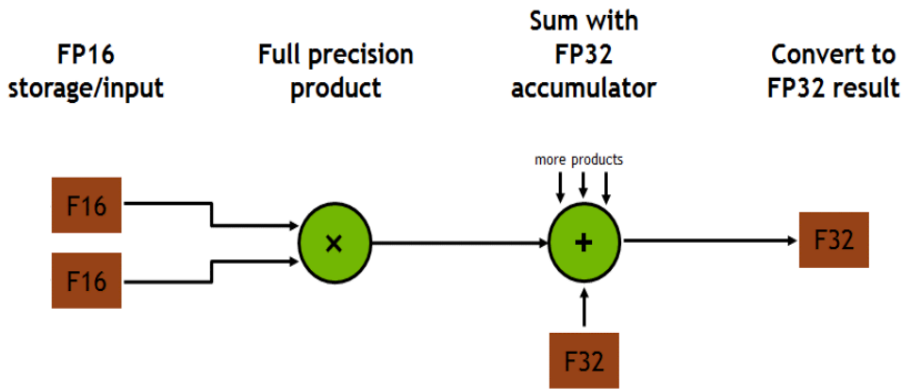
FP16 vs FP32



- 메모리 사용량 2배 감소 및 처리량 8배 증가
- 표현 범위 감소
  → back propagation과정에서 기울기 값들이 FP16으로는 표현 불가한 범위에 존재
  → 너무 크거나 너무 작은 경우
  → weight의 오차 발생 → 오차 누적→ loss 증가

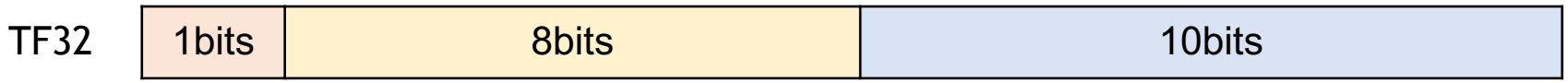
- 연산 속도 증가, 계산 정확도 감소

*mixed precision
: FP16 input & operation → 연산 속도 증가
loss scaling → 기울기에 큰 값 곱해 표현 가능한 범위로 scaling → FP32 output



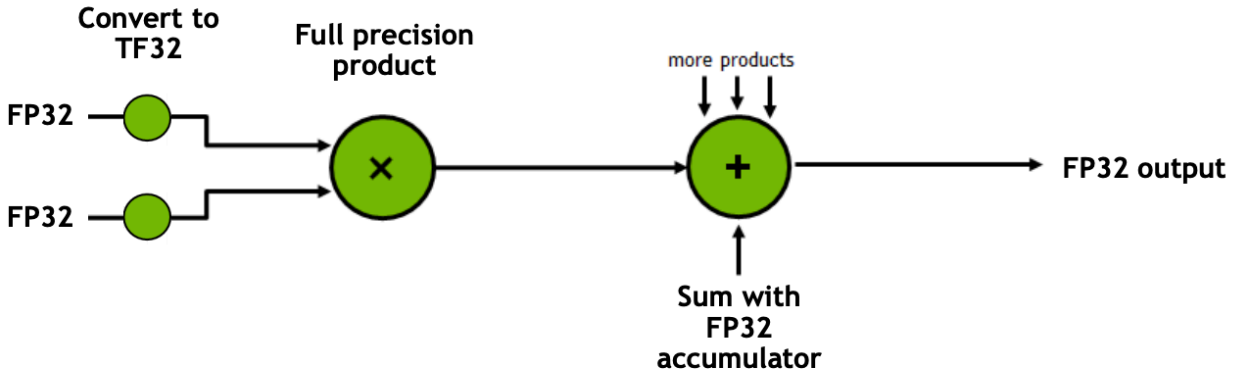
Tensor core – TF32

- **matrix-multiply-and-accumulate**
- TF32 → tensor core mode (not a type)
  - convolution, matrix multiplication 할 때만 TF32로 변환



  - 8bits exponent → FP32와 동일한 범위 → 오차 감소
  - 10bits mantissa → FP32보다는 정밀도 떨어지지만 어느정도 보장
  - **FP32에 비해 처리량 16배 증가 but 계산 정확도 희생 필요**

- tensor core 작동 순서
  1. FP32 input
  2. Tensor core mode위해 TF32로 변환
  3. TF32 행렬 곱셈
  4. FP32 누적



Tensor core

- GPU → **32 threads**씩 동일한 작업
  - row,col → 8 배수 matrix 사용 시 최적화 (16x16, 32x8 …)
- Tesla V100 tensor core
  - SM 당 8 tensor core
- Turing tensor core
  - **inference위해 INT8, INT4 연산 추가**
  - quantization 기법 통해 연산에 필요한 **비트 수 감소 → 속도 증가**
- Ampere (A100) tensor core – CUDA8
  - V100 이상의 성능 (2.5 ~ 5배 (sparse model))
  - sparse model 지원

    → 0 제거하여 희소 행렬 연산 → 연산량 감소

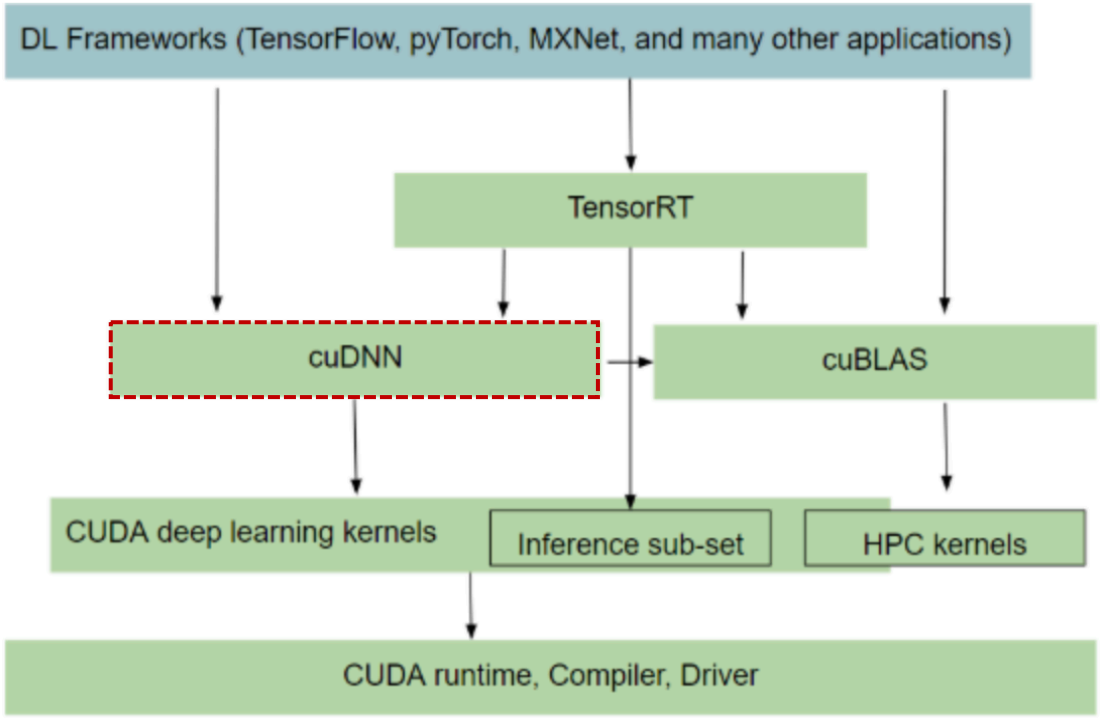
    → 차원 수 줄여서 오버피팅 방지

cuDNN

- CUDA를 활용한 신경망 구성 및 학습의 **가속화를 위한 라이브러리**
- convolution, pooling, normalization, activation(ReLu, Tanh, Sigmoid …) 등의 forward, backward 지원
- FP32, FP16 및 TF32 부동 소수점 형식과 INT8 및 UINT8 정수 형식(inference) 지원
  - 핵심 연산 가속화 위해 **TF32**를 기본 값으로 채택
- 지원하는 framework

Caffe Caffe2 Chainer Microsoft Cognitive Toolkit MATLAB mxnet
PaddlePaddle PyTorch TensorFlow torch Wolfram Language

***tensor RT**
:GPU 상에서 최대 **추론** 처리량과 효율성을 제공하도록 설계된 고성능 추론 엔진 → 실시간 추론 (자율 주행 등)



cuDNN

- 버전에 맞는 CUDA 및 cuDNN 설치 필요
  - GPU마다 설치 가능한 버전이 다를 수 있음

    → compute compatibility 확인

버전	Python 버전	컴파일러	빌드 도구	cuDNN	CUDA
tensorflow_gpu-1.13.1	2.7, 3.3-3.6	GCC 4.8	Bazel 0.19.2	7.4	10.0
tensorflow_gpu-1.12.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.2.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.4.5	5.1	8
tensorflow_gpu-1.1.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.4.2	5.1	8
tensorflow_gpu-1.0.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.4.2	5.1	8

Supported NVIDIA Hardware	CUDA Version	CUDA Compute Capability	CUDA Driver Version
▸ NVIDIA Ampere GPU architecture ▸ Turing ▸ Volta ▸ Pascal ▸ Maxwell ▸ Kepler	CUDA 11.1	SM 3.5 and later	r450, r455

Architecture	OS Name	OS Version	Distro Information		
			Kernel	GCC	Glibc
x86_64	RHEL	7.8	3.10.0	4.8.5	2.17
		8.2[1]	4.18	8.3.1	2.28
	Ubuntu	18.04.4 LTS	4.15.0	8.2.0	2.27
		16.04.6 LTS	4.5.0	5.4.0	2.23

CUDA process flow

- **GPU memory 할당**

  cudaMalloc((void**) &d_A, size);

- **CPU memory의 데이터를 GPU memory로 복사**

  cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);

- **kernel launching**

  Kernel<<<blocks,threads>>>(parameters);

- **연산 결과를 다시 CPU memory로 복사**

  cudaMemcpy(h_A, d_A, size, cudaMemcpyDeviceToHost);

- **GPU memory 해제**

  cudaFree(d_A);

cuDNN process flow

- **CUDA process와 비슷하게 진행**
  1. GPU memory 할당
  2. CPU memory data를 GPU memory 복사
  3. cuDNN 사용위한 핸들러 생성
  4. data, filter, convolution descriptor 선언
  5. data shape & type, convolution filter & stride 등 설정
  6. convolution forward algorithm 설정
  7. convolution에 필요한 메모리 공간 있으면 할당
  8. convolution 실행
  9. memory 해제
- **descriptor**
  - data, filter 등의 정보를 저장하는 구조체
  - tensor descriptor : layer의 feature map 형태 저장 → input, output
  - filter descriptor : filter (weight) 형태 저장
  - convolution descriptor : padding, stride, filter 정보 저장
  - convolution 함수 호출 시, tensor descriptor (input/output), filter, convolution descriptor 사용

cuDNN process flow

- **GPU 메모리 할당 & data 복사 (CPU to GPU)**

  cudaMalloc((void**) &d_A, size);

  cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);

  cudaMemcpy(devPtrF, devPtrF_h, size, cudaMemcpyHostToDevice);

- **cuDNN 사용 위한 핸들러 설정**

  cudnnCreate (& handle_)

- **descriptor 생성**

  cudnnCreateTensorDescriptor (& cudnnIdesc)

  cudnnCreateTensorDescriptor (& cudnnOdesc)

  cudnnCreateFilterDescriptor (& cudnnFdesc)

  cudnnCreateConvolutionDescriptor (& cudnnConvDesc)

5.2.36. cudnnSetConvolutionNdDescriptor()

```
cudnnStatus_t cudnnSetConvolutionNdDescriptor(
    cudnnConvolutionDescriptor_t    convDesc,
    int                             arrayLength,
    const int                       padA[],
    const int                       filterStrideA[],
    const int                       dilationA[],
    cudnnConvolutionMode_t          mode,
    cudnnDataType_t                 dataType)
```

cuDNN api

cuDNN process flow

- **descriptor 설정 (각 변수들은 미리 설정)**

  cudnnSetConvolutionNdDescriptor (cudnnConvDesc, convDim, padA, convstrideA,
  dilationA, CUDNN_CONVOLUTION, CUDNN_DATA_FLOAT);

- **사용할 알고리즘 선택**

  cudnnConvolutionFwdAlgo_t algo = **CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM**;

- **연산에 필요한 메모리 공간 있다면 할당**

  cudnnGetConvolutionForwardWorkspaceSize (handle_, cudnnIdesc, cudnnFdesc,
  cudnnConvDesc, cudnnOdesc, algo, & workSpaceSize));

  if (workSpaceSize> 0) { cudaMalloc (& workSpace, workSpaceSize); }

- **선택한 알고리즘 실행 (convolution 수행)**

  cudnnConvolutionForward (handle_, (void *) (& alpha), cudnnIdesc, devPtrI, cudnnFdesc, devPtrF, cudnnConvDesc,
  algo, workSpace, workSpaceSize, (void *) (& beta), cudnnOdesc, devPtrO));

기능
`cudnnRNNForwardInference`
`cudnnRNNForwardTraining`
`cudnnRNNBackwardData`

cuDNN process flow

- **memory 해제**

  cudnnSetTensorNdDescriptor (cudnnIdesc, CUDNN_DATA_FLOAT, convDim + 2, dimA, strideA);

  cudnnSetConvolutionNdDescriptor (cudnnConvDesc, convDim, padA, convstrideA,
  dilationA, CUDNN_CONVOLUTION, CUDNN_DATA_FLOAT);

  cudnnDestroyTensorDescriptor (& cudnnIdesc);

  cudnnDestroyTensorDescriptor (& cudnnOdesc);

  cudnnDestroyFilterDescriptor (& cudnnFdesc);

  cudnnDestroyConvolutionDescriptor (& cudnnConvDesc);

  cudnnDestroy(& handle_);

  cudaFree(d_A);

  cudaFree(h_A);

  cudaFree(devPtrF);

Q & A